Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computación

Bases de Datos NoSQL

Proyecto - Fase I

Preparador:

Robinson Roa

Estudiantes:

Christian Padrón

C.I.V-26.996.952

Ismael Guerrero

C.I. V-28.073.233

Índice

Introducción

Para este proyecto se dividio la carga de trabajo a manera que, Ismael Guerrero manejo la recepcion de datos de la API, creó las funciones getVideojuegosbyEmpresa, getEmporesa y getPlataformas, aparte de que se encargó de todos los procesos que involucran estas funciones y la carga en los arreglo que después se utilizan para el proceso de inserción.

Christian Padrón se encargó de definir los esquemas de videojuegos, empresas y plataformas, y la inserción de los datos en la base de datos, las cuales también se pueden encontrar en el Index.

Entre los dos se realizaron las consultas hacia la base de datos.

Explicación

Rama en git:

Para el desarrollo de este proyecto se creó una rama en git llamada “Padron_Guerrero” donde se mantiene el historial de todo el trabajo realizado.

GameAPIs:

Se decidió que para este proyecto se usaria RAWG.io, y las funciones desarrolladas se desarrollaron en base a esta página, por este motivo se definieron las siguiente 3 funciones:

getVideojuegosbyEmpresa

Tiene como objetivo que, a través de un id de empresa, se recuperen todos los videojuegos producidos por dicha desarrolladora, para hacer eso se usa la URL de la API, su clave, el tamaño de la página, y precisamente, l el ID del desarrollador, esta función retorna arreglos de videojuegos asociados a una sola desarrolladora.

```
async getVideojuegosByEmpresa(DevId) {
    // console.log(`buscando videojuegos de ${DevId}`);
    let fetchedGames = [];

    try {

        //Hacemos la solicitud a la API y almacenamos la respuesta:
        const res = await axios.get(`${this.baseUrl}games`, {
        params: {//Estos son los parametros para la solicitud HTTP a la API
            key: this.apiKey,
```

```
                page_size: this.pageSize,//¿Por que retorna de 40 en 40 y no de
150 en 150?

                // page: page,//Pagina de la API que se esta consumiendo

                developers : DevId

            }

            });



            const fetchedPage = res.data.results;

            // gamesCount = res.data.count;



            if (!fetchedPage || fetchedPage.length === 0) {

                console.warn(`No se encontraron videojuegos para el desarrollador
con ID: ${DevId}`);

                return fetchedGames;

            }



            fetchedGames = fetchedGames.concat(res.data.results);//El atributo
results es el arreglo de videojuegos en la respuesta http (Segun la documentacion
de RAWG.IO)



            // Agregar el campo DeveloperId a cada juego

            fetchedGames = fetchedGames.map(game => {

                game.DeveloperID = DevId;

                game.platforms = game.platforms.map( p => {return
p.platform.id});//Si se quiere guardar la plataforma
```

```
                return game;

            });



            // console.log(fetchedGames[0]);

            //Finalmente retornamos el array de videojuegos captados:

            return fetchedGames;

        } catch (error) {

            console.error('Hubo un error solicitando los videojuegos: ',
error.message);

        }

    };
```

getEmpresa

Esta función es más trivial, puesto que lo que realiza es una consulta de varias empresas, utilizando la URL de la página, la clave de la API, y el tamaño de la página, retornando un arreglo de las mismas.

```


          async getEmpresas() {

              //Asumimos que las empresas son quienes desarrollan los
videojuegos

              //Por lo que llenaremos la coleccion empresas con desarrolladores

              console.log("Leyendo Empresas  disponibles.\n")
```

```
            let fetchedInterprise = [];



            try {

                    //Hacemos la solicitud y la guardamos en un arreglo:

                    let resDevelopers = await
axios.get(`${this.baseUrl}developers`, {

                        params: {//Estos son los parametros para la solicitud
HTTP a la API

                            key: this.apiKey,

                            page_size: this.pageSize,//¿Por que retorna de 40
en 40 y no de 150 en 150?

                        }

                    });



                    //Evaluamos el estado de la respuesta de la solicitud:

                    if(resDevelopers.status>=200 &&
resDevelopers.status<=299){

                        fetchedInterprise =
fetchedInterprise.concat(resDevelopers.data.results);

                        // console.log(`En la pagina ${page} hemos captado
${resDevelopers.data.results.length} Desarrolladores`);

                    }



                console.log(`Se han recibido ${fetchedInterprise.length}
Empresas.`)
```

```
                //Retornamos las empresas leidas:
                return fetchedInterprise;

            } catch (error) {
                console.error('\n\nHubo un error solicitando las Empresas: ',
error);
            }
        };
```

