Universidad Autónoma de Baja California

Facultad de ciencias químicas e Ingeniería

Plan de Ingeniero en Software y tecnologías emergentes

Bases de datos (351)

Proyecto Final “Diseño, Creación e Implementación de un Modelo de Base de Datos Relacionales”

Docente:

Sukey Sayonara Nakasima Lopez

Participante(es):

Héctor Miguel Macías Baltazar (1272124)

Luis Eduardo Galindo Amaya (1274895)

Tijuana , 2 dic 2023

1. Introducción

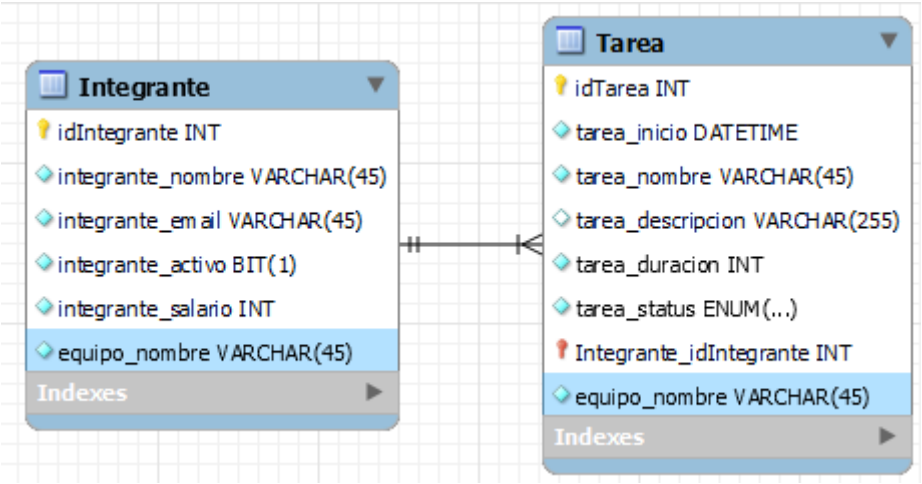
A lo largo de este proyecto, se abordarán desafíos comunes en el diseño de bases de datos, como la gestión de relaciones entre entidades, la optimización del rendimiento y la adaptabilidad a futuras expansiones. Además, se examinarán herramientas y tecnologías modernas que facilitan la creación y administración efectiva de bases de datos relacionales.

1.1. Descripción

Para el proyecto final, se optó por desarrollar un sistema de gestión de proyectos que permite a los líderes de proyecto crear tareas con duraciones específicas y realizar un seguimiento de las horas trabajadas. Los usuarios se dividen en administradores y miembros del equipo. Los miembros del equipo tienen la capacidad de registrar sus horas de entrada y salida, así como marcar tareas como completadas. Por otro lado, los administradores cuentan con privilegios adicionales, como la capacidad de modificar salarios, así como crear y eliminar tareas.

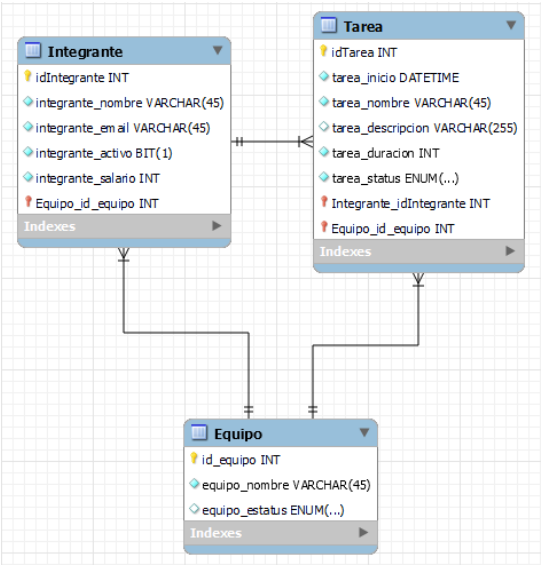
1.2. Normalización

El objetivo fundamental de la normalización es minimizar la redundancia de datos y prevenir anomalías durante las operaciones de inserción, actualización y eliminación. A medida que se avanza en las formas normales, se logra una mayor eficiencia en el almacenamiento y gestión de la información, asegurando la coherencia y facilitando el mantenimiento a lo largo del tiempo.
El modelo inicial contaba con dos relaciones: integrante y tarea. Este modelo ya se encontraba en la primera forma normal, pues todos los atributos de cada tupla contenían un solo valor (valores atómicos indivisibles) por columna.



