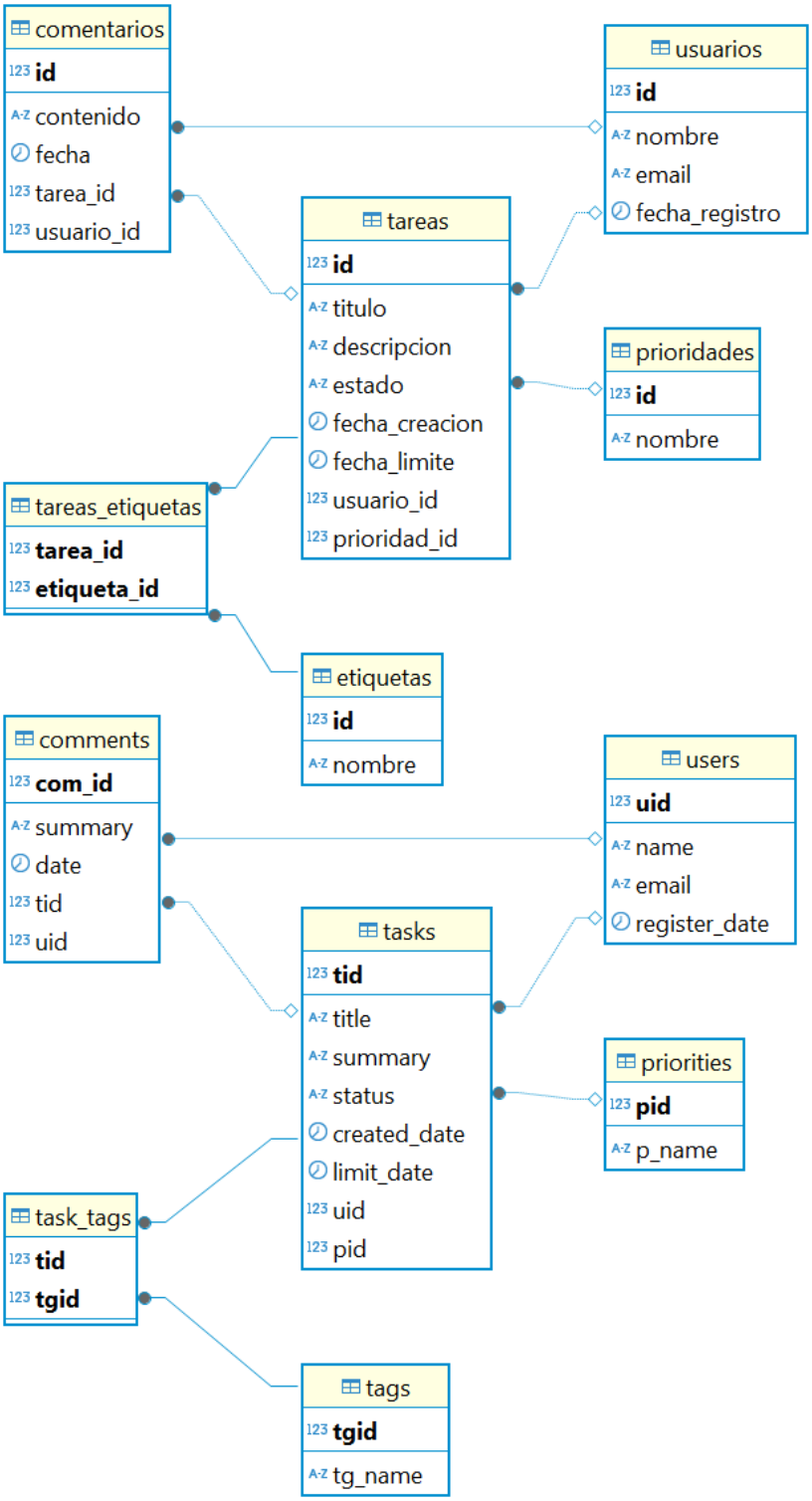
Ejercicios – Tema Segundo Corte

- Bases de Datos que se usará:



1. **Procedimiento:** Crear una vista de todas las tareas asociadas a una categoría (etiqueta)

```
CREATE OR REPLACE PROCEDURE crear_vista_tareas_por_categoria()
LANGUAGE plpgsql
AS $$
BEGIN
  CREATE OR REPLACE VIEW vista_tareas_categoria AS
  SELECT t.id, t.titulo, t.descripcion, e.nombre AS etiqueta
  FROM tareas t
  JOIN tareas_etiquetas te ON t.id = te.tarea_id
  JOIN etiquetas e ON e.id = te.etiqueta_id;
END;
$$;
```