GetPlataformas

Esta función consulta la API en busca de plataformas, utilizando el url y la clave, y recibe la respuesta en forma de un arreglo de objetos de plataformas la cual retorna.

```
        async getPlataformas(){

            //Leemos las platafromas disponibles desde la API:
            console.log("Leyendo Plataformas disponibles.\n")
            let fetchedPlatforms = [];

            try {

                // console.log(`Solicitud a realizar:
${this.baseUrl}platforms`)
                //Hacemos la solicitud a la API y almacenamos la respuesta:
```

```
                const res = await axios.get(`${this.baseUrl}platforms`, {
                    params: {//Estos son los parametros para la solicitud HTTP a la API
                        key: this.apiKey,
                    }
                });

                const fetchedPage = res.data.results;

                //Agregamos la pagina captada al array de juegos:
                fetchedPlatforms = fetchedPlatforms.concat(res.data.results);//El atributo results es el arreglo de videojuegos en la respuesta http (Segun la documentacion de RAWG.IO)

                console.log(`Se han recibido ${fetchedPlatforms.length} Plataformas.`)
                //Retornamos el array de plataformas captadas:
                return fetchedPlatforms;

            } catch (error) {
                console.error('\n\nHubo un error solicitando las plataformas: ', error);
            }
        };
```

Lectura de los datos:

Para realizar la lectura de datos, por parte del documento Index se utilizan las funciones definidas anteriormente, junto a algo de manipulación de la data:

Lo primero que sucede es que se piden tanto las plataformas como las empresas con las funciones getPlataformas y getEmpresas respectivamente, guardando los resultados en un arreglo, los cuales serán transferidos a dos variables que serán los arreglos dedicados a albergar desarrolladores y plataformas, posteriormente ve va empresa por empresa, agarrando el ID para realizar una consulta solicitando videojuegos pertenecientes a dicha desarrolladora, de esto deberíamos recibir un arreglo de juegos el cual se concatena al arreglo videojuegos que contendrá todos los juegos de todas las empresas, y será utilizado para generar las inserciones.

```
    const getData = async () => {

        console.log("\n");



        //Para poder resolver las 3 promesas de forma concurrente:

                                        //     const     solictudes     =     await
Promise.all([getVideojuegos(),getPlataformas(),getEmpresas()]);

                                              const     solictudes     =     await
Promise.all([gameAPI.getPlataformas(),gameAPI.getEmpresas()]);



        //Separamos los listados captados

        plataformas = solictudes[0];

        empresas = solictudes[1];
```

```
          //Leemos videojuegos por cada desarrollador para asi poder relacionar
estas colecciones con el uso del patron Embeber

        // videojuego.DeveloperID -> Empresa.id

        console.log("Leyendo Videojuegos disponibles por Desarrolladores:\n")

        for (let i = 0; i < empresas.length; i++) {

             //Estas solicitudes deben ser realizadas secuencialmente para evitar
bloquear el script con llamadas a la API.

            let empresa = empresas[i];

            try {

                //Lectura de los juegos desde la API de Rawg:

                            console.log(`\tLeyendo  videojuegos  desarrollados  por
${empresa.name}.`)

                                                    let   juegosLeidos   =   await
gameAPI.getVideojuegosByEmpresa(empresa.id);



                //Impresion de prueba:

                                    //  console.log(`\tPrimer  videojuego  leido:
${juegosLeidos[0].platforms}`)



                //Aviso de juegos leidos:

                                console.log(juegosLeidos.length  ?  `\tSe  leyeron
${juegosLeidos.length} desarrollados por ${empresa.name}\n` : `No se encontraron
juegos desarrollados por ${empresa.name}\n`);
```

```
                    //Sumamos los juegos leidos a nuestro arreglo de videojuegos
totales:

                videojuegos = videojuegos.concat(juegosLeidos);

            } catch (error) {

                    console.error(`\tError obteniendo videojuegos para la empresa
${empresa.name}\n`);

            }

        }



           console.log(`\nHemos captado ${videojuegos.length} Videojuegos entre
todos los desarrolladores!`);

         // console.log(`VideoJuego:\n${JSON.stringify(videojuegos[0], null, 2)}
Videojuegos`);

    }



    await getData(); //Para esperar que termine la lectura de datos.
```

