INFORME

DE AVANCES

Proyecto Final: Importación de Datos a MySQL

A. Descripcion del Proyecto

Este documento describe el proceso de importación de datos desde un archivo CSV hacia una base de datos MySQL. Se utilizará la tabla CLIENTE como ejemplo y se mostrarán los pasos a seguir para cargar los datos desde el archivo CLIENTE.csv a la base de datos.

B. Archivos Incluidos

- **CLIENTE.csv**: Archivo CSV que contiene los datos de los clientes.
- **Script SQL**: Archivo .sql que contiene el esquema de la tabla CLIENTE.
- **Este documento PDF**: Explicación detallada de los pasos de importación y enlaces a los archivos en el repositorio de GitHub.

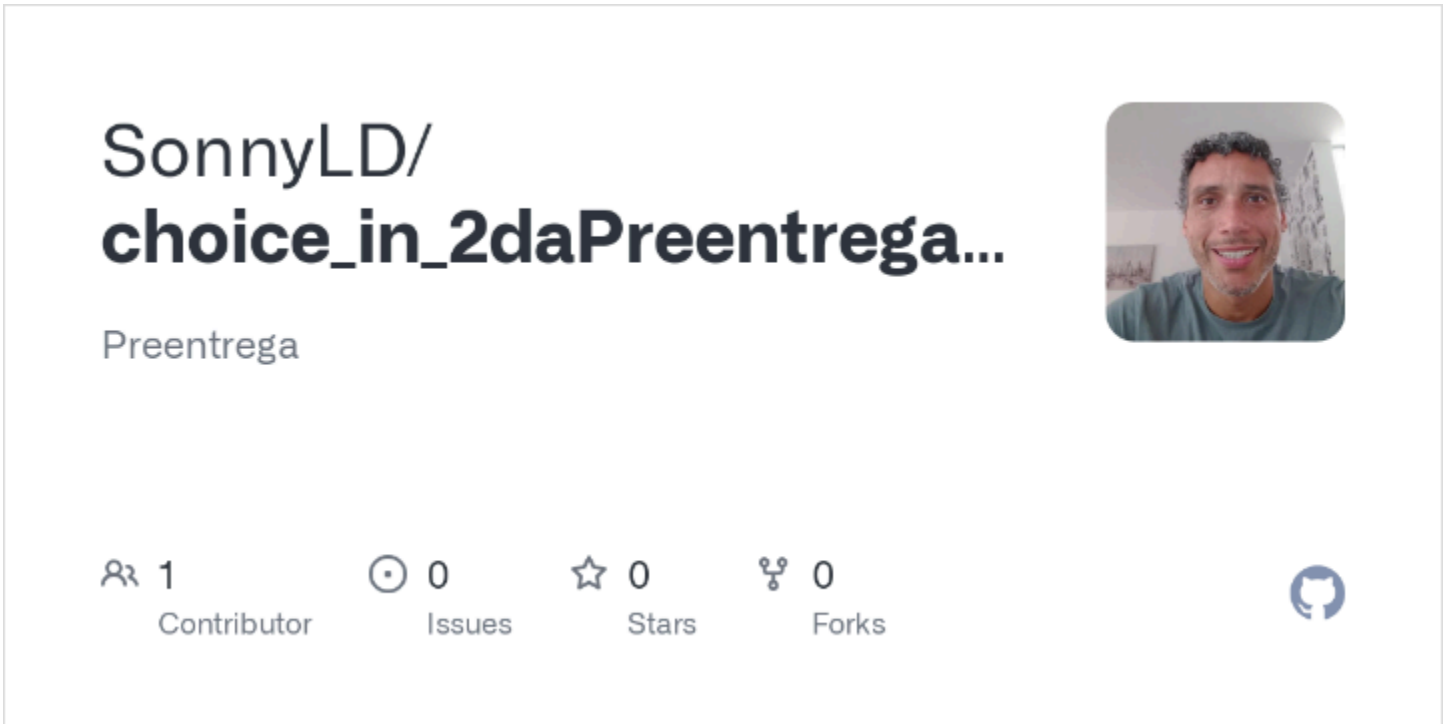
C. Creación de la Tabla CLIENTE

Script de Creación de la Tabla

Antes de importar los datos, asegúrate de haber creado la tabla CLIENTE. A continuación, se muestra el script SQL utilizado para crear la tabla:

```sql
CREATE TABLE CLIENTE(
cliente_id INT NOT NULL AUTO_INCREMENT,
nombre VARCHAR(100),
apellido VARCHAR(100),
documento_identidad VARCHAR(30),
email VARCHAR(200),
teléfono VARCHAR(50),
categoría_id INT,
PRIMARY KEY (cliente_id),
FOREIGN KEY (categoría_id) REFERENCES CLIENTE_CATEGORIA(categoria_id)
) COMMENT 'INFORMACION BASICA DEL CLIENTE';
```

