QA TRAINING LEAGUE, RETO FINAL SOBRE BASES DE DATOS: EJERCICIO (A) – OCOCHO'S BARBERÍA

JONATHAN ANDRÉS VARGAS SEPÚLVEDA

Trabajo explicativo para optar a pasar a las siguientes semanas

Juan Pineda

SOFKAU
TRAINING LEAGUE QA
MEDELLÍN
2023

AGRADECIMIENTOS

A todos mis compañeros por el apoyo prestado, y haber demostrado tantos avances en esta temática.

CONTENIDO

pág.

1. INTRODUCCIÓN 6
1.1 Formulación del problema 6
1.2 Objetivos del proyecto 6
1.2.1 Objetivo General 6
1.2.2 Objetivos Específicos 6
2. DISEÑO DE LA BD 7
2.1 Modelo E-R 7
2.2 Modelo Relacional 8
2.2.1 Normalización del MR 8
2.3 Declaración de la BD en SQL 9
2.4 Consultas de visualización 12
2.5 Vistas importantes 13
2.6 Procedimientos almacenados 16
2.7 Triggers 16
3. POBLACIÓN DE LA BD 19
4. CONCLUSIONES Y CONSIDERACIONES FINALES 22
REFERENCIAS 23

LISTA DE FIGURAS

pág.

Figura 1 MER. 7
Figura 2 MR. 8
Figura 3 Declaración BD (1). 9
Figura 4 Declaración BD (2) 9
Figura 5 Declaración BD (3). 10
Figura 6 Declaración BD (4). 11
Figura 7 Consultas de visualización (1). 12
Figura 8 Consultas de visualización (2). 12
Figura 9 Figura 7 Consultas de visualización (3). 13
Figura 10 Declaración de vistas (1). 14
Figura 11 Declaración de vistas (2). 14
Figura 12 Declaración de vistas (3). 15
Figura 13 Creación de SP (1). 16
Figura 14 Creación de SP (2). 16
Figura 15 Nuevas tablas para almacenar información de Triggers. 17
Figura 16 Creación de Triggers (1). 18
Figura 17 Creación de Triggers (2). 18
Figura 18 Estructura del proyecto de conexión. 19
Figura 19 Contrato para crear conexión de BD. 19
Figura 20 Constantes necesarias para la conexión. 20
Figura 21 Métodos para consultas SELECT e INSERT. 20
Figura 22 Parámetros de configuración y consulta a realizar sin formato. 20

Figura 23 Métodos para insertar Proveedores. 21

Figura 24 Llamado desde main del método anterior. 21

Figura 25 Registros tabla Proveedor. 21

1. INTRODUCCIÓN

Se realiza una lectura del reto final BD, donde se piden los requerimientos de diseño vistos a lo largo de las dos pasadas semanas. Es importante tener en cuenta la conexión a la base de datos, por lo que se hace especial énfasis en esta. El presente documento cuenta con formulación de problema y objetivos, diseño de base de datos, población de esta y conclusiones.

1.1 FORMULACIÓN DEL PROBLEMA

El problema principal de este trabajo es la conexión con Java y MySQL, por lo que se trazan objetivos acordes con dar solución a este problema bajo el contexto del Ejercicio A, es decir, la barbería.

1.2 OBJETIVOS DEL PROYECTO

Se plantea el objetivo general y cómo se alcanzará a través de objetivos específicos.

1.2.1 Objetivo General

Realizar una conexión fiable entre Java y MySQL, para hacer consultas de todo tipo.

1.2.2 Objetivos Específicos

- Diseñar una BD (Modelo ER, MR, normalización, y definición de la BD en SQL).
- Probar el funcionamiento de la BD (10 consultas para visualizar las tablas, 6 vistas, 4 procedimientos y triggers).
- Conectar Java con MySQL con consultas básicas.
- Poblar la BD con 50 datos por tabla.

