TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Operaciones transaccionales.
- La instrucción AUTOCOMMIT.
- Captura de errores en Node.

OPERACIONES TRANSACCIONALES

Ejecutar múltiples operaciones transaccionales con instrucción COMMIT

En el siguiente ejemplo, se ejecutarán transacciones múltiples, las cuales contemplan la creación de una tabla en una base de datos, la inserción de valores en dicha tabla, y finalmente, guardarla cada una en la base de datos, utilizando una declaración final.

```
Begin;

CREATE TABLE Emp_Test1 (emp_id INT NOT NULL, emp_name character(10)
NOT NULL, emp_address character(20) NOT NULL, emp_phone
character(14), emp_salary INT NOT NULL, date_of_joining date NOT
NULL);

INSERT INTO Emp_Test1 (emp_id, emp_name, emp_address, emp_phone,
emp_salary, date_of_joining) VALUES (1, 'ABC', 'Pune', '1234567890',
20000, '01-01-2020');

INSERT INTO Emp_Test1 (emp_id, emp_name, emp_address, emp_phone,
emp_salary, date_of_joining) VALUES (2, 'PQR', 'Pune', '1234567890',
20000, '01-01-2020');

INSERT INTO Emp_Test1 (emp_id, emp_name, emp_address, emp_phone,
emp_salary, date_of_joining) VALUES (3, 'XYZ', 'Mumbai',
'1234567890', 35000, '02-01-2020');

End transaction;
Commit;
```

EJECUTAR MÚLTIPLES OPERACIONES TRANSACCIONALES CON INSTRUCCIÓN ROLLBACK

En el ejemplo a continuación, también se ejecutarán operaciones múltiples, esta vez considerando la instrucción ROLLBACK.

Consistirá en agregar un nuevo libro a la tabla Books, en el cual se contempla la condición de conocer si dicho libro ya existe en la tabla, y en base a ello, se ejecuta o no la inserción.

```
DECLARE @BookCount int
Begin AddBook;

INSERT INTO Books VALUES (20, 'Book15', 'Cat5', 2000);

SELECT @BookCount = count(*) FROM Books WHERE name ='Book15';

IF @BookCount > 1
        BEGIN
                ROLLBACK AddBook
                print 'Un libro con el mismo nombre ya existe'
        END
ELSE
        BEGIN
                COMMIT AddBook
                print 'El nuevo libro ha sido agregado
satisfactoriamente'
        END

End transaction;
```

LA INSTRUCCIÓN AUTOCOMMIT

Algunos Sistemas Gestores de Bases de Datos, como PostgreSQL, tienen activada por defecto la variable AUTOCOMMIT. Esto quiere decir que automáticamente se aceptan todos los cambios realizados después de la ejecución de una sentencia SQL (INSERT, UPDATE, DELETE), y luego no es posible deshacerlos. Por lo tanto, la sentencia COMMIT en este caso es innecesaria.

```
SET AUTOCOMMIT=true;
-- o
SET AUTOCOMMIT=1;
-- obtener el estado actual del  autocommit:
SELECT @@autocommit;
```

CAPTURA DE ERRORES EN NODE

Los errores en Node.js se manejan mediante excepciones. Para ello, se crea una excepción utilizando la palabra clave throw.

```
throw value
```

Tan pronto como JavaScript ejecuta esta línea, el flujo normal del programa se detiene, y el control se retiene al manejador de excepciones más cercano. Generalmente, en el código del lado del cliente, value puede ser cualquier valor de JavaScript, incluida una cadena, un número, o un objeto.

En Node.js no se lanzan cadenas, solo se lanzan objetos Error.

Objetos de error: es una instancia del objeto Error, o extiende la clase Error.

```
throw new Error('se quedó sin café')
```

PROCESANDO ERRORES Y CONDICIONES PARA LA OPERACIÓN DE TRANSACCIONES EN NODE

Manejo de excepciones: un controlador de excepciones es una declaración try/catch. Cualquier excepción planteada en las líneas de código, incluidas en el bloque try, se maneja en el correspondiente catch:

```
try {
  //lineas de código
} catch (e) {
}
```

Siendo en este ejemplo, el valor de excepción. También se pueden agregar varios controladores, que detectan diferentes tipos de errores.

Excepciones no detectadas: si se lanza una excepción no detectada durante la ejecución de su programa, éste se bloqueará. Para resolverlo, escuche el evento uncaughtException, en el objeto process:

```
process.on('uncaughtException', (err) => {
    console.error('Hubo un error no detectado', err)
    process.exit(1) //Obligatoria (según los documentos del nodo)
})
```

Para esto, no es necesario importar el módulo principal process, ya que se inyecta automáticamente.