El archivo SQL completo con la estructura de la base de datos se encuentra en el repositorio de GitHub, accesible en el siguiente enlace:



D. Importación de Datos desde Archivo CSV

A continuación se detalla el proceso para importar el archivo CLIENTE.csv a la tabla CLIENTE en MySQL.

Método 1: Usando el Comando LOAD DATA INFILE:

- **Abre MySQL Workbench** y asegúrate de estar conectado a tu base de datos.

- Ejecuta el siguiente comando SQL para cargar los datos desde el archivo CSV a la tabla CLIENTE:

```
-- Datos de CLIENTE(HAY UN ARCHIVO .CSV)
-- LOAD DATA LOCAL INFILE 'C:\\Users\\SONNY\\Desktop\\Curso de SQL\\primera_pre_entrega\\CLIENTE.csv'
-- INTO TABLE CLIENTE
-- FIELDS TERMINATED BY ','
-- LINES TERMINATED BY '\n'
-- IGNORE 1 ROWS
-- (cliente_id, nombre, apellido, documento_identidad, email, telefono, categoria_id);
```

Verifica los Datos: Después de ejecutar el comando, verifica que los datos se hayan cargado correctamente en la tabla:

```
SELECT * FROM choice_in.cliente;
```

Confirmación: Si todo se ha realizado correctamente, los datos del archivo CLIENTE.csv estarán presentes en la tabla CLIENTE.

A. Método 2: Importación Gráfica en MySQL Workbench

1. **Abre MySQL Workbench**.
2. Selecciona tu base de datos (choice_in).
3. Haz clic derecho sobre la base de datos y selecciona **"Table Data Import Wizard"**.
4. **Selecciona el Archivo CSV**: Busca y selecciona el archivo CLIENTE.csv en tu computadora.
5. **Configura las Opciones de Importación**:
- Asegúrate de que las columnas estén correctamente delimitadas por comas.
- Especifica que la primera fila del archivo contiene los nombres de las columnas.
6. **Selecciona la Tabla CLIENTE** como destino para los datos.
7. **Finaliza la Importación**: Sigue los pasos del asistente de importación para completar el proceso.

B. Enlaces de GitHub

- Para acceder a los archivos utilizados en este proyecto, como el script SQL y el archivo CSV, visita el siguiente enlace a GitHub:

SonnyLD/
choice_in_2daPreentrega...

Preentrega

1 Contributor 0 Issues 0 Stars 0 Forks

C. Archivos Incluidos en el Repositorio:

- **CLIENTE.csv**: Contiene los datos de los clientes.
- **script.sql**: Script para la creación de las tablas.
- **Instrucciones en PDF**: Este documento en formato PDF, que describe el proceso paso a paso.

D. Paso a Paso para la Importación en MySQL Workbench

El siguiente es un resumen visual del proceso de importación en MySQL Workbench usando la opción gráfica:

1. **Accede a la Herramienta de Importación.**
2. **Selecciona el Archivo CSV.**
3. **Configura los Delimitadores y las Columnas.**
4. **Confirma la Tabla de Destino.**
5. **Verifica la Importación de Datos.**

Conclusiones

Este documento detalla los pasos para importar un archivo CSV en MySQL y proporciona tanto un método basado en comandos como una guía para realizarlo de manera gráfica. Además, se incluyen los enlaces a los archivos en GitHub para referencia y uso en futuros proyectos.

Ejemplo de Contenido del Archivo CSV (CLIENTE.csv)

```
-- Datos de CLIENTE
INSERT INTO CLIENTE (cliente_id, nombre, apellido, documento_identidad, email, teléfono, categoría_id) VALUES
(1, 'Benji', 'Robertshaw', 'Genderqueer', 'brobertshaw0@odnoklassniki.ru', '560-647-3621', 1),
(2, 'Erda', 'Cuxon', 'Female', 'ecuxon1@boston.com', '523-688-4175', 2),
(3, 'Terrance', 'Lissandrini', 'Male', 'tlissandrini3@yale.edu', '399-744-5095', 1);
```

Objetivo General del Trabajo

El sistema de bases de datos que se ha diseñado tiene como objetivo gestionar un servicio de reserva de paquetes turísticos, el cual incluye clientes, proveedores, pagos, reservas, vuelos, alojamientos y actividades. Este sistema permite almacenar y procesar de manera eficiente la información relacionada con las reservas de paquetes turísticos, asegurando la integridad de los datos mediante el uso de claves foráneas y triggers. Además, se emplean funciones, procedimientos almacenados y vistas para optimizar la consulta y manipulación de los datos.

