Guia para activar el proyecto de base de Datos

Por Torres Rodrigo Comision nª50055

1.Creación de la DB

Paso 1: Conéctese a la base de datos o selecciona la base de datos donde deseas ejecutar los comandos.

Use proyect_schema;

Paso 2: Crea la tabla `clients`.

Paso 3: Crea la tabla `cobros`.

Paso 4: Crea la tabla `consumibles`

1.Creación de la DB

Paso 5: Crear la tabla `equipos`.

Paso 6: Crear la tabla `pedido_proveedores`.

Paso 7: Crear la tabla `pedidos`.

Paso 8: Crear la tabla `presupuestos`.

1.Creación de la DB

Paso 9: Crear la tabla `proveedores`.

Paso 10: Crear la tabla `proyectos`.

Paso 11: Crear la tabla `servicio`.

Paso 12: Crear la tabla `tareas`.

1.Creación de la DB

Paso 13: Crear la tabla `ventas`

Paso 14: Crear la tabla `horarios`.

2. Agregado de Claves Foráneas

Paso 1: Agregar la clave foránea en `clients`.

Paso 2: Agregar la clave foránea en `cobros`.

Paso 3: Agregar la clave foránea en `proveedores`.

Paso 4: Agregar la claves foráneas en `pedidos`.

3. Creación de vistas

Paso 1: Creación de Vista `view_clients`.

Paso 2: Creación de Vista `view_cobros`.

Paso 3: Creación de Vista `view_consumibles`.

Paso 4: Creación de Vista `view_equipos`.

Paso 5: Creación de Vista `view_presupuestos`.

Paso 6: Creación de Vista `view_pedidos`.

Paso 7: Creación de Vista `view_proveedores`.

Paso 8: Creación de Vista `view_pedido_proveedores`.

3. Creación de vistas

Paso 9: Creación de Vista `view_proyectos`.

Paso 10: Creación de Vista `view_servicio`.

Paso 11: Creación de Vista `view_tareas`.

Paso 12: Creación de Vista `view_ventas`

4.INSERTS

Realice la instalación de los insert globalmente

5.Triggers

Realiza los siguientes pasos para agregar funcionalidades a la tabla servicio:

1. Agregar Columnas:
   - Se agrega una columna llamada estado a la tabla servicio con el tipo de datos TINYINT(1) que representa el estado del servicio (activo o inactivo). Se establece un valor predeterminado de 1, donde 1 significa activo y 0 inactivo.
   - Se agregan dos columnas adicionales: updated_by (tipo VARCHAR(50)) y updated_at (tipo TIMESTAMP). updated_by guarda información sobre quién realizó la última actualización, y updated_at almacena la fecha y hora de la última actualización. Se asignan valores predeterminados para ambas columnas.

2. Crear Vista Lógica:

   - Se crea una vista lógica llamada view_servs_act que selecciona ciertos campos de la tabla servicio donde el estado es igual a 1 (activo).

5.Triggers

3. Crear Triggers:

A. Trigger Before (before_actualizar_servicio):

- Se crea un trigger que se activa antes de una actualización en la tabla servicio.
- Asigna automáticamente el valor del usuario actual a updated_by.
- Establece la marca de tiempo actual en updated_at.
- Actualiza el campo estado con su propio valor.

B. Trigger After (after_actualizar_servicio):

- Se crea un trigger que se activa después de una actualización en la tabla servicio.
- Inserta un registro en la tabla logs registrando la acción de actualización del servicio, el usuario que realizó la acción, la fecha y la hora.

5.Triggers

C. Creación de la tabla Bitacora (logs):

- Se crea una tabla llamada logs que almacena información sobre las acciones realizadas en la base de datos.
- Contiene columnas como id, accion, usuario, fecha, y hora.

5.Triggers

D. Triggers para la Tabla de Bitácora:

- Se crean dos triggers, uno antes (before_insertar_producto) y otro después (after_insertar_producto) de insertar un registro en la tabla logs.
- El trigger antes de la inserción asigna automáticamente el usuario actual, la fecha y la hora.
- El trigger después de la inserción registra la acción de insertar un producto, el usuario, la fecha y la hora.

E. Triggers para la Tabla de Bitácora:

- Se crean dos triggers, uno antes (before_insertar_producto) y otro después (after_insertar_producto) de insertar un registro en la tabla logs.
- El trigger antes de la inserción asigna automáticamente el usuario actual, la fecha y la hora.
- El trigger después de la inserción registra la acción de insertar un producto, el usuario, la fecha y la hora.