Excepciones con promesas (promises): al usar promesas, se pueden encadenar diferentes operaciones, y manejar los errores al final:

```
hacerAlgo1()
  .then(hacerAlgo2())
  .then(hacerAlgo3())
  .catch(err => console.error(err))
```

¿Cómo identificar dónde ocurrió el error? Realmente es difícil saberlo, pero se pueden manejar errores en cada una de las funciones que llama (hacerAlgoX), y dentro del controlador de errores se arrojará un nuevo error, el cual llamará al catch exterior:

```
const hacerAlgo1 = () => {
  //...
  try {
    //...
  } catch (err) {
    //... manejarlo localmente
    throw new Error(err.message)
  }
  //...
}
```

Para poder manejar errores localmente, sin tener que hacerlo en la función que llamamos, se debe romper la cadena, crear una función en cada un .then(), y procesar la excepción:

```
hacerAlgo1
  .then(() => {
    return hacerAlgo2().catch(err => {
      //manejo del error
      throw err //romper la cadena del error!
    })
  })
  .then(() => {
    return hacerAlgo2().catch(err => {
      //manejo del error
      throw err //romper la cadena del error!
    })
  })
  .catch(err => console.error(err))
```

DIFERENTES TIPOS DE ERRORES EN NODE JS

Sin considerar los errores causados por un código mal escrito, la mayoría de los códigos en las aplicaciones de node js se consideran "errores operativos". Éstos representan problemas de tiempo de ejecución, cuyos resultados son esperados, y deben tratarse de manera adecuada. Los errores operativos no significan que la aplicación en sí tenga errores, pero deben ser manejados cuidadosamente por los desarrolladores.

Si se categorizan los tipos de errores más comunes serían los siguientes:

- Errores de operación de datos.
- Errores en tiempo de ejecución.
- Errores controlados o Condiciones de borde.
- Errores generados por el Motor de base de datos.

ERRORES DE OPERACIÓN DE DATOS

Estos suelen ser los generados por código mal escrito, o por el uso de parámetros que no cumplen con los requisitos necesarios. Como ya hemos visto, se pueden gestionar fácilmente mediante sentencias try catch, promesas, o funciones asíncronas.

ERRORES EN TIEMPO DE EJECUCIÓN

Estos ocurren cada vez que sucede algo inesperado en la aplicación, y generalmente provocan problemas catastróficos que pueden bloquear el programa. Como muchos frameworks, Node. js proporciona un mecanismo para anticipar los errores antes de que ocurran.

ERRORES CONTROLADOS O CONDICIONES DE BORDE

Establecer condiciones de contorno es una excelente manera de mantener los errores controlados en sus aplicaciones. Node ofrece varias formas de establecer estas condiciones, son:

- Mediante el objeto Error.
- Uso de errores personalizados para manejar errores operativos.
- Uso de software intermedio.

- Hacer que tu aplicación se reinicie.
- Captura de todas las excepciones no detectadas.
- Captura de todas las promesas rechazadas.

ERRORES GENERADOS POR EL MOTOR DE BASE DE DATOS

Como ya lo hemos analizado anteriormente en este curso, el manejo de errores de la base de datos se puede hacer de manera efectiva, a través del objeto de resultado devuelto por cada consulta. Sin embargo, hay más errores específicos del motor de la base de datos, los cuales se pueden encontrar al trabajar con Postgres. Algunos de éstos se clasifican en los siguientes códigos de error:

- Código de error 42601:

  Nombre: syntax_error.

  Descripción: la palabra clave debe tener el espaciado correcto, o un punto y coma para ser identificada.

- Código de error 42501:

  Nombre: privilegio_insuficiente

  Descripción: el comando GRANT debe ejecutarse para otorgar privilegios de usuario de SQL, como: INSERT, SELECT o UPDATE.

- Código de error 42602:

  Nombre: invalid_name

  Descripción: un nombre de tabla o base de datos tiene letras mayúsculas o minúsculas incorrectas, caracteres, o una combinación de ambos errores.

- Código de error 42622:

  Nombre: nombre_demasiado_largo

  Descripción: el identificador supera el límite de 63 bytes. Esto se aplica a los nombres de bases de datos, tablas, esquemas, y otros nombres de identificadores de objetos de bases de datos de Postgres.

- Código de error 42939:

  Nombre: nombre_reservado

Descripción: la base de datos o tabla ya tiene un valor reservado de SQL.

- Código de error 42703:

  Nombre: undefined_column

  Descripción: el nombre de la columna no existe.