Funciones

Las funciones en este trabajo tienen como propósito facilitar el cálculo y la consulta de datos específicos relacionados con las reservas y las opiniones de los clientes.

1. **Función TotalPagosPorReserva**

- **Objetivo:** Calcular el total de pagos realizados para una reserva específica.
- **Datos involucrados:** La tabla PAGO y la columna reserva_id de la tabla RESERVA.
- **Descripción:** Esta función recibe un reserva_id como parámetro y devuelve la suma de todos los pagos asociados con esa reserva. Esto permite conocer el monto total pagado hasta el momento para una reserva determinada.

```sql
-- Función para obtener el total de pagos por reserva
DELIMITER //
CREATE FUNCTION TotalPagosPorReserva(reservaId INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10, 2);
    SELECT SUM(monto) INTO total
    FROM PAGO
    WHERE reserva_id = reservaId;
    RETURN IFNULL(total, 0);
END //
DELIMITER ;
```

2. Función CalificacionPromedioCliente

- **Objetivo: Obtener la calificación promedio que ha dado un cliente a los servicios contratados.**
- **Datos involucrados: La tabla OPINION_CALIFICACION y las columnas cliente_id y calificación.**
- **Descripción: Recibe un cliente_id como parámetro y calcula el promedio de todas las calificaciones otorgadas por ese cliente. Esto puede ser útil para evaluar la satisfacción del cliente con los distintos servicios ofrecidos (vuelos, alojamientos, actividades).**

```sql
-- Funcion para la calificacion promedio del cliente
DELIMITER //
CREATE FUNCTION CalificacionPromedioCliente(clienteId INT)
RETURNS DECIMAL(3, 2)
DETERMINISTIC
BEGIN
    DECLARE promedio DECIMAL(3, 2);
    SELECT AVG(calificación) INTO promedio
    FROM OPINION_CALIFICACION
    WHERE cliente_id = clienteId;
    RETURN IFNULL(promedio, 0);
END //
DELIMITER ;
```

Procedimientos Almacenados (Stored Procedures)

Los procedimientos almacenados permiten automatizar tareas recurrentes relacionadas con la gestión de reservas, pagos y actualizaciones de estado.

1. Procedimiento CrearReserva

- **Objetivo:** Crear una nueva reserva para un cliente en un paquete turístico.
- **Datos involucrados:** Las tablas RESERVA, CLIENTE y PAQUETE_TURISTICO.

- **Descripción:** Este procedimiento inserta una nueva entrada en la tabla RESERVA con el cliente_id, paquete_id y la fecha de reserva. El estado de la reserva se establece como "Pendiente" inicialmente.

```
-- -- Crear Stored Procedure
DELIMITER //
CREATE PROCEDURE CrearReserva(
    IN clienteId INT,
    IN paqueteId INT,
    IN fechaReserva DATE)
BEGIN
    INSERT INTO RESERVA (cliente_id, paquete_id, fecha_reserva, estado)
    VALUES (clienteId, paqueteId, fechaReserva, 'Pendiente');
END //
DELIMITER ;
```

2. Procedimiento ActualizarEstadoReserva

- **Objetivo:** Actualizar el estado de una reserva existente.
- **Datos involucrados:** La tabla RESERVA.
- **Descripción:** Este procedimiento recibe un reserva_id y un nuevo estado, y actualiza el estado de la reserva en la tabla RESERVA. Esto es útil para manejar cambios en el flujo de trabajo de las reservas, como "Confirmada", "Cancelada", "Pendiente", etc.

```
DELIMITER //
CREATE PROCEDURE ActualizarEstadoReserva(
    IN reservaId INT,
    IN nuevoEstado VARCHAR(50))
BEGIN
    UPDATE RESERVA
    SET estado = nuevoEstado
    WHERE reserva_id = reservaId;
END //
DELIMITER ;
```