Modelo en primera forma normal.

Sin embargo, claramente existía una dependencia parcial entre integrante y tarea. Ambas relaciones tenían el mismo atributo *equipo_nombre*. De acuerdo con la segunda formal normal, cada atributo no clave, depende de forma funcional completa de la llave primaria. Si hay atributos que dependen solo en parte del atributo clave, entonces esa parte, formará otra tabla. Dicho esto, para eliminar la dependencia del atributo *equipo_nombre* de la claves primarias de ambas relaciones, se creó una tabla equipo.



Modelo en segunda forma normal.

Finalmente, para cumplir con la tercera forma normal, dividimos el resto de relaciones en tantas tablas fueran necesarias para evitar dependencias transitivas. El diagrama del modelo final se muestra en el siguiente apartado.

1.3. Diagrama

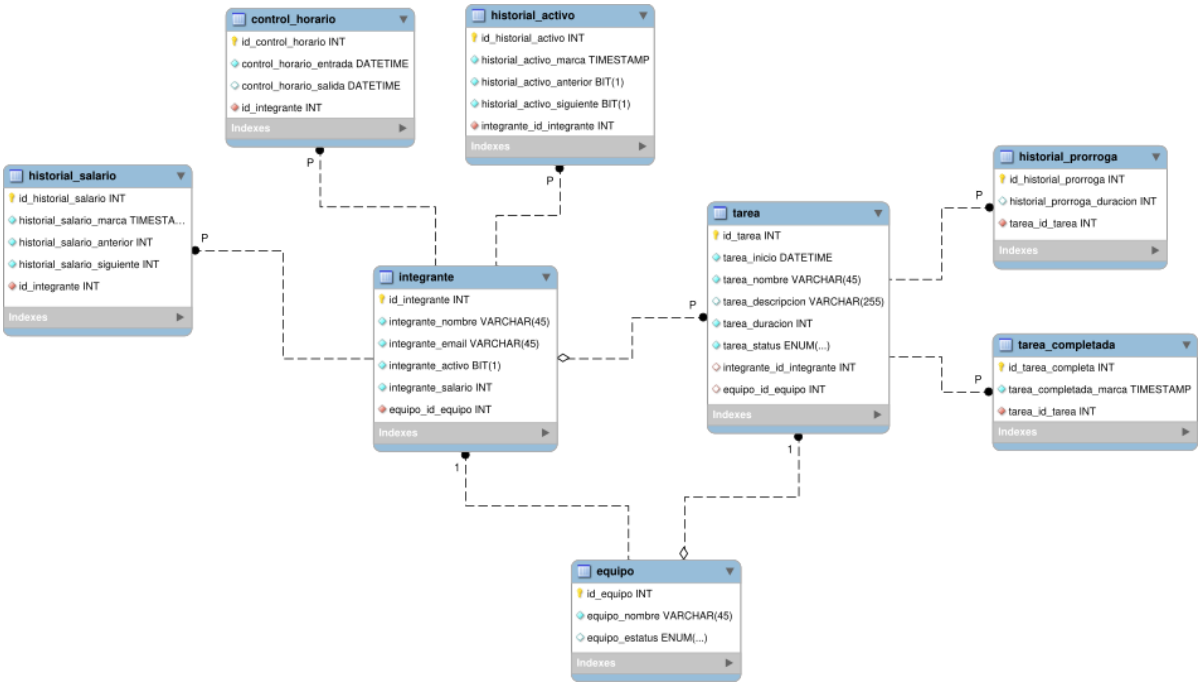


Diagrama del modelo final.

2. Entidades del sistema

2.1. Integrantes

Los integrantes del equipo representan a las personas que trabajan en el proyecto contiene datos como nombre, correo electrónico, estado activo, salario, y referencia al equipo al que pertenecen.

3. Equipos

Representa los grupos de trabajo dentro del proyecto, por ejemplo, pueden haber grupos de diseño, de desarrollo, etc. Almacena información sobre los equipos de trabajo, como su nombre y estado (activo o inactivo).

3.1. Tarea

Almacena información sobre las tareas de los proyectos, como la fecha de inicio, nombre, descripción, duración, estado, y referencias a los integrantes y equipos relacionados.

3.2. Historial salario

Guarda el historial de cambios en los salarios de los integrantes así como la fecha en la que se realiza el cambio.