2. DISEÑO DE LA BD

2.1 MODELO E-R

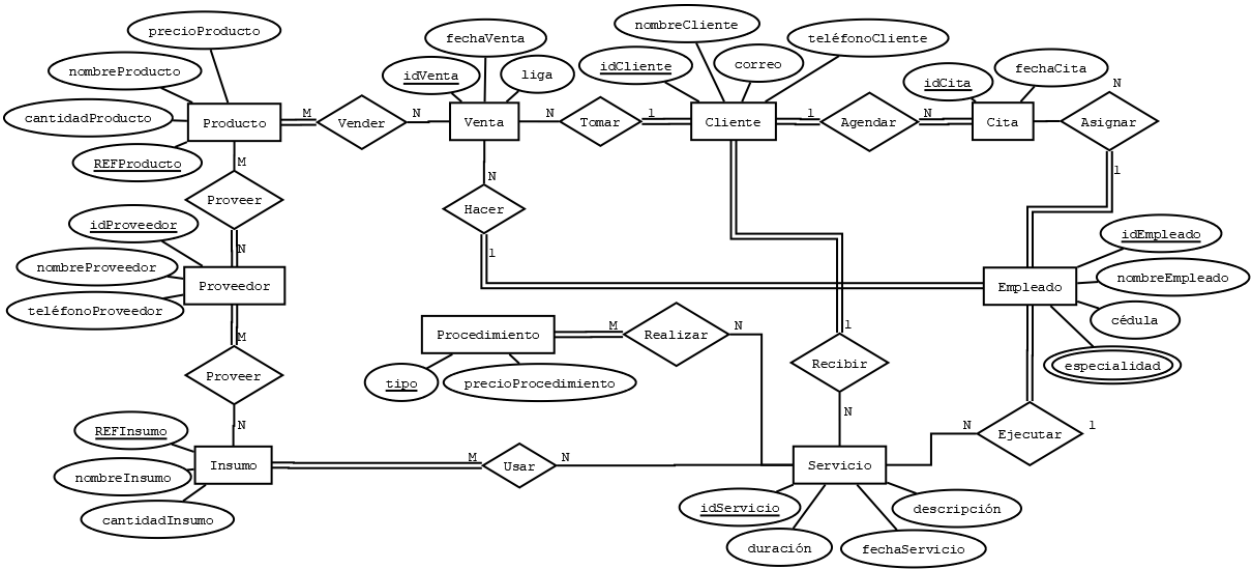

Figura 1 MER.

Para la realización del MER (Figura 1) se plantean varias entidades y relaciones entre ellas. Lo más importante para tener en cuenta son los siguientes puntos:

- Un proveedor puede proveer tanto insumos como productos. Un proveedor puede proveer muchos de estos, y uno de estos puede ser provisto por muchos proveedores.
- Un producto puede ser vendido en muchas ventas, y en una venta se pueden vender muchos productos.
- Una venta la puede hacer un solo empleado, y un empleado puede hacer muchas ventas.
- Un cliente puede tomar muchas ventas, y una venta solo puede ser tomada por un cliente.
- Una cita se le agenda a un solo cliente, y a un cliente se le agendan muchas citas.
- En una cita se puede asignar un solo empleado, y un empleado puede ser asignado a varias citas.
- Un insumo se puede usar en muchos servicios ejecutados por un solo empleado, y en un servicio se pueden usar muchos insumos.
- Un empleado puede ejecutar muchos servicios donde se realicen muchos procedimientos. Además, un procedimiento puede ser realizado en muchos servicios.

Esta lógica es extraída del funcionamiento de una barbería real, llamada Barbería Rand (Rand, 2023). Además, los atributos se disponen de la manera más simple que garantice el correcto funcionamiento de la barbería.