3. Procedimiento ProcesarPago

- **Objetivo: Registrar un nuevo pago para una reserva.**
- **Datos involucrados: Las tablas PAGO y RESERVA.**
- **Descripción: Este procedimiento inserta un nuevo registro en la tabla PAGO con el reserva_id, método de pago y el monto, marcando el pago como "Completado". Es fundamental para la gestión del sistema de pagos en el proceso de reserva.**

```
DELIMITER //
CREATE PROCEDURE ProcesarPago(
    IN p_reserva_id INT,
    IN p_metodo_pago VARCHAR(50),
    IN p_monto DECIMAL(10, 2)
)
BEGIN
    INSERT INTO PAGO(reserva_id, metodo_pago, monto, estatus)
    VALUES (p_reserva_id, p_metodo_pago, p_monto, 'Completado');
END //
DELIMITER ;
```

Triggers

Los triggers permiten automatizar la ejecución de acciones específicas en respuesta a eventos en las tablas, como inserciones o actualizaciones.

1. Trigger before_insert_reserva

- **Objetivo:** Evitar que se realicen reservas con fechas en el pasado.
- **Datos involucrados:** La tabla RESERVA.
- **Descripción:** Este trigger se ejecuta antes de insertar una nueva reserva y valida que la fecha de reserva no sea anterior a la fecha actual. Si la fecha es en el pasado, lanza un error.

```
-- Creación de triggers (ejemplo)
DELIMITER $$
CREATE TRIGGER before_insert_reserva
BEFORE INSERT ON RESERVA
FOR EACH ROW
BEGIN
    IF NEW.fecha_reserva < CURDATE() THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La fecha de reserva no puede ser en el pasado.';
    END IF;
END $$
DELIMITER ;
```

2. Trigger ActualizarDisponibilidadPaquete

- **Objetivo: Actualizar automáticamente la disponibilidad de un paquete turístico después de que se realice una reserva.**
- **Datos involucrados: Las tablas RESERVA y PAQUETE_TURISTICO.**
- **Descripción: Este trigger se activa después de insertar una nueva reserva y reduce la disponibilidad del paquete turístico reservado en uno.**

```
DELIMITER //
CREATE TRIGGER ActualizarDisponibilidadPaquete
AFTER INSERT ON RESERVA
FOR EACH ROW
BEGIN
    UPDATE PAQUETE_TURISTICO
    SET disponibilidad = disponibilidad - 1
    WHERE paquete_id = NEW.paquete_id;
END //
DELIMITER ;
```

Vistas

Las vistas simplifican el acceso a los datos combinados de varias tablas para facilitar consultas específicas.

1. Vista ReservasActivas

- **Objetivo:** Mostrar las reservas que están confirmadas, junto con el nombre del cliente y el nombre del paquete.
- **Datos involucrados:** Las tablas RESERVA, CLIENTE y PAQUETE_TURISTICO.
- **Descripción:** Esta vista combina datos de RESERVA, CLIENTE y PAQUETE_TURISTICO para mostrar una lista de las reservas confirmadas junto con la información relevante del cliente y el paquete turístico.

```
-- Crear vistas
CREATE VIEW ReservasActivas AS
SELECT r.reserva_id, c.nombre, c.apellido, p.nombre_paquete, r.fecha_reserva
FROM RESERVA r
JOIN CLIENTE c ON r.cliente_id = c.cliente_id
JOIN PAQUETE_TURISTICO p ON r.paquete_id = p.paquete_id
WHERE r.estado = 'Confirmada';
```

2. Vista OpinionesPorCliente

- **Objetivo:** Mostrar las opiniones y calificaciones dadas por los clientes a los servicios contratados.
- **Datos involucrados:** Las tablas OPINION_CALIFICACION y CLIENTE.
- **Descripción:** Esta vista muestra la relación entre el cliente y las opiniones que ha emitido, detallando la entidad calificada (vuelo, alojamiento o actividad), el comentario y la calificación.

```sql
CREATE VIEW OpinionesPorCliente AS
SELECT o.opinion_id, c.nombre, c.apellido, o.tipo_entidad, o.comentario, o.calificación
FROM OPINION_CALIFICACION o
JOIN CLIENTE c ON o.cliente_id = c.cliente_id;
```

Conclusión

Este sistema proporciona una arquitectura robusta para la gestión de paquetes turísticos, cubriendo la administración de clientes, reservas, pagos, proveedores y opiniones. Las funciones, procedimientos almacenados, triggers y vistas aseguran la integridad de los datos y facilitan las operaciones comunes, como la creación de reservas, el procesamiento de pagos y la actualización automática de disponibilidades.