Definición de los esquemas para las colecciones

Los esquemas se encargan de recibir y simplificar los datos provenientes de la API, sabiendo que hay datos como plataformas, valoraciones, géneros, entre otros que son objetos compuestos de varios objetos en Videojuegos, la simplificación se realizó eliminando los datos que a nuestra discreción se consideraban prescindibles quedándonos solo con los datos importantes como títulos, diminutivos, algunos ids, etc. En este sentido, este diseño se realizó para facilitar la asociación con otras “tablas” es decir, se puede buscar un videojuego, preguntar por el desarrollador, recibir el resultado y con este consultar en empresas para hacer una búsqueda por ese valor.

Estructura de Videojuegos:

```
const videojuegoSchema = new mongoose.Schema({

//Estas 2 siguientes referencian a las otras 2 colecciones

DeveloperID : {type: Number },

platforms: {type:[Number]},



// Demas atributos:

id: { type: Number, required: true },

slug: { type: String, required: true },

name: { type: String, required: true },

released: { type: Date },

rating: { type: Number },

rating_top: { type: Number },

ratings: [{

  title: { type: String },

  count: { type: Number },

  percent: { type: Number }

}],

ratings_count: { type: Number },

reviews_text_count: { type: Number },

added: { type: Number },

metacritic: { type: Number },
```

```
  playtime: { type: Number },
  suggestions_count: { type: Number },
  updated: { type: Date },
  reviews_count: { type: Number },
  parent_platforms: [{
    id: { type: Number },
    name: { type: String, minlength: 1, maxlength: 100 },
    slug: { type: String }
  }],
  genres: [{
    name: { type: String },
    slug: { type: String },
  }],
  tags: [{
    name: { type: String },
    slug: { type: String, match: /^[a-zA-Z0-9_]+$/ },
  }],
});
```

Siguiendo este lineamiento, Empresas (considerando que son empresas desarrolladoras) y plataformas siguieron una filosofía similar para ser construidas.

Estructura de Plataformas:

```
const plataformaScheme = new mongoose.Schema({

  id: { type: Number, required: true },

  name: { type: String, required: true },

  slug: { type: String, required: true },

  games_count: { type: Number },

});
```

Estructura de Empresas:

```
const empresaScheme = new mongoose.Schema({

  id: { type: Number, required: true },

  name: { type: String, required: true },

  slug: { type: String, required: true },

  games_count: { type: Number, required: true },

  games: [{ type: Number }],

});
```

Inserción de los Datos en la Base de Datos

Para la inserción de datos se realiza lo siguiente:

Inserción de Videojuegos

Para este apartado se crea un objeto “Videojuego” llamado nuevoVideojuego, el cual es llenado con los datos recibidos de cada objeto del arreglo de videojuegos, cabe destacar que cada videojuego contiene un arreglo lleno de al menos un tipo de datos, para esto se llena de forma iterativa , es en este proceso de datos que se descartan varios campos del objeto original de videojuegos, como lo podría ser tiendas o “Stores”.

```
//Inserto las videojuegos

for (const juego of videojuegos) {

    const nuevoVideojuego = new Videojuego({

      id: juego.id,

      name: juego.name,

      slug: juego.slug,

      released: juego.released,

      rating: juego.rating,

      rating_top: juego.rating_top,

      ratings_count: juego.ratings_count,

      reviews_text_count: juego.reviews_text_count,

      added: juego.added,

      metacritic: juego.metacritic,

      playtime: juego.playtime,

      suggestions_count: juego.suggestions_count,

      updated: juego.update,

      reviews_count: juego.reviews_count,

      platforms: juego.platforms,
```

```
    DeveloperID: juego.DeveloperID,
  });
  for (const rat of juego.ratings){//
      const aux = {
          tittle: rat.title,
          count: rat.count,
          percent: rat.percent,
      };
      nuevoVideojuego.ratings.push(aux);
  }
  for (const plat of juego.parent_platforms){//
      const aux = {
          id: plat.id,
          name: plat.name,
          slug: plat.slug,
      };
      nuevoVideojuego.parent_platforms.push(aux);
  }
  for (const gen of juego.genres){//
      const aux = {
          id: gen.id,
          name: gen.name,
          slug: gen.slug,
```

```
        };

        nuevoVideojuego.genres.push(aux);

    }

    for (const tag of juego.tags){//

        const aux = {

            name: tag.name,

            slug: tag.slug,

        };

        nuevoVideojuego.tags.push(aux);

    }

    await mongoClient.insertar('Videojuego', nuevoVideojuego);

}
```