2.2 MODELO RELACIONAL

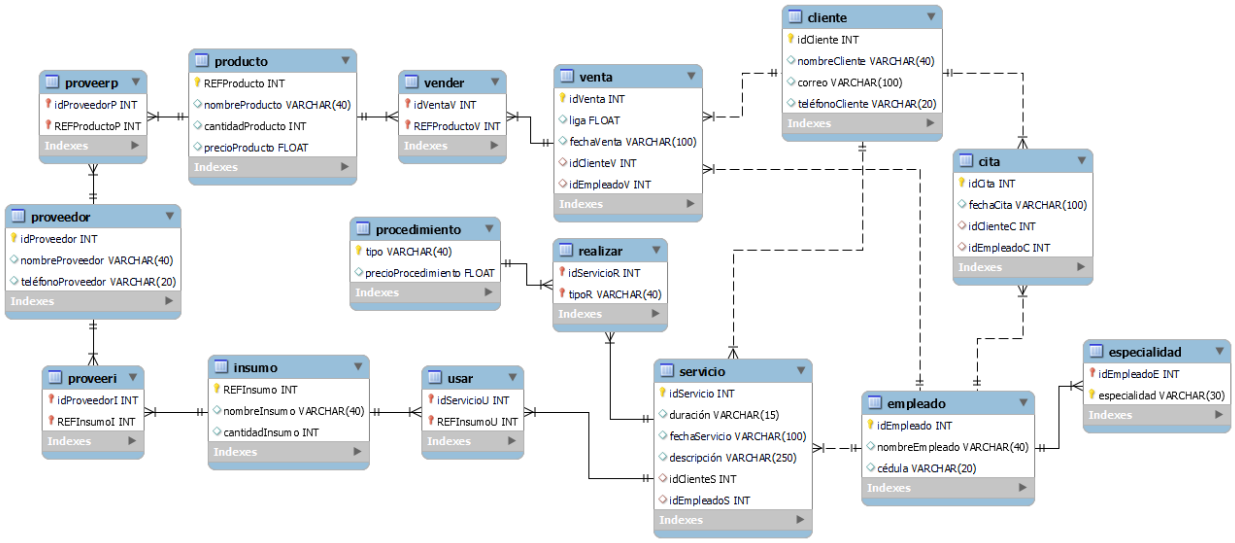

Figura 2 MR.

Para generar el MR de la BD para Ococho's Barbería, de la Figura 2, se utiliza el software Workbench, y utilizando la reglas de transformación de ME-R a MR, se obtiene un diseño limpio y simple, con tablas que me dan toda la información necesaria.

Además, gracias a todos los conceptos y conocimientos interiorizados a lo largo de las semanas, se puede garantizar que el MR está normalizado, como se verá en la siguiente sección.

2.2.1 Normalización del MR

Se tienen en cuenta las primeras 3 formas normales, así:

- **1FN**

Todos los atributos en el MR de la Figura 2 son atómicos y no son multivaluados, no pueden existir registros duplicados bajo esta estructura, y cada tabla tiene su llave primaria.

- **2FN**

Además de que se cumple la 1FN, cada valor en el MR de la Figura 2 depende únicamente de la llave primaria.

- **3FN**

Y luego de cumplir con la 2FN, también las columnas en el modelo que no están incluidas en la llave primaria no dependen transitivamente de esta.

2.3 DECLARACIÓN DE LA BD EN SQL

Una vez el MR está normalizado y listo para ser trasladado a sentencias SQL, se empiezan por las tablas que no tienen claves foráneas, ya que no necesitan que exista una tabla anterior para ser creadas. Así, se siguen en orden las sentencias que se muestran en las Figuras más adelante.

```sql
-- ----------------------------------------
-- Creando y usando la BD
-- ----------------------------------------
CREATE DATABASE ocochosBarberia;
USE ocochosBarberia;

-- ----------------------------------------
-- Se crean las tablas correspondientes
-- ----------------------------------------
-- Tabla Proveedor
CREATE TABLE proveedor (
    idProveedor INT AUTO_INCREMENT,
    nombreProveedor VARCHAR(40),
    telefonoProveedor VARCHAR(20),
    PRIMARY KEY (idProveedor)
);

-- Tabla Producto
CREATE TABLE producto (
    REFProducto INT AUTO_INCREMENT,
    nombreProducto VARCHAR(40),
    cantidadProducto INT DEFAULT 0,
    precioProducto FLOAT DEFAULT 0,
    PRIMARY KEY (REFProducto)
);

-- Tabla Insumo
CREATE TABLE insumo (
    REFInsumo INT AUTO_INCREMENT,
    nombreInsumo VARCHAR(40),
    cantidadInsumo INT DEFAULT 0,
    PRIMARY KEY (REFInsumo)
);
```

Figura 3 Declaración BD (1).

```sql
-- Tabla Proveer Prodcuto
CREATE TABLE proveerP (
    idProveedorP INT,
    REFProductoP INT,
    PRIMARY KEY (idProveedorP, REFProductoP),
    FOREIGN KEY (idProveedorP) REFERENCES proveedor (idProveedor),
    FOREIGN KEY (REFProductoP) REFERENCES producto (REFProducto)
);

-- Tabla Proveer Insumo
CREATE TABLE proveerI (
    idProveedorI INT,
    REFInsumoI INT,
    PRIMARY KEY (idProveedorI, REFInsumoI),
    FOREIGN KEY (idProveedorI) REFERENCES proveedor (idProveedor),
    FOREIGN KEY (REFInsumoI) REFERENCES insumo (REFInsumo)
);

-- Tabla Cliente
CREATE TABLE cliente (
    idCliente INT AUTO_INCREMENT,
    nombreCliente VARCHAR(40),
    correo VARCHAR(100),
    telefonoCliente VARCHAR(20),
    PRIMARY KEY (idCliente)
);

-- Tabla Empleado
CREATE TABLE empleado (
    idEmpleado INT AUTO_INCREMENT,
    nombreEmpleado VARCHAR(40),
    cedula VARCHAR(20),
    PRIMARY KEY (idEmpleado)
);
```