3.3. Control horario

Gestiona el control de horario de los integrantes registrando la entrada y salida, además de tener una referencia al integrante asociado.

3.4. Historial prórroga

Registra la cantidad de días que se agregan para terminar una tarea.

3.5. Tarea completada

Marca el tiempo en el cual se marca una tarea como terminada.

3.6. Historial activo

Registra el historial de actividad de los miembros del equipo, estado anterior y siguiente.

4. Relaciones entre los modelos

Los tres modelos principales de la base de datos son las tareas, los integrantes y los equipos. Los demás modelos de la base de datos emergen de estas tablas.

4.1. Equipos

Representa los grupos de trabajo dentro del proyecto. Un equipo puede tener muchas tareas, ya que un equipo tiene varios integrantes. Por lo tanto las relaciones son:

- **De uno a muchos con Tareas**: Cada equipo está asociado con muchas tareas. Esto significa que un equipo puede trabajar en varias tareas dentro del proyecto.

- **De uno a muchos con Integrantes**: Cada equipo tiene varios integrantes. Indica que hay múltiples miembros en un equipo, y estos miembros participan en las tareas asignadas al equipo.

4.2. Tareas

Las tareas representan los trabajos que se deben realizar para poder completar el proyecto, se debe llevar un registro detallado de cada paso que se da en el proyecto:

- **De uno a muchos con historial prórroga**: Una tarea puede posponerse varias veces antes de marcarla como completada.

- **De una a una con tarea completada**: Una tarea solo puede ser completada una vez.

4.3. Integrantes

Los integrantes son la base para el desarrollo del proyecto y es indispensable que se tenga control sobre las cosas que hacen y los gastos que se atribuyen a ellos:

- **De uno a muchos con historial salario**: Se le puede subir el salario múltiples veces a un integrante durante el proyecto.

- **De uno a muchos con Control horario**: Un integrante puede trabajar varios turnos durante el proyecto, estos turnos pueden ser de duración variable.

- **De uno a muchos con historial activo:** Un integrante puede salir y volver de un proyecto, por lo que puede estar activo o inactivo.

5. Funciones

5.1. pago_total

Calcular el pago total multiplicando horas por salario

- **horas**: Este parámetro representa la cantidad de horas trabajadas.
- **salario**: Este parámetro representa el salario por hora.

```
DELIMITER %%
CREATE FUNCTION pago_total(
    horas INT,
    salario INT
)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE pago INT;
    SET pago = horas * salario;
    RETURN pago;
END
%%
```

5.2. horas_trabajadas

Obtener el número de horas trabajadas del integrante durante todo el lapso del proyecto

- **p_id_integrante**: Este parámetro toma el identificador del integrante

```
DELIMITER %%
CREATE FUNCTION horas_trabajadas(
    p_id_integrante INT
)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE total_horas INT;

    SELECT
        SUM(TIMESTAMPDIFF(hour, control_horario_entrada,
control_horario_salida))
    INTO total_horas
    FROM control_horario
    WHERE control_horario_salida IS NOT NULL
    AND id_integrante = p_id_integrante;

    RETURN total_horas;
END
%%
```

5.3. convertir_segundos_horas

Convertir segundos a horas

- **segundos**: Cantidad de segundos a convertir

```
DELIMITER %%
CREATE FUNCTION convertir_segundos_horas(
    segundos INT
)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE resultado INT;
    SET resultado = FLOOR(segundos / 3600);
    RETURN resultado;
END
%%
```

6. Procedimientos

6.1. costo_total_proyecto

Obtener el costo total del proyecto, suma todos los salarios de los integrantes por el número de horas trabajadas.

```
DELIMITER %%
CREATE PROCEDURE costo_total_proyecto()
BEGIN
    SELECT
        sum(pago_total(
            horas_trabajadas(id_integrante),
            integrante_salario
        )) as "Costo total proyecto"
    FROM integrante;
END
%%
```

6.2. salario_mas_alto_equipo

Obtener el salario más alto en el equipo. Para esto se utiliza la función AVG

- **p_id_equipo**: Este parámetro toma el identificador del equipo

```
DELIMITER %%
CREATE PROCEDURE salario_mas_alto_equipo(
    IN p_id_equipo INT
)
BEGIN
    SELECT
        MAX(integrante_salario)
    FROM integrante
    WHERE equipo_id_equipo = p_id_equipo;
END
%%
```

