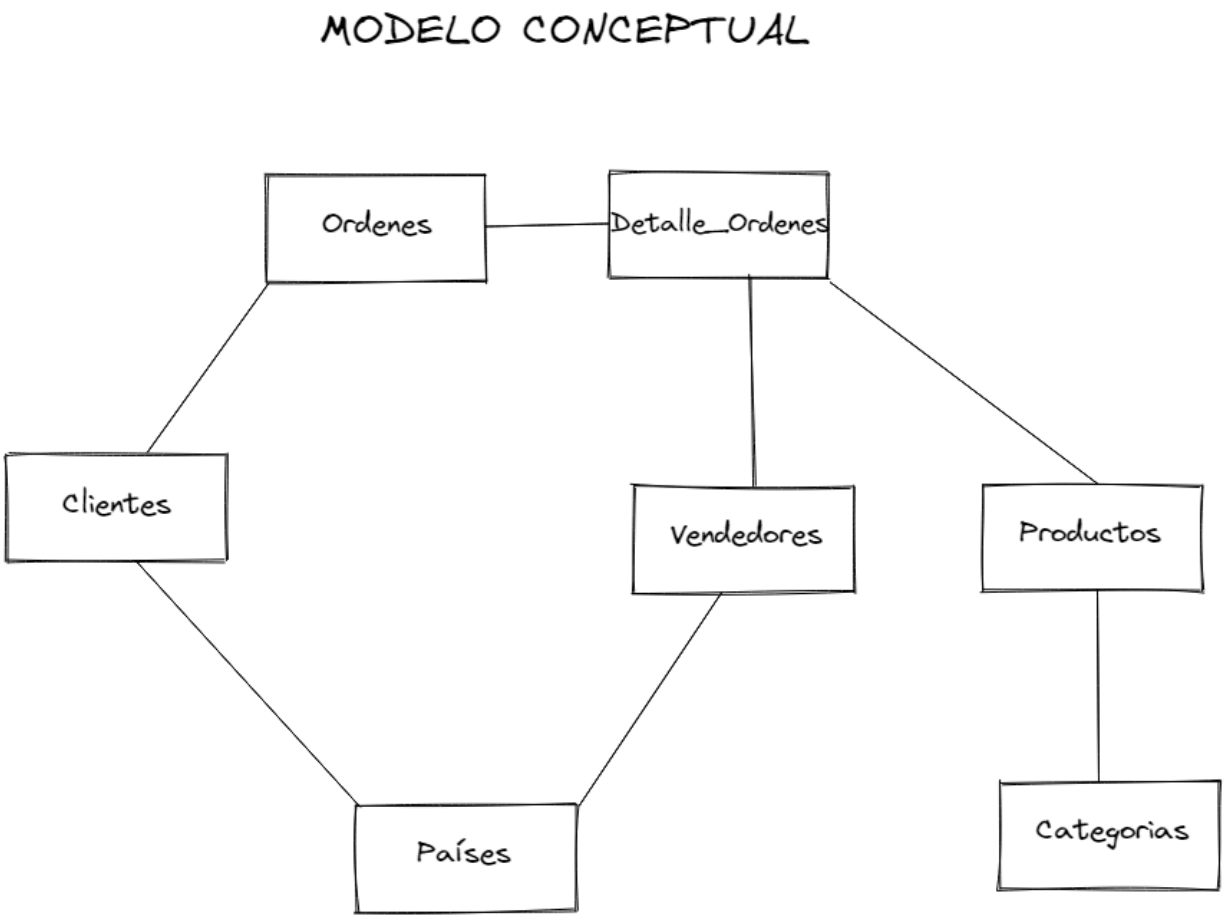
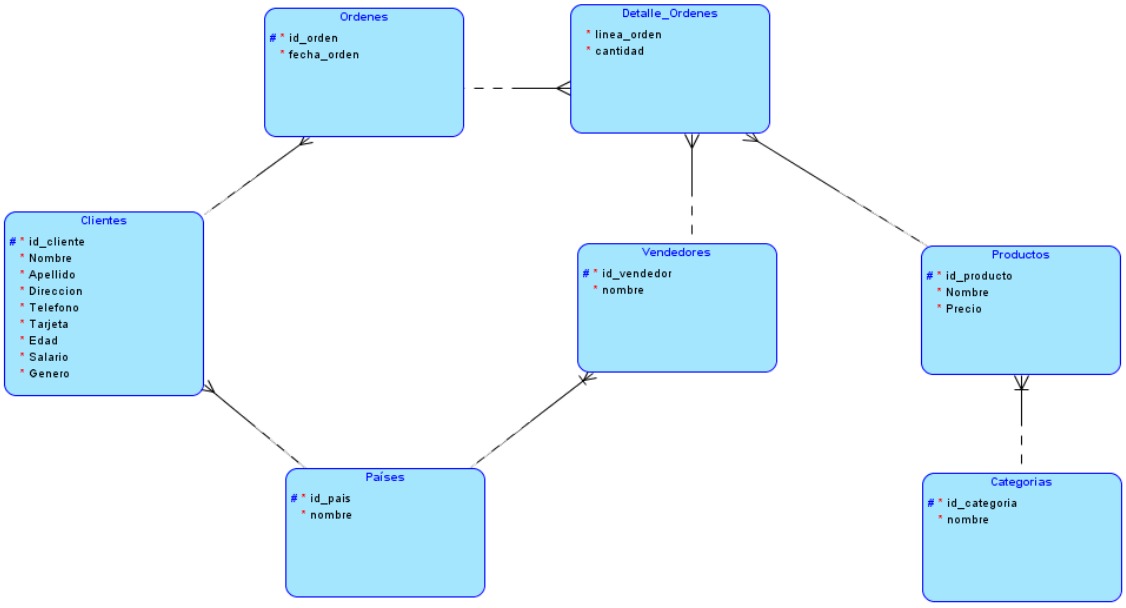
Edin Montenegro - 201709311