Figura 4 Declaración BD (2)

```sql
-- Tabla Especialidad del Empleado
CREATE TABLE especialidad (
    idEmpleadoE INT,
    especialidad VARCHAR(30),
    PRIMARY KEY (idEmpleadoE, especialidad),
    FOREIGN KEY (idEmpleadoE) REFERENCES empleado (idEmpleado)
);

-- Tabla Cita
CREATE TABLE cita (
    idCita INT AUTO_INCREMENT,
    fechaCita VARCHAR(100),
    idClienteC INT,
    idEmpleadoC INT,
    PRIMARY KEY (idCita),
    FOREIGN KEY (idClienteC) REFERENCES cliente (idCliente),
    FOREIGN KEY (idEmpleadoC) REFERENCES empleado (idEmpleado)
);

-- Tabla Venta
CREATE TABLE venta (
    idVenta INT AUTO_INCREMENT,
    liga FLOAT DEFAULT 0,
    fechaVenta VARCHAR(100),
    idClienteV INT,
    idEmpleadoV INT,
    PRIMARY KEY (idVenta),
    FOREIGN KEY (idClienteV) REFERENCES cliente (idCliente),
    FOREIGN KEY (idEmpleadoV) REFERENCES empleado (idEmpleado)
);

-- Tabla Vender
CREATE TABLE vender (
    idVentaV INT,
    REFProductoV INT,
    PRIMARY KEY (idVentaV, REFProductoV),
    FOREIGN KEY (idVentaV) REFERENCES venta (idVenta),
    FOREIGN KEY (REFProductoV) REFERENCES producto (REFProducto)
);
```

Figura 5 Declaración BD (3).

```sql
-- Tabla Servicio
CREATE TABLE servicio (
    idServicio INT AUTO_INCREMENT,
    duracion INT DEFAULT 0, -- ¡En horas!
    fechaServicio VARCHAR(100),
    descripcion VARCHAR(250),
    idClienteS INT,
    idEmpleadoS INT,
    PRIMARY KEY (idServicio),
    FOREIGN KEY (idClienteS) REFERENCES cliente (idCliente),
    FOREIGN KEY (idEmpleadoS) REFERENCES empleado (idEmpleado)
);

-- Tabla Procedimiento
CREATE TABLE procedimiento (
    tipo VARCHAR(40) DEFAULT 'CORTE',
    precioProcedimiento FLOAT DEFAULT 0,
    PRIMARY KEY (tipo)
);

-- Tabla Realizar
CREATE TABLE realizar (
    idServicioR INT,
    tipoR VARCHAR(40) DEFAULT 'CORTE',
    PRIMARY KEY (idServicioR, tipoR),
    FOREIGN KEY (idServicioR) REFERENCES servicio (idServicio),
    FOREIGN KEY (tipoR) REFERENCES procedimiento (tipo)
);

-- Tabla Usar
CREATE TABLE usar (
    idServicioU INT,
    REFInsumoU INT,
    PRIMARY KEY (idServicioU, REFInsumoU),
    FOREIGN KEY (idServicioU) REFERENCES servicio (idServicio),
    FOREIGN KEY (REFInsumoU) REFERENCES insumo (REFInsumo)
);
```

Figura 6 Declaración BD (4).

2.4 CONSULTAS DE VISUALIZACIÓN

Se crean algunas consultas para visualizar la información de las tablas principales. Hay que tener en cuenta que estas no se ejecutaran en esta sección, sino que se utilizan posteriormente en la conexión con Java y MySQL.

Así, las consultas de visualización creadas son:

```
-- -----------------------------------------------
-- Sentencias de visualización de la información
-- -----------------------------------------------
USE ocochosBarberia;

-- -----------------------------------------------
-- Visualizar stock de productos
-- -----------------------------------------------
SELECT
        REFProducto AS 'Referencia',
        nombreProducto AS 'Producto',
        cantidadProducto AS 'Unidades',
        precioProducto AS 'Precio'
FROM producto;

-- -----------------------------------------------
-- Visualizar stock de insumos
-- -----------------------------------------------
SELECT
        REFInsumo AS 'Referencia',
        nombreInsumo AS 'Producto',
        cantidadProducto AS 'Unidades'
FROM insumo;

-- -----------------------------------------------
-- Visualizar todas las citas agendadas
-- -----------------------------------------------
SELECT
        idCita AS 'Cita N°',
        fechaCita AS 'Agendada para el',
        c.nombreCliente AS 'Cliente'
FROM cita, cliente AS c
WHERE idClienteC = c.idCliente;
```