Insertar plataformas:

A Partir de aquí los procesos son parecidos pero más sencillos que en videojuegos, ya que no hay que iterar en arreglos.

```
//Inserto las plataformas

    for (const plat of plataformas) {

        const nuevaPlataforma = new Plataforma({

          id: plat.id,

          name: plat.name,

          slug: plat.slug,

          games_count: plat.games_count
```

```
        });

        await mongoClient.insertar('Plataforma', nuevaPlataforma);

    }
```

Insertar empresas:

```
      //inserto las empresas

          for (const empresa of empresas) {

              const nuevaEmpresa = new Empresa({

                id: empresa.id,

                name: empresa.name,

                slug: empresa.slug,

                games_count: empresa.games_count,

                games: empresa.games.id // linea basura que coloque para
mentirle al codigo diciendo que insertaba algo

              });

              for (const gam of empresa.games){//se insertan los ids de los
juegos de cada empresa para poder hacer busquedas posteriores de juegos por
empresa desarrolladora

                  nuevaEmpresa.games.push(gam.id);

              }
```

```
            await mongoClient.insertar('Empresa', nuevaEmpresa);

        }
```

Consultas

Consulta 1

Para realizar esta consulta primero comprobamos si el arreglo está o no vacío, luego se obtiene una referencia a la colección Videojuego, y se crea una consulta en donde se toma la propiedad a comparar, y dentro del objeto de la propiedad cual es el campo que se va a comparar y con que, esta consulta se ejecuta en la colección y se retorna la respuesta.

```
async consulta1(generos){


    try{
        //Validamos que el arreglo pasado no esta vacio
        if (!generos || !generos.length) {
            throw new Error("Debes proporcionar una lista de géneros no vacía");
            return [];
        }


        //Obtenemos la coleccion:
        const collection = this.db.collection("Videojuego");


        // Con este Query garantizamos que para el arreglo de obteneros 'genres'
```

```
        // para cada objeto su atributo name debe estar en el arreglo de nombres
pasado por parametros:

        const query = { genres: { $elemMatch: { name: { $in: generos } } } };



        // Ejecuta la consulta y devuelve los juegos encontrados

        const r = await collection.find(query).toArray();



        // console.log(r.length)

        return r;

    }catch(error){

        console.log(error.message)

    }

}
```

Consulta 2

Se selecciona la colección de videojuegos, y se utiliza la aggregate sobre videojuegos para realizar una búsqueda en comprenda los rangos “fechaInicio” y “fechaFin”, luego asociarlos por el campo del id del desarrollador com la empresa, para posteriormente filtrar comparando el id de dicha empresa con el solicitado, y con el project se toman los atributos pertinentes.

```
async consulta2(empresas, fechaInicio, fechaFin){

    try{
        const collection = this.db.collection("Videojuego");

          const r = await collection.aggregate([
            {   //Todos los juegos cuya fecha de lanzamiento este entre las
fechas dadas
```

```
                $match: {
                    released: {
                        $gte: fechaInicio,
                        $lt: fechaFin
                    }//,
                    //DeveloperID: { $in: empresas }
                }
            },
            {   //Buscamos asociar cada juego son su empresa desarrolladores
                $lookup: {
                    from: 'Empresa',
                    localField: 'DeveloperID',
                    foreignField: 'id',
                    as: 'Empresa'
                }
            },
            {//Tomamos en cuenta solo las empresas cuyo nombre esten en el array
pasado por parametros
                $match: {
                    'Empresa.name': { $in: empresas }
                }
            },
            {//Tomamos solamente los atributos relevantes
                $project: {
                    gameID : '$id',
                    gameSlug : '$slug',
                    gameName: '$name',
                    gameReleaseDate : '$released',
                    EmpresaName: '$Empresa.name',
                    EmpresaID: 'Empresa.id'
                }
            }
        ]).toArray();

        // console.log(r.length)
        return r;

    }catch(error){
        console.log(error.message)
```

```
    }

};
```