6.3. salario_mas_bajo_equipo

Obtener el salario más bajo en el equipo con la función MIN

- **p_id_equipo**: Este parámetro toma el identificador del equipo

```
DELIMITER %%
CREATE PROCEDURE salario_mas_bajo_equipo(
    IN p_id_equipo INT
)
BEGIN
    SELECT
        MIN(integrante_salario)
    FROM integrante
    WHERE equipo_id_equipo = p_id_equipo;
END
%%
```

6.4. calcular_promedio_salario

Calcular el promedio de salario de todos los equipos utilizando la función AVG

```
DELIMITER %%
CREATE PROCEDURE calcular_promedio_salario()
BEGIN
    SELECT AVG(integrante_salario)
    FROM integrante;
END
%%
```

6.5. marcar_tarea_completada

Marcar una tarea como completada, al marcarla como

- **p_id_tarea**: Este parámetro toma el identificador de la tarea

```
DELIMITER %%
CREATE PROCEDURE marcar_tarea_completada(
    IN p_id_tarea INT
)
BEGIN
    -- Actualizar el estado de la tarea a 'completado'
    UPDATE tarea
    SET tarea_status = 'COMPLETADO'
    WHERE id_tarea = p_id_tarea;
END
%%
```

6.6. marcar_tarea_cancelada

Marcar una tarea como cancelada

- **p_id_tarea**: Este parámetro toma el identificador de la tarea

```
DELIMITER %%
CREATE PROCEDURE marcar_tarea_cancelada(
    IN p_id_tarea INT
)
BEGIN
    -- Actualizar el estado de la tarea a 'cancelado'
    UPDATE tarea
    SET tarea_status = 'CANCELADO'
    WHERE id_tarea = p_id_tarea;
END
%%
```

6.7. agregar_prorroga

Agregar días extras a la duración de una tarea, al crearlo se crea un registro en historial prórroga.

- **p_id_tarea**: Este parámetro toma el identificador de la tarea
- **p_duracion_prorroga**: Tiempo extra que se agregara a la tarea

```
DELIMITER %%
CREATE PROCEDURE agregar_prorroga(
    IN p_id_tarea INT,
    IN p_duracion_prorroga INT
)
BEGIN
    DECLARE v_tarea_duracion_actual INT;

    -- Obtener la duración actual de la tarea
    SELECT tarea_duracion
    INTO v_tarea_duracion_actual
    FROM tarea
    WHERE id_tarea = p_id_tarea;

    -- Actualizar la duración de la tarea
    UPDATE tarea
    SET tarea_duracion = v_tarea_duracion_actual + p_duracion_prorroga
    WHERE id_tarea = p_id_tarea;
END
%%
```

6.8. control_integrante

Marcar la entrada o salida de un integrante al entrar se creará un nuevo registro y al salir se actualizará la hora de salida del registro creado esta función se encarga de determinar el tiempo que se trabaja.

- **p_id_integrante**: Este parámetro toma el identificador del integrante

```
DELIMITER %%
CREATE PROCEDURE control_integrante(
    IN p_integrante_id INT
)
BEGIN
    DECLARE v_id_control INT;

    -- Guardar la última entrada
    SELECT id_control_horario
    INTO v_id_control
    FROM control_horario
    WHERE p_integrante_id = id_integrante
    AND control_horario_salida IS NULL;

    IF v_id_control IS NULL THEN
```

```
        -- Crear un nuevo registro de control horario para la entrada
        INSERT INTO control_horario(
            id_integrante
        )
        VALUES (
            p_integrante_id
        );

        SELECT 'creado' AS mensaje;
    ELSE
        -- Actualizar el registro de control horario para la salida
        UPDATE control_horario
        SET
            control_horario_salida = CURRENT_TIMESTAMP,
            id_integrante = p_integrante_id
        WHERE id_control_horario = v_id_control;

        SELECT 'actualizado' AS mensaje;
    END IF;
END
%%
```

6.9. crear_integrante

Crear un nuevo integrante en el equipo, para crear un nuevo integrante es indispensable tener como mínimo un equipo al cual se vaya a integrar.