Figura 7 Consultas de visualización (1).

```
-- -----------------------------------------------
-- Visualizar servicios realizados
-- -----------------------------------------------
SELECT
        idServicio AS 'Servicio N°',
        duracion AS 'Duración',
        fechaServicio AS 'Realizado en',
        descripción AS 'Descripción'
FROM servicio;

-- -----------------------------------------------
-- Visualizar empleados en nomina
-- -----------------------------------------------
SELECT
        nombreEmpleado AS 'Empleado',
        cedula AS 'Identificación'
FROM empleado;

-- -----------------------------------------------
-- Visualizar clientes registrados
-- -----------------------------------------------
SELECT
        nombreCliente AS 'Cliente',
        correo AS 'Correo',
        telefonoCliente AS 'Teléfono'
FROM cliente;

-- -----------------------------------------------
-- Visualizar proveedores de productos e insumos
-- -----------------------------------------------
SELECT
        nombreProveedor AS 'Proveedor',
        telefonoProveedor AS 'Teléfono'
FROM proveedor;
```

Figura 8 Consultas de visualización (2).

```sql
-- ---------------------------------------------------------
-- Visualizar procedimientos ofrecidos actualmente
-- ---------------------------------------------------------
SELECT
        tipo AS 'Ofrecemos',
        precioProcedimiento AS 'Precio'
FROM procedimiento;


-- ---------------------------------------------------------
-- Visualizar la cantidad de ventas realizadas
-- ---------------------------------------------------------
SELECT COUNT(idVenta) AS 'Cantidad de ventas realizadas'
FROM venta;


-- ---------------------------------------------------------
-- Visualizar los empleados y sus citas asignadas
-- ---------------------------------------------------------
SELECT nombreEmpleado AS 'Empleado',
GROUP_CONCAT(c.idCita SEPARATOR ' | ') AS 'Cita(s)'
FROM empleado
INNER JOIN cita AS c ON idEmpleado = c.idEmpleadoC
GROUP BY idEmpleado;
```

Figura 9 Figura 7 Consultas de visualización (3).

2.5 VISTAS IMPORTANTES

Dado que la BD fue diseñada de una manera simple y sencilla, el acceso a toda la información más importantes puede ser realizado por algunas vistas. Otra información menos relevante puede generar estructuras en las vistas mucho más complejas, pero que, a la hora de arrojar la información, no aportan nada.

Por esto se seleccionan las vistas que muestran las especialidades que tienen cada uno de los empleados, qué procedimientos se hicieron en cada servicio, qué insumos fueron consumidos en cada servicio, cuánto dinero se ha ganado cada empleado por comisiones, ver que citas hay agendadas por cliente, y ver el tiempo total que ha trabajado cada empleado para así pagarle después.

Las vistas hacen uso de funciones como suma y concatenar por grupo, para calcular los totales, y para mostrar datos múltiples que se relacionen con otros datos iguales. Así, se presentan las declaraciones de las vistas descritas.

```sql
-- -----------------------------------------------------
-- Vistas Importantes
-- -----------------------------------------------------
USE ocochosBarberia;

-- -----------------------------------------------------
-- Visualizar las especialidades de cada empleado
-- -----------------------------------------------------
CREATE VIEW especialidadesEmpleado AS
SELECT
        e.nombreEmpleado AS 'Empleado',
        GROUP_CONCAT(es.especialidad SEPARATOR ', ') AS 'Especialidad(es)'
FROM empleado AS e
INNER JOIN especialidad AS es
ON e.idEmpleado = es.idEmpleadoE
GROUP BY e.idEmpleado;

-- -----------------------------------------------------
-- Visualizar los procedimientos realizados en un
-- servicio
-- -----------------------------------------------------
CREATE VIEW procedimientosEnServicio AS
SELECT
        s.idServicio AS 'Servicio N°',
        GROUP_CONCAT(p.tipo SEPARATOR ', ') AS 'Procedimiento(s)'
FROM servicio AS s
INNER JOIN realizar AS r
ON s.idServicio = r.idServicioR
INNER JOIN procedimiento AS p
ON r.tipoR = p.tipo
GROUP BY s.idServicio;
```

Figura 10 Declaración de vistas (1).

```sql
-- -----------------------------------------------
-- Visualizar el tiempo total trabajado por cada
-- empleado
-- -----------------------------------------------
CREATE VIEW tiempoTotalEmpleado AS
SELECT
        e.nombreEmpleado AS 'Empleado',
        SUM(s.duracion) AS 'Tiempo total laborado'
FROM empleado AS e
INNER JOIN servicio AS s
ON e.idEmpleado = s.idEmpleadoS;
```