Consulta 3

Para esta consulta se conecta con la colección Videojuego y se hace un aggregate con el cual se compara la cantidad de plataformas “real” con la cantidad de plataformas indicadas por el pase de parámetros y después se emparejan los videojuegos que cumplan la condición con las plataformas, mostrando los datos pertinentes.

```
async consulta3(cantidadDePlataformas){

    const collection = this.db.collection("Videojuego");

    const r = await collection.aggregate([
        {

            //Solo los videojuegos que esten para mas de 'cantidadDePlataformas'
plataformas
            $match: {
                //$id: 405,
                $expr: {
                    $gt: [{ $size: "$platforms" },cantidadDePlataformas]
                }
            }
        },
        {   //hacemos el match para cada plataforma disponible
            $lookup: {
                from: "Plataforma",
                localField: "platforms",
                foreignField: "id",
                as: "platformsData"
            }
        },
        {   //Proyectamos los atributos importantes
            $project: {
```

```
                id:1,
                DeveloperID:1,
                name:1,
                slug:1,
                platformsData:1
            }
        }
    ]).toArray();


    return r;
}
```

Consulta 4

Se realiza un aggregate sobre Videojuego, donde se compara la cantidad de plataformas con el valor requerido, para después asociarlo con las plataformas de la colección plataformas y retornar los campos requeridos.

```
async consulta3(cantidadDePlataformas){

    const collection = this.db.collection("Videojuego");

    const r = await collection.aggregate([
        {

            //Solo los videojuegos que esten para mas de 'cantidadDePlataformas'
plataformas
            $match: {
                //$id: 405,
                $expr: {
                    $gt: [{ $size: "$platforms" },cantidadDePlataformas]
                }
            }
        },
        {   //hacemos el match para cada plataforma disponible
            $lookup: {
                from: "Plataforma",
```

```
                localField: "platforms",
                foreignField: "id",
                as: "platformsData"
            }
        },
        {   //Proyectamos los atributos importantes
            $project: {
                id:1,
                DeveloperID:1,
                name:1,
                slug:1,
                platformsData:1
            }
        }
    ]).toArray();


    return r;
}
```

Consulta 5

Se realiza un aggregate sobre la colección Videojuegos donde se calcula el promedio de valoración de los juegos, y después se filtran primero por la cantidad de géneros de los videojuegos con el valor solicitado, y posteriormente comparando las valoraciones con el promedio, posteriormente se toman los campos pertinentes.

```
async consulta5(cantidadDeGeneros){

    const collection = this.db.collection("Videojuego");

    const r = await collection.aggregate([
        {//Esto es para calcular el promedio del rating de todos los documentos
de la coleccion videojuegos y mantener ese campo en una "variable"
            $setWindowFields: {
                output: {
                    avgRating: { $avg: "$rating" }
                }
            }
        },
```

```
        {   //Evaluamos que se cumplan las 2 condiciones
            $match: {
                $expr: {
                    $and: [
                        { $lt: [cantidadDeGeneros, { $size: "$genres" }] },//Mas
de X generos
                        { $gt: ["$rating", "$avgRating"] }//Rating meyor al
promedio
                    ]
                }
            }
        },
        {
            $project: {
                id: 1,
                name: 1,
                rating:1,
                avgRating:1,
                genresCount: { $size: "$genres" }
            }
        }
    ]).toArray();

    return r

}
```