Manual Técnico

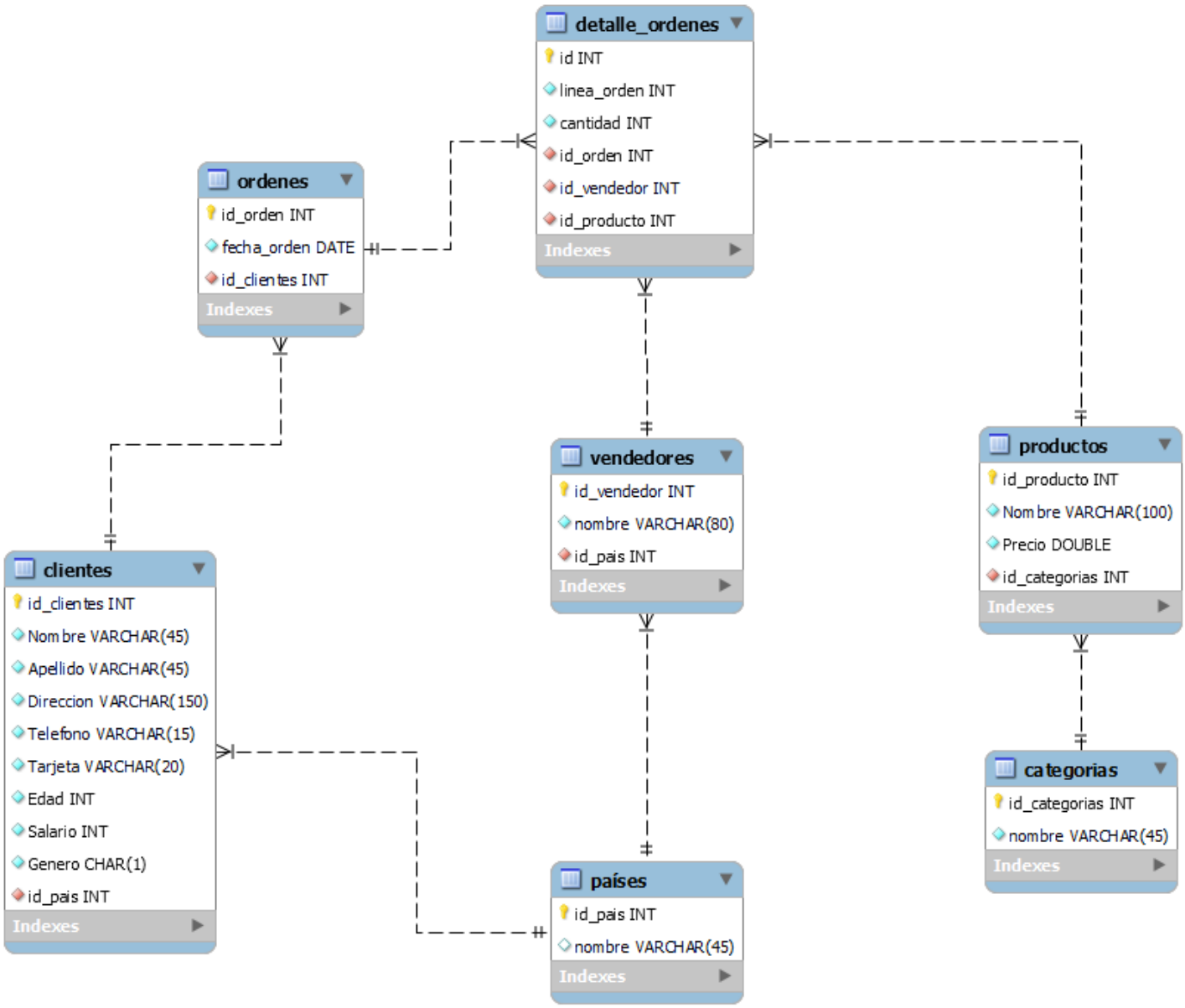
Esquema Conceptual:



Esquema Lógico:



Esquema Físico



Descripción de tablas:

Esta base de datos cuenta con 7 tablas con las siguientes descripciones:

- **Tabla Clientes**: Esta tabla contiene todos los datos del cliente y es una entidad débil, ya que un campo en esta tabla hace referencia a otra tabla para elegir el id del país.
  PK: id_clientes.
  FK: id_pais de la tabla Países.

- **Tabla Ordenes:** Esta tabla contiene las ordenes realizadas en las cuales se especifican los productos, cantidad, vendedor y a que cliente corresponde dicha orden, es una entidad débil.
  PK: id_orden.
  FK: id_cliente de la tabla Clientes.

- **Tabla Detalle Ordenes:** Esta tabla contiene el detalle de cada orden de la tabla ordenes, esto a través del id de cada orden se generan los detalles, es una entidad débil.
  PK: id
  FK:id_orden de la tabla órdenes.
  FK:id_vendedor de la tabla vendedores.
  FK:id_producto de la tabla productos.

- **Tabla Vendedores:** Esta tabla contiene la información personal de los vendedores de esta empresa, y también es una entidad débil.
  PK: id_vendedor.
  FK: id_pais de la tabla Países.

- **Tabla Productos:** Esta tabla almacena el stock de productos que se comercializan en esta empresa, es una entidad débil.
  PK: id_producto.
  FK: id_categorias.

- **Tabla Países:** Esta entidad es fuerte ya que no depende de algún campo en otra tabla, en esta tabla se almacenan los países que a lo largo del tiempo se han registrado en el ingreso del personal de ventas y de los clientes.
  PK: Id_pais

- **Tabla Categorías:** Esta entidad es de tipo fuerte, y posee la información que, a lo largo del tiempo en el ingreso de productos, se han logrado establecer sus respectivas categorías.
  PK:id_categorias

Descripción de la API:

Este api se realizó con el lenguaje de Golang en su versión 1.18.

Para la creación de las rutas GET, se utilizó la librería gorilla/mux, la cual ayuda de manera fácil y rápida el poder implementar un servicio de http request, se crearon 10 rutas con el método GET (1 ruta para cada consulta).