Figura 11 Declaración de vistas (2).

```sql
-- ------------------------------------------------------
-- Visualizar los insumos usados en un servicio
-- ------------------------------------------------------
CREATE VIEW insumosEnServicio AS
SELECT
        s.idServicio AS 'Servicio N°',
        GROUP_CONCAT(i.nombreInsumo SEPARATOR ', ') AS 'Insumo(s)',
        GROUP_CONCAT(i.cantidadInsumo SEPARATOR ' | ') AS 'Unidad(es)'
FROM servicio AS s
INNER JOIN usar AS u
ON s.idServicio = u.idServicioU
INNER JOIN insumo AS i
ON u.REFInsumoU = i.REFInsumo
GROUP BY s.idServicio;
-- ------------------------------------------------------
-- Visualizar la cantidad total de ligas que se ha
-- ganado un empleado
-- ------------------------------------------------------
CREATE VIEW ligaTotalEmpleado AS
SELECT
        e.nombreEmpleado AS 'Empleado',
        SUM(v.liga) AS 'La liga total'
FROM empleado AS e
INNER JOIN venta AS v
ON e.idEmpleado = v.idEmpleadoV;

-- ------------------------------------------------------
-- Visualizar las citas que tiene agendadas cada cliente
-- ------------------------------------------------------
CREATE VIEW citasCliente AS
SELECT
        c.nombreCliente AS 'Cliente',
        GROUP_CONCAT(ci.fechaCita SEPARATOR ' | ') AS 'Fecha(s) para cita(s)'
FROM cliente AS c
INNER JOIN cita AS ci
ON c.idCliente = ci.idClienteC
GROUP BY c.idCliente;
```

Figura 12 Declaración de vistas (3).

2.6 PROCEDIMIENTOS ALMACENADOS

Los procedimientos que se consideran importantes son creados, siendo estos los correspondientes a la creación, lectura, actualización y eliminación de un empleado. Entonces se usan las sentencias INSERT, SELECT, UPDATE y DELETE, como se muestra en las Figuras más adelante.

```
-- ------------------------------------------------------
-- Procedimientos CRUD para empleados
-- ------------------------------------------------------
USE ocochosBarberia;

-- ------------------------------------------------------
-- Procedimiento Agregar
-- ------------------------------------------------------
DELIMITER //
CREATE PROCEDURE agregarEmpleado (IN idE INT,
                                  IN nombreE VARCHAR(40),
                                  IN cedulaE VARCHAR(20))
BEGIN
INSERT INTO empleado
VALUES
    (idE, nombreE, cedulaE);
END;
//

-- ------------------------------------------------------
-- Procedimiento Consultar
-- ------------------------------------------------------
DELIMITER //
CREATE PROCEDURE empleadoPorId (IN idE INT)
BEGIN
SELECT
        nombreEmpleado AS 'Empleado',
        cedula AS 'Identificación'
FROM empleado
WHERE idEmpleado = idE;
END;
//
```

Figura 13 Creación de SP (1).

```
-- ------------------------------------------------------
-- Procedimiento Actualizar
-- ------------------------------------------------------
DELIMITER //
CREATE PROCEDURE actualizarEmpleado (IN idE INT,
                                     IN nombreE VARCHAR(40),
                                     IN cedulaE VARCHAR(20))
BEGIN
UPDATE empleado
SET
    nombreEmpleado = nombreE,
    cedula = cedulaE
WHERE idEmpleado = idE;
END;
//

-- ------------------------------------------------------
-- Procedimiento Borrar
-- ------------------------------------------------------
DELIMITER //
CREATE PROCEDURE borrarEmpleadoPorId (IN idE INT)
BEGIN
DELETE FROM empleado WHERE idEmpleado = idE;
END;
//
```

Figura 14 Creación de SP (2).

2.7 TRIGGERS

Para crear los 4 triggers propuestos, se hizo necesaria la creación de dos tablas adicionales, ya que estos, iban a estar pendientes de cuándo se agrega, elimina o actualiza un empleado, además de cuándo se agrega un nuevo proveedor. Así, se generan nuevas tablas con las sentencias que se presentan a continuación.

```
-- ------------------------------------------------------
-- Triggers CUD para empleados y C para proveedores
-- ------------------------------------------------------
USE ocochosBarberia;

-- ------------------------------------------------------
-- Creación de nueva tabla para registrar acciones CUD
-- sobre la tabla empleado
-- ------------------------------------------------------
CREATE TABLE controlCambiosEmpleado (
    usuario VARCHAR(100),
    accion VARCHAR(40),
    fecha DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- ------------------------------------------------------
-- Creación de nueva tabla para registrar acción C sobre
-- la tabla proveedor
-- ------------------------------------------------------
CREATE TABLE controlCambiosProveedor (
    usuario VARCHAR(100),
    accion VARCHAR(40),
    fecha DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Figura 15 Nuevas tablas para almacenar información de Triggers.

Luego, la declaración de los triggers corresponde con las sentencias que se muestran en las Figuras más adelante. En particular, en el trigger de actualización de empleado, se usa un condicional que me agrega a la tabla de control una acción dependiendo de si es válida o no, ya que no es posible cambiar la cédula de alguien, pero si su nombre.

A pesar de que la BD si permite cambiar la cédula, en el control está acción queda invalida, por lo que debe volverse a modificar esta información. La lógica para evitar esto puede ser añadida posteriormente con sobreescritura o algún método dentro del programa Java conectado a la BD.