Consulta 6

Después de comprobar que el arreglo de etiquetas es válido se procede a agarrar una referencia a la colección Videojuego, después se crea un pipeline el cual consta de comparar las etiquetas del videojuego con las etiquetas del arreglo, después se ordena de forma ascendente por el campo de la fecha y en el project se escoge que imprimir.

```
async consulta6(etiquetas){
```

```
    try {
        // Validamos que el arreglo pasado no esté vacío
        if (!etiquetas || !etiquetas.length) {
          throw new Error("Debes proporcionar una lista de etiquetas no vacía");
          return [];
        }

        // Obtenemos la colección
        const collection = this.db.collection("Videojuego");

        // Construimos el pipeline de agregación
        const pipeline = [
          {
            // Filtra juegos cuyos nombres esten en el array etiquetas
            $match: {
              tags: { $elemMatch: { name: { $in: etiquetas } } },
            },
          },
          {
            // Ordenamos por fecha de lanzamiento
            $sort: { released: 1 }, // 1 para orden ascendente, -1 para
descendente
          },
          {//hacemos project para los atributos relevantes
            $project: {
                id: 1,
                name: 1,
                slug: 1,
                released: 1,
                tags:1,
                // tagsName: {
                //   $map: {
                //     input: "$tags", // Arreglo de entrada a la función map
                //     as: "$t", // Alias para cada elemento del arreglo
                //     in: "$$t.name"}
                //   }
                // }
              }
```

```
          }
        ];

        // Ejecuta la consulta y devuelve los juegos encontrados
        const r = await collection.aggregate(pipeline).toArray();

        return r;
      } catch (error) {
        console.log(error.message);
      }

}
```

Consulta 7

Se revisa que el arreglo sea válido, posteriormente se crea un pipeline el cual se encarga de buscar los juegos que contengan los géneros solicitados comparándolos con el arreglo de géneros pasado por parámetro, posterior a eso se realiza el producto cartesiano de los géneros, seguido a eso se agrupan los géneros por su nombre y se saca el promedio de su valoración para, por último, ejecutar el aggregate de este pipeline en la colección de videojuego.

```
  async consulta7(generos){

    // Valida que se proporcione un arreglo de géneros válido
    if (!generos || !generos.length || !generos.every(genero => typeof genero ===
"string")) {
        throw new Error("Debes proporcionar un arreglo válido de nombres de
géneros (strings)");
    }

    // Pipeline de agregación para calcular el promedio de calificación por
género
    const pipeline = [
        {   //tomamos en cuenta los juegos que pertenezcan al conjunto de generos
dado:
            $match: {
```

```
                genres: { $elemMatch: { name: { $in: generos } } },
              }
        },
        {   //Desempaquetamos cada juego por genero:
            $unwind : "$genres"
        },
        {   //Calculamos el promedio de los ratings agrupando por genero
            $group: {
                _id: "$genres.name",
                avgRating : {$avg: "$rating"}
            }
        },
        {   //Este project es para mejorar la lectura en el test:
            $project: {
                _id:0,
                avgRating:1,
                genreName: "$_id",

            }
        }

    ];

    // Obtenemos la colección
    const collection = this.db.collection("Videojuego");

    // Ejecuta la agregación y combina los resultados
    const r = await collection.aggregate(pipeline).toArray();
    return r;

}
```

Consulta 8

Primero verificamos que el arreglo sea válido, después, se referencia a la colección videojuegos, y se procede a preparar la consulta, primero utilizando la función regedex para hacer la búsqueda por patrón, y se genera el aggregate en

videojuegos, donde con el regedex se hace una búsqueda en el campo name de los juegos, posteriormente con el project se escogen el id y el nombre para su impresión.

```
  async consulta8(palabra){
    // Valida que se proporcione una palabra clave
    if (!palabra || typeof palabra !== "string") {
      throw new Error("Debes proporcionar una palabra clave válida");
    }

    const collection = this.db.collection("Videojuego");

    // Crea una expresión regular para buscar la palabra clave (ignora
mayúsculas/minúsculas)
    const regex = new RegExp(palabra, 'i');

    // Busca juegos donde el nombre coincida con la expresión regular
    // const query = { name: { $regex: regex } };

    // Ejecuta la consulta y devuelve los juegos encontrados
    const r = await collection.aggregate([
        {//Los juegos deben contener la palabra en su nombre
            $match: {
                name: { $regex: RegExp(palabra, 'i') }
                //Esta Regex es para buscar el nombre sin tomar en cuenta las
mayusculas
            }
        },
        {   //Proyectamos solo los atributos relevantes
            //para validar las salidas de la consulta
            $project: {
                id: 1,
                name: 1
            }
        }
    ]).toArray();
    return r;
  }
```

Consulta 9:

Consulta 10:

Link de la rama del repositorio:

https://github.com/ismaelgt24/bdd-nosql-proyecto-fase-1/tree/Padron_Guerrero