- **p_nombre**: Nombre del nuevo integrante del equipo.
- **p_email**: Email del nuevo integrante, debe ser único
- **p_salario**: salario por hora
- **p_equipo_id**: indicador del equipo al que se integrará

```
  DELIMITER %%
  CREATE PROCEDURE crear_integrante(
      IN p_nombre VARCHAR(45),
      IN p_email VARCHAR(45),
      IN p_salario INT,
      IN p_equipo_id INT
  )
  BEGIN
      -- Insertar un nuevo integrante en la base de datos
      INSERT INTO integrante(
          integrante_nombre,
          integrante_email

,
          integrante_salario,
          equipo_id_equipo
      )
      VALUES (
          p_nombre,
          p_email,
          p_salario,
```

```
            p_equipo_id
        );
    END
    %%
```

6.10. crear_equipo

Crear un nuevo equipo al proyecto

- **p_nombre**: Nombre del nuevo equipo.

```
DELIMITER %%
CREATE PROCEDURE crear_equipo(
    IN p_nombre VARCHAR(45)
)
BEGIN
    -- Insertar un nuevo equipo en la base de datos
    INSERT INTO equipo (
        equipo_nombre,
        equipo_estatus
    )
    VALUES (
        p_nombre,
        "activo"
    );
END
%%
```

6.11. crear_tarea

Crear una nueva tarea en el proyecto, la tarea debe ser asignada a un equipo primero antes de asignarse a un integrante.

- **p_inicio**: Fecha de inicio de la tarea
- **p_nombre**: Título de la tarea
- **p_descripcion:** Descripción detallada de la tarea a realizar
- **p_duracion**: Días que tomará realizar la tarea

```
DELIMITER %%
CREATE PROCEDURE crear_tarea(
    IN p_inicio DATETIME,
    IN p_nombre VARCHAR(45),
    IN p_descripcion VARCHAR(255),
    IN p_duracion INT
)
BEGIN
    -- Insertar una nueva tarea en la base de datos
    INSERT INTO tarea (
        tarea_inicio,
        tarea_nombre,
        tarea_descripcion,
```

```
        tarea_duracion
    )
    VALUES (
        p_inicio,
        p_nombre,
        p_descripcion,
        p_duracion
    );
END
%%
```

6.12. asignar_equipo_tarea

Asignar un equipo a una tarea existente

- **p_id_tarea**: Este parámetro toma el identificador de la tarea
- **p_equipo_id**: indicador del equipo al que se integrará

```
DELIMITER %%
CREATE PROCEDURE asignar_equipo_tarea(
    IN p_id_equipo INT,
    IN p_id_tarea INT
)
BEGIN
    -- Actualizar el equipo asignado a la tarea
    UPDATE tarea
    SET equipo_id_equipo = p_id_equipo
    WHERE id_tarea = p_id_tarea;
END
%%
```

6.13. asignar_integrante_tarea

Asignar un integrante a una tarea existente, la tarea debe estar asignada al mismo equipo al que pertenece el integrante sí no no se podría realizar la asignación.

- **p_id_integrante**: Este parámetro toma el identificador del integrante
- **p_id_tarea**: Este parámetro toma el identificador de la tarea

```
DELIMITER %%
CREATE PROCEDURE asignar_integrante_tarea(
    IN p_id_integrante INT,
    IN p_id_tarea INT
)
BEGIN
    DECLARE v_id_equipo INT;
    DECLARE v_id_equipo_integrante INT;

    -- Obtener el equipo al que pertenece la tarea
    SELECT equipo_id_equipo INTO v_id_equipo
    FROM tarea
```

```
    WHERE id_tarea = p_id_tarea;

    -- Obtener el equipo al que pertenece el integrante
    SELECT equipo_id_equipo INTO v_id_equipo_integrante
    FROM integrante
    WHERE id_integrante = p_id_integrante;

    -- Verificar si el integrante pertenece al equipo de la tarea
    IF v_id_equipo = v_id_equipo_integrante THEN
        -- Asignar la tarea al integrante
        UPDATE
            tarea
        SET
            integrante_id_integrante = p_id_integrante,
            tarea_status = 'PENDIENTE'
        WHERE
            id_tarea = p_id_tarea;

        SELECT 'Tarea asignada correctamente' AS mensaje;
    ELSE
        SELECT 'El integrante no pertenece al equipo de la tarea' AS
mensaje;
    END IF;
END
%%
```