```
-- ------------------------------------------------
-- Creando Trigger para registro Agregar Empleado
-- ------------------------------------------------
DELIMITER //
CREATE TRIGGER insEmpleado
AFTER INSERT ON empleado
FOR EACH ROW
BEGIN
INSERT INTO controlCambiosEmpleado
VALUES (USER(), 'AGREGAR', NOW());
END;
//

-- ------------------------------------------------
-- Creando Trigger para registro Eliminar Empleado
-- ------------------------------------------------
DELIMITER //
CREATE TRIGGER delEmpleado
AFTER DELETE ON empleado
FOR EACH ROW
BEGIN
INSERT INTO controlCambiosEmpleado
VALUES (USER(), 'ELIMINAR', NOW());
END;
//
```

Figura 16 Creación de Triggers (1).

```
-- ------------------------------------------------
-- Creando Trigger para registro Actualizar Empleado
-- ------------------------------------------------
DELIMITER //
CREATE TRIGGER updEmpleado
AFTER UPDATE ON empleado
FOR EACH ROW
BEGIN
IF OLD.cedula <> NEW.cedula AND OLD THEN
    INSERT INTO controlCambiosEmpleado
    VALUES (USER(), 'INVALIDO', NOW());
ELSEIF OLD.nombreEmpleado <> NEW.cédula THEN
    INSERT INTO controlCambiosEmpleado
    VALUES (USER(), 'ACTUALIZAR', NOW());
END IF;
END;
//

-- ------------------------------------------------
-- Creando Trigger para registro Agregar Proveedor
-- ------------------------------------------------
DELIMITER //
CREATE TRIGGER insProveedor
AFTER INSERT ON proveedor
FOR EACH ROW
BEGIN
INSERT INTO controlCambiosProveedor
VALUES (USER(), 'AGREGAR', NOW());
END;
//
```

Figura 17 Creación de Triggers (2).

3. POBLACIÓN DE LA BD

La población de la base de datos se realiza gracias a la herramienta javaFaker, la cual permite a partir de unos campos predefinidos, generar cualquier combinación de registros deseados. Así, se crea una estructura de paquetes en el proyecto de Java que hará la conexión.

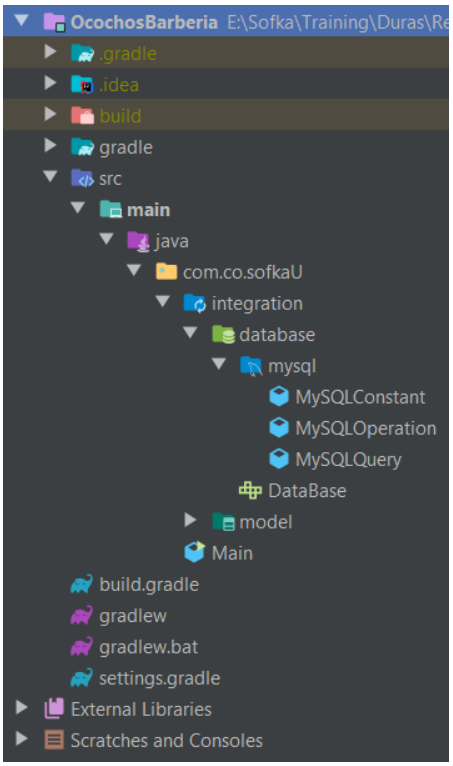

Figura 18 Estructura del proyecto de conexión.

La estructura del proyecto, muestra dos paquetes principales dentro de la integración, database y model. En database se aloja, por ahora, la lógica de conexión para MySQL, donde a partir de un contrato o base, está se implementa. Luego, en model se encuentran todas las clases que corresponden a una tabla en la BD, para así poder acceder fácilmente a sus campos.

Cada modelo cuenta con sus atributos como columnas en la tabla, y su correspondiente constructor.

Se declaran algunas constantes necesarias para la conexión en la clase MySQLConstant, y los métodos necesarios para realizar operaciones en la clase MySQLOperation. Luego para la aplicación puntal, en la clase MySQLQuery se realiza la implementación de todos los métodos para interactuar y poblar la BD. Esto se ve reflejado en las Figuras más adelante que toman como ejemplo el modelo Proveedor.