En la conexión con la base de datos, para este caso en especifico se utiliza el DBMS de MYSQL, por ello para lograr la comunicación entre el api y MYSQL fue necesario el uso de la librería “go-sql-driver/mysql” , la cual lleva los siguientes parámetros para su conexión:

```go
var conn = MySQLConnection()

func MySQLConnection() *sql.DB {
    usuario := "root"
    pass := 503802
    host := "tcp(localhost:3306)"
    db := "proyecto1"
    conn, err := sql.Open("mysql", fmt.Sprintf("%s:%d@%s/%s", usuario, pass, host, db))
    if err != nil {
        fmt.Println("HAY ERROR: \n", err)
    } else {
        fmt.Println("se ha conectado a mysql!")
    }
    return conn
```

En cada consulta se ingresa un query dentro del método http y para el retorno se crea un struct para genera el formato JSON de acuerdo a lo que deseamos devolver al usuario, a continuación se muestra el struct y la consulta para el reporte 1:

```go
// STRUCTS PARA JSONS DE CONSULTAS
type Consulta1 struct {
    Id_cliente int    `json:"ID"`
    Nombre     string `json:"NOMBRE"`
    Apellido   string `json:"APELLIDO"`
    Pais       string `json:"PAIS"`
    Monto      string `json:"MONTO_TOTAL"`
    No_Compras string `json:"veces_compra"`
}
```

En este struct como se puede observar contiene 6 campos, debido a que en la requisición de este reporte se solicita que se muestren estos atributos, es por ello por lo que es necesario este struct para poder la información de manera correcta.

```go
func Reporte1(response http.ResponseWriter, request *http.Request) {
    response.Header().Add("content-type", "application/json")
    var listUsr []Consulta1
    query := `SELECT c.id_clientes AS ID,c.Nombre AS NOMBRE,c.Apellido AS APELLIDO,p.nombre AS PAIS,SUM(productos.Precio*d.cantidad) AS MONTO_TOTAL, COUNT(c.id_clientes) AS veces_
    FROM clientes c
    INNER JOIN ordenes o
    ON c.id_clientes = o.id_clientes
    INNER JOIN detalle_ordenes d
    ON o.id_orden = d.id_orden
    INNER JOIN productos
    ON d.id_producto = productos.id_producto
    INNER JOIN países p
    ON c.id_pais = p.id_pais
    GROUP BY c.id_clientes
    ORDER BY veces_compra DESC
    LIMIT 1;`
    result, err := conn.Query(query)
    if err != nil {
        fmt.Println(err)
    }
    for result.Next() {
        var usr Consulta1
        err := result.Scan(&usr.Id_cliente, &usr.Nombre, &usr.Apellido, &usr.Pais, &usr.Monto, &usr.No_Compras)
        if err != nil {
            fmt.Println(err)
        }
        listUsr = append(listUsr, usr)
    }
    json.NewEncoder(response).Encode(listUsr)
}
```

Como se observa en la imagen anterior se ingresa el query dentro del método se hace la requisición a la DB y esta en su respuesta se ingresa al struct correspondiente.

Endpoint

Se crearon 10 endpoints, uno para cada reporte respectivamente. Los endpoints utilizados son los siguientes:

```
router.HandleFunc("/1", consultas.Reporte1).Methods("GET")
router.HandleFunc("/2", consultas.Reporte2).Methods("GET")
router.HandleFunc("/3", consultas.Reporte3).Methods("GET")
router.HandleFunc("/4", consultas.Reporte4).Methods("GET")
router.HandleFunc("/5", consultas.Reporte5).Methods("GET")
router.HandleFunc("/6", consultas.Reporte6).Methods("GET")
router.HandleFunc("/7", consultas.Reporte7).Methods("GET")
router.HandleFunc("/8", consultas.Reporte8).Methods("GET")
router.HandleFunc("/9", consultas.Reporte9).Methods("GET")
router.HandleFunc("/10", consultas.Reporte10).Methods("GET")
```

Estos endpoints se encuentra de manera local en el puerto 8000