7. Triggers

7.1. after_update_integrante_activo

Registra la actualización de estado de un integrante, si paso de activo a inactivo o viceversa.

```
DELIMITER %%
CREATE TRIGGER after_update_integrante_activo
AFTER UPDATE ON integrante
FOR EACH ROW
BEGIN
    IF NEW.integrante_activo != OLD.integrante_activo THEN
        INSERT INTO historial_activo (
            historial_activo_marca,
            historial_activo_anterior,
            historial_activo_siguiente,
            integrante_id_integrante
        ) VALUES (
            CURRENT_TIMESTAMP,
            OLD.integrante_activo,
            NEW.integrante_activo,
            NEW.id_integrante
        );
    END IF;
END;
```

```
%%
```

7.2. after_update_tarea

Marca la fecha en la que se completó una tarea en la tabla 'tarea completada'.

```
DELIMITER %%
CREATE TRIGGER after_update_tarea
AFTER UPDATE ON mydb.tarea
FOR EACH ROW
BEGIN
    IF NEW.tarea_status = 'COMPLETADO' THEN
        INSERT INTO tarea_completada (
            tarea_completada_marca,
            tarea_id_tarea
        )
        VALUES (
            CURRENT_TIMESTAMP(),
            NEW.id_tarea
        );
    END IF;
END
%%
```

7.3. before_update_duracion_tarea

Agrega el tiempo de prórroga que se agregó a la tarea antes de actualizar la duración.

```
DELIMITER %%
DROP TRIGGER IF EXISTS before_update_duracion_tarea %%
CREATE TRIGGER before_update_duracion_tarea
BEFORE UPDATE ON tarea
FOR EACH ROW
BEGIN
    IF NEW.tarea_duracion != OLD.tarea_duracion THEN
        INSERT INTO historial_prorroga (
            tarea_id_tarea,
            historial_prorroga_duracion
        )
        VALUES (
            NEW.id_tarea,
            NEW.tarea_duracion - OLD.tarea_duracion
        );
    END IF;
END
%%
```

7.4. after_update_salario

Agrega el salario anterior al historial después de actualizar el salario de un integrante.

```
DELIMITER %%
DROP TRIGGER IF EXISTS after_update_salario %%
CREATE TRIGGER after_update_salario
AFTER UPDATE ON integrante
FOR EACH ROW
BEGIN
    IF NEW.integrante_salario != OLD.integrante_salario THEN
        INSERT INTO historial_salario (
            historial_salario_anterior,
            historial_salario_siguiente,
            id_integrante
        )
        VALUES (
            OLD.integrante_salario,
            NEW.integrante_salario,
            NEW.id_integrante
        );
    END IF;
END;
%%
```

7.5. before_insert_integrante

Evita que se inserten integrantes con salario negativo antes de la inserción.

```
DELIMITER %%
DROP TRIGGER IF EXISTS before_insert_integrante %%
CREATE TRIGGER before_insert_integrante
BEFORE INSERT
ON integrante FOR EACH ROW
BEGIN
    IF NEW.integrante_salario < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El salario debe ser mayor a 0';
    END IF;
END;
%%
```

7.6. after_insert_integrante

Agrega el salario de un integrante al historial después de insertar un nuevo integrante.

```
DELIMITER %%
DROP TRIGGER IF EXISTS after_insert_integrante %%
CREATE TRIGGER after_insert_integrante
AFTER INSERT
ON integrante FOR EACH ROW
BEGIN
    INSERT INTO historial_salario (
        historial_salario_anterior,
        historial_salario_siguiente,
        id_integrante
    )
    VALUES (
        0,
        NEW.integrante_salario,
        NEW.id_integrante
    );
END
%%
```

7.7. before_delete_tarea

Previene que se eliminen tareas que ya han sido completadas.

```
DELIMITER %%
DROP TRIGGER IF EXISTS before_delete_tarea %%
CREATE TRIGGER before_delete_tarea
BEFORE DELETE ON tarea
FOR EACH ROW
BEGIN
    IF OLD.tarea_status = 'COMPLETADO' THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'No se puede eliminar una tarea completada';
    END IF;
END;
%%
```

7.8. after_delete_tarea

Elimina las entradas del historial de prórrogas cuando se elimina su tarea asociada.

```
DELIMITER %%
DROP TRIGGER IF EXISTS after_delete_tarea %%
CREATE TRIGGER after_delete_tarea
AFTER DELETE
ON tarea FOR EACH ROW
```

```
BEGIN
    DELETE FROM historial_prorroga
    WHERE tarea_id_tarea = OLD.id_tarea;
END;
%%
```