```java
package com.co.sofkaU.integration.database;

import java.sql.SQLException;

public interface DataBase {

    void configureDBConnection();

    void executeSelectStatement();

    void executeInsertStatement();

    void closeConnection();

    void printResultSet() throws SQLException;
}
```

Figura 19 Contrato para crear conexión de BD.

```java
package com.co.sofkaU.integration.database.mysql;

public class MySQLConstant {

    public static final String MY_SQL_JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";

    public static final String CONNECTION_STRING = "jdbc:mysql://%s/%s?user=%s&password=%s";
}
```

Figura 20 Constantes necesarias para la conexión.

Se muestran las únicas modificaciones hechas al método de conexión visto en clase, donde se agregan dos métodos nuevos para realizar consultas, así.

```java
@Override
public void executeSelectStatement() {
    try {
        configureDBConnection();
        resultSet = statement.executeQuery(sqlStatement);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}

@Override
public void executeInsertStatement() {
    try {
        configureDBConnection();
        int resultSet = statement.executeUpdate(sqlStatement);
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```

Figura 21 Métodos para consultas SELECT e INSERT.

Luego, se configura la conexión, mostrando solo las modificaciones hechas al método visto en clase, así.

```java
public class MySQLQuery {
    private static final String SOCKET = "localhost:3306";
    private static final String DBNAME = "ocochosBarberia";
    private static final String USER = "ococho";
    private static final String PASSWORD = "hola123";
    private static final MySQLOperation mySQLOp = new MySQLOperation();
    private static final Faker faker = new Faker();

    public static final String INSERT_PROVEEDOR =
            "INSERT INTO proveedor (nombreProveedor, telefonoProveedor) VALUES ('%s', '%s');";
```

Figura 22 Parámetros de configuración y consulta a realizar sin formato.

Así, luego de tener la conexión como la vista en clase, se declaran los métodos necesarios para realizar la inserción, así.

```
public static void insert(String insert) {
    mySQLOp.setSqlStatement(insert);
    mySQLOp.executeInsertStatement();
}

public static void insertProveedor(String insert) {
    for (int i = 1; i <= 50 ; i++) {
        Proveedor proveedor =
                new Proveedor(faker.name().firstName(),  telefonoProveedor: "321" + faker.number().digits( count: 7));
        insert(String.format(insert, proveedor.getNombreProveedor(), proveedor.getTelefonoProveedor()));
    }

}
```

Figura 23 Métodos para insertar Proveedores.

Y se llama desde la clase Main para ejecutar e insertar los datos en la BD.

```
public class Main {

    public static void main(String[] args) throws SQLException {
        openConnection();
        insertProveedor(INSERT_PROVEEDOR);
        closeConnection();
    }
}
```

Figura 24 Llamado desde main del método anterior.

Este proceso se repite para cada tabla, por lo que, de ser necesario, referirse al proyecto directamente para verificar la sintaxis y funcionamiento.

Finalmente, se muestra el resultado de las inserciones para está tabla, donde los registros siguen hasta alcanzar los 50.

idProveedor	nombreProveedor	telefonoProveedor
1	Edmundo	3212118034
2	Kandy	3212366744
3	Torri	3219290587
4	Frank	3211323820
5	Elda	3218502476
6	Solomon	3213835283
7	Lynne	3217883264
8	Rory	3213553125
9	Eliz	3213414415
10	Brenna	3212130054

Figura 25 Registros tabla Proveedor.

4. CONCLUSIONES Y CONSIDERACIONES FINALES

El alcance del proyecto fue justo para el tiempo brindado, sin embargo, se considera un poco repetitivo al momento de la ejecución de ciertos conceptos, por lo que reducir este aspecto, podría hacer que se abarcaran otro tipo de temáticas adicionales.

El conocimiento obtenido a partir del desarrollo de la actividad es amplio y útil, dejando claros los conceptos que abarcan el diseño y la manipulación de una base de datos desde Java. Además, se pudo explorar nuevas herramientas y métodos como JavaFaker y Gradle.

La participación de los compañeros de training evidencia que no solo fue una obtención de conocimientos individual, sino también grupal.

REFERENCIAS

Rand. (2023). *Barbería Rand*. Obtenido de Barbería Rand: https://www.barberiarand.com/