6.Funciones

Este script realiza los siguientes pasos para definir dos funciones en la base de datos:

Función `insertar_cliente`:

- Objetivo:
  - Insertar un nuevo cliente en la tabla `clients` con los parámetros proporcionados.
  - Obtener el ID correspondiente al nuevo cliente.
- Pasos Detallados:
  - Se define una función llamada `insertar cliente` que acepta parámetros como `p_nombre`, `p_email`, `p_numero`, `p_direccion`, y `p_id_pedidos`.
  - Dentro de la función, se declara una variable `nuevo_id_cliente` para almacenar el nuevo ID del cliente.
  - Se realiza la inserción de los datos en la tabla `clients` utilizando los parámetros proporcionados.
  - Se obtiene el último ID insertado en la tabla `clients` y se asigna a la variable `nuevo id cliente`.
  - La función devuelve el nuevo ID del cliente.

6.Funciones

Función `insertar_proveedor`:

- Objetivo:
  - Insertar un nuevo proveedor en la tabla `proveedores` con los parámetros proporcionados.
  - Obtener el ID correspondiente al nuevo proveedor.
- Pasos Detallados:
  - Se define una función llamada `insertar_proveedor` que acepta parámetros como `p_nombre`, `p_cuit`, `p_numero`, `p_email`, y `p_id_pedido_proveedores`.
  - Dentro de la función, se declara una variable `nuevo_id_proveedor` para almacenar el nuevo ID del proveedor.
  - Se realiza la inserción de los datos en la tabla `proveedores` utilizando los parámetros proporcionados.
  - Se obtiene el último ID insertado en la tabla `proveedores` y se asigna a la variable `nuevo_id_proveedor`.
  - La función devuelve el nuevo ID del proveedor.

6.Funciones

Cada función mejora la modularidad y reutilización del código al encapsular la lógica de inserción y obtención de ID para clientes y proveedores, respectivamente

7.Store Procedure

Este script realiza las siguientes acciones:

1. Creación del primer stored procedure (`OrdenarTabla`):
   - Objetivo:
     - Ordenar una tabla específica según un campo y un criterio de ordenamiento.
   - Pasos Detallados:
     - Se define un procedimiento almacenado llamado `OrdenarTabla` que acepta tres parámetros: `tabla_nombre`, `campo_orden`, y `ordenamiento`.
     - Dentro del procedimiento, se construye una consulta dinámica utilizando la función `CONCAT` para formar la instrucción SQL de selección con el nombre de la tabla, el campo por el cual ordenar y el criterio de ordenamiento.
     - Se prepara y ejecuta la consulta dinámica con la instrucción `PREPARE stmt FROM @query;` y `EXECUTE stmt;`.
     - Se libera la consulta preparada con `DEALLOCATE PREPARE stmt;`.
     - El procedimiento está delimitado por `DELIMITER //`.

7.Store Procedure

2. Creación del segundo stored procedure (`ActualizarRegistros`):
   - Objetivo:
     - Actualizar registros en una tabla específica según una condición.
   - Pasos Detallados:
     - Se define un procedimiento almacenado llamado `ActualizarRegistros` que acepta cuatro parámetros: `tabla_nombre`, `campo_actualizar`, `nuevo_valor`, y `condicion`.
     - Dentro del procedimiento, se construye una consulta dinámica utilizando la función `CONCAT` para formar la instrucción SQL de actualización con el nombre de la tabla, el campo a actualizar, el nuevo valor y la condición.
     - Se prepara y ejecuta la consulta dinámica con la instrucción `PREPARE stmt FROM @query;` y `EXECUTE stmt;`.
     - Se libera la consulta preparada con `DEALLOCATE PREPARE stmt;`.
     - El procedimiento está delimitado por `DELIMITER //`.

7.Store Procedure

3. Llamadas a los stored procedures:

- Se realiza una llamada al procedimiento `OrdenarTabla` para ordenar la tabla 'clients' por el campo 'nombre' de manera ascendente.
- Se realiza una llamada al procedimiento `ActualizarRegistros` para actualizar el campo 'direccion' en la tabla 'clients' para el registro con id = 1, estableciendo la nueva dirección como 'Nueva Direccion'.