8. Vistas

8.1. vista_tareas_pendientes

Muestra las tareas pendientes junto con la información del integrante asignado.

```
CREATE OR REPLACE VIEW vista_tareas_pendientes AS
SELECT
    id_tarea,
    tarea_inicio,
    tarea_nombre,
    tarea_descripcion,
    tarea_duracion,
    tarea_status,
    integrante_nombre
FROM
    tarea
    INNER JOIN integrante
    ON integrante_id_integrante = id_integrante
WHERE
    tarea_status = 'PENDIENTE';
```

8.2. vista_tareas_canceladas

Muestra las tareas canceladas junto con la información del integrante asignado.

```
CREATE OR REPLACE VIEW vista_tareas_canceladas AS
SELECT
    id_tarea,
    tarea_inicio,
    tarea_nombre,
    tarea_descripcion,
    tarea_duracion,
    tarea_status,
    integrante_nombre
FROM
    integrante
    RIGHT JOIN tarea
    ON integrante_id_integrante = id_integrante
WHERE
    tarea_status = 'CANCELADO';
```

8.3. vista_tareas_status

Muestra todas las tareas y sus estados, incluyendo las tareas completadas.

```
CREATE OR REPLACE VIEW vista_tareas_status AS
SELECT
    *
FROM
    tarea
    RIGHT JOIN tarea_completada
    ON id_tarea_completa=id_tarea;
```

8.4. equipos_tareas_view

Muestra las tareas de cada equipo, incluyendo la información del equipo y la tarea.

```
CREATE VIEW equipos_tareas_view AS
SELECT
    e.id_equipo,
    e.equipo_nombre,
    t.id_tarea,
    t.tarea_nombre,
    t.tarea_descripcion,
    t.tarea_duracion,
    t.tarea_status
FROM equipo e
    LEFT JOIN tarea t
    ON e.id_equipo = t.equipo_id_equipo

UNION

SELECT
    e.id_equipo,
    e.equipo_nombre,
    t.id_tarea,
    t.tarea_nombre,
    t.tarea_descripcion,
    t.tarea_duracion,
    t.tarea_status
FROM tarea t
    RIGHT JOIN equipo e
    ON e.id_equipo = t.equipo_id_equipo
ORDER BY id_equipo;
```

9. Acciones comunes

9.1. Crear equipos

Los equipos permiten agrupar a los integrantes en grupos de trabajo, un grupo tiene varios integrantes pero un integrante solo puede estar en un grupo.

```
CALL crear_equipo('Equipo 1');
```

9.2. Crear integrantes

Es importante recordar que para crear un integrante es indispensable crear un equipo antes, no pueden existir los integrantes sin estar en un equipo.

```
CALL crear_equipo('Equipo 1');

CALL crear_integrante(
  'Juan Pérez',                 -- nombre
  'juan.perez@example.com',     -- correo
  50,                           -- salario
  1                             -- id del equipo
);
```

9.3. Crear tareas

La función crear_tarea se diseñó para añadir una nueva tarea a un sistema. Es importante tener en cuenta que, al cerrar una tarea, no se asigna automáticamente a ningún equipo. Por lo tanto, si se desea asignar la tarea a un miembro específico, primero se debe asignar la tarea a un equipo en el sistema.

```
CALL crear_tarea(
    '2023-11-23',
    'Desarrollar Módulo de Autenticación',
    'Implementar el sistema de autenticación para el software',
    8
);
```

9.4. Asignar tareas a equipos

Para poder asignar un equipo debemos conocer el id de ambos registros, y utilizar la función asignar_equipo_tarea ligarlos uno al otro.

```
CALL crear_equipo('Equipo 3');  -- Equipo ID = 3

CALL crear_tarea(               -- tarea ID = 1
    '2023-11-23',
    'Desarrollar Módulo de Autenticación',
    'Implementar el sistema de autenticación para el software',
    8
);

CALL asignar_equipo_tarea(3,1);
```

9.5. Asignar tareas a integrantes

Para asignar una tarea a un integrante primero debemos asignar la tarea a un equipo con la función asignar_equipo_tarea.

```
CALL crear_equipo('Equipo 3');  -- equipo ID=3


CALL crear_tarea(               -- tarea ID=5
    '2023-11-23',
    'Desarrollar Módulo de Autenticación',
    'Implementar el sistema de autenticación para el software',
    8
);

CALL asignar_equipo_tarea(3,5);
```

Una vez creada la tarea podemos asignarla a un integrante del equipo, ambos deben estar asignados al mismo equipo sí no la asignación no podrá completarse adecuadamente.

```
CALL crear_integrante(          -- integrnate ID=4
  'Juan Pérez',
  'juan.perez@example.com',
  50,
  3
);


CALL asignar_integrante_tarea(3, 5);
```

9.6. Registrar entrada y salida

Una de las funciones principales del sistema es llevar registro de las entradas y salidas de los integrantes del equipo, se puede imaginar esta función como un reloj de fichar. Al llamarse esta función por primera vez se genera un registro con la entrada y al llamarla por segunda vez se agrega el registro de salida.

```
CALL crear_integrante(          -- integrnate ID=2
     'Juan Pérez',
     'juan.perez@example.com',
     50,
     3
);

CALL control_integrante(2);
```

9.7. Actualizar salario

Para actualizar el salario de un integrante solamente debemos usar update, el modelo tiene un trigger que se encargará de registrar el historial de salarios del empleado.

```
UPDATE integrante
SET integrante_salario = 400    -- nuevo salario
WHERE id_integrante = 1;        -- id del integrante
```

9.8. Agregar más tiempo a las tareas

La función agregar_prorroga extiende el plazo original de la tarea en 10 días adicionales. Para hacer la prórroga se debe conocer el identificador de la tarea.

```
CALL crear_tarea(               -- tarea ID = 1
    '2023-11-23',
    'Desarrollar Módulo de Autenticación',
    'Implementar el sistema de autenticación para el software',
    8
);

CALL agregar_prorroga(1, 10);   -- 10 dias más
```

9.9. Cancelar tareas

Cancelar tareas no elimina sus registros pero impide que una tarea se pueda marcar como completada y que no se pueda asignar

```
CALL crear_tarea(               -- tarea ID = 1
     '2023-11-23',
     'Desarrollar Módulo de Autenticación',
     'Implementar el sistema de autenticación para el software',
     8
);

call marcar_tarea_cancelada(1);
```

9.10. marcar tareas completadas

Cuando una tarea se termina se puede agregar al registro de tareas utilizando el procedimiento tarea completada

```
CALL crear_tarea(               -- tarea ID = 1
     '2023-11-23',
     'Desarrollar Módulo de Autenticación',
     'Implementar el sistema de autenticación para el software',
     8
);

call marcar_tarea_completada(1);
```

9.11. Calcular costos del proyecto

Calcular y proporcionar el costo total del proyecto en el sistema. Esto implica sumar todos los costos de los salarios de los integrantes de los equipos.

```
CALL costo_total_proyecto();
```

9.12. Salario mínimo en el personal

busca y devuelve el salario más bajo dentro del personal del equipo en el sistema. Esto permite identificar el salario mínimo entre los miembros del equipo.

```
CALL salario_mas_bajo_equipo();
```

9.13. Salario máximo en el personal

Se encarga de encontrar y presentar el salario más alto entre los miembros del equipo en el sistema.

```
CALL salario_mas_alto_equipo()
```

9.14. promedio de salarios

Calcula el promedio de salarios del personal en el sistema.

```
CALL calcular_promedio_salario();
```

10. Conclusión

Durante el transcurso de este proyecto, logramos llevar a cabo la implementación exitosa de un sistema de gestión de proyectos utilizando MySQL y MySQL Workbench. A lo largo del semestre, aplicamos de manera práctica todos los conceptos estudiados, ampliando nuestras habilidades en el diseño y administración de bases de datos.

Para el proyecto pusimos en práctica habilidades como búsqueda, selección, eliminación e inserción. También utilizamos funciones, procedimientos y triggers para hacer más fácil la manipulación de los registros. Además, aplicamos conceptos de diseño como llaves foráneas y llaves primarias en las tablas.

11. Fuentes

Wikimedia Foundation. (2023, September 1). *Reloj de Fichar*. Wikipedia. https://es.wikipedia.org/wiki/Reloj_de_fichar

GeeksforGeeks. (2022, November 29). *Different types of procedures in mysql*. GeeksforGeeks. https://www.geeksforgeeks.org/different-types-of-procedures-in-mysql/

GeeksforGeeks. (2022a, January 18). *MySQL: Creating stored function*. GeeksforGeeks. https://www.geeksforgeeks.org/mysql-creating-stored-function/