2. **Vista materializada:** Usuarios con tareas pendientes.

```
CREATE MATERIALIZED VIEW usuarios_con_tareas_pendientes AS
SELECT u.id AS usuario_id, u.nombre, t.id AS tarea_id, t.estado
FROM usuarios u
JOIN tareas t ON u.id = t.usuario_id
WHERE t.estado = 'pendiente';
```

3. **Transacción:** Cerrar tareas expiradas por fecha.

```
BEGIN;

UPDATE tareas
SET estado = 'cerrada'
WHERE fecha_limite < CURRENT_DATE
  AND estado != 'cerrada';

COMMIT;
```

4. **Transacción:** Clonar tareas a otro usuario.

```
BEGIN;

INSERT INTO tareas (titulo, descripcion, estado, fecha_creacion, fecha_limite, usuario_id, prioridad_id)
SELECT titulo, descripcion, estado, fecha_creacion, fecha_limite, 2 AS nuevo_usuario, prioridad_id
FROM tareas
WHERE usuario_id = 1;

COMMIT;
```

5. Tabla temporal con información completa de tareas.

```
CREATE TEMP TABLE temp_tareas AS
SELECT t.*, u.nombre AS usuario, p.nombre AS prioridad
FROM tareas t
JOIN usuarios u ON t.usuario_id = u.id
JOIN prioridades p ON t.prioridad_id = p.id;
```

6. Procedimiento para llenar cualquier tabla utilizando ciclos.

```
CREATE OR REPLACE PROCEDURE llenar_tabla(tabla TEXT, valor_base INT, cantidad INT)
LANGUAGE plpgsql
AS $$
DECLARE
  i INT;
BEGIN
  FOR i IN 1..cantidad LOOP
    EXECUTE format('INSERT INTO %I VALUES (%s)', tabla, valor_base + i);
  END LOOP;
END;
$$;

CALL llenar tabla('mi_tabla', 1000, 10);
```

7. **Función:** Nombre del usuario con su tarea.

```
CREATE OR REPLACE FUNCTION obtener_nombre_tarea()
RETURNS TABLE(nombre TEXT, tarea TEXT) AS $$
BEGIN
  RETURN QUERY
  SELECT u.nombre, t.titulo
  FROM tareas t
  JOIN usuarios u ON t.usuario_id = u.id;
END;
$$ LANGUAGE plpgsql;
```

8. **Procedimiento:** Concatenar etiquetas y prioridad de las tareas.

```
CREATE OR REPLACE FUNCTION reporte_tareas_completo()
RETURNS TABLE(no INT, nombre_completo TEXT, tarea TEXT, prioridad TEXT, etiquetas TEXT) AS $$
BEGIN
  RETURN QUERY
  SELECT
    ROW_NUMBER() OVER () AS no,
    u.nombre,
    t.titulo,
    p.nombre AS prioridad,
    STRING_AGG(e.nombre, ', ') AS etiquetas
  FROM tareas t
  JOIN usuarios u ON t.usuario_id = u.id
  JOIN prioridades p ON t.prioridad_id = p.id
  LEFT JOIN tareas_etiquetas te ON t.id = te.tarea_id
  LEFT JOIN etiquetas e ON te.etiqueta_id = e.id
  GROUP BY u.nombre, t.titulo, p.nombre;
END;
$$ LANGUAGE plpgsql;
```

9. Procedimiento para crear vistas normales o materializadas con CASE

```sql
CREATE OR REPLACE FUNCTION reporte_tareas_completo()
RETURNS TABLE(no INT, nombre_completo TEXT, tarea TEXT, prioridad TEXT, etiquetas TEXT) AS $$
BEGIN
  RETURN QUERY
  SELECT
    ROW_NUMBER() OVER () AS no,
    u.nombre,
    t.titulo,
    p.nombre AS prioridad,
    STRING_AGG(e.nombre, ', ') AS etiquetas
  FROM tareas t
  JOIN usuarios u ON t.usuario_id = u.id
  JOIN prioridades p ON t.prioridad_id = p.id
  LEFT JOIN tareas_etiquetas te ON t.id = te.tarea_id
  LEFT JOIN etiquetas e ON te.etiqueta_id = e.id
  GROUP BY u.nombre, t.titulo, p.nombre;
END;
$$ LANGUAGE plpgsql;
```

```sql
CREATE OR REPLACE PROCEDURE crear_vista_dinamica(tipo TEXT, nombre_vista TEXT)
LANGUAGE plpgsql
AS $$
BEGIN
  CASE tipo
    WHEN 'materializada' THEN
      EXECUTE format('CREATE MATERIALIZED VIEW %I AS SELECT * FROM tareas', nombre_vista);
    WHEN 'normal' THEN
      EXECUTE format('CREATE VIEW %I AS SELECT * FROM tareas', nombre_vista);
    ELSE
      RAISE NOTICE 'Tipo de vista no válido.';
  END CASE;
END;
$$;
```

```sql
CALL crear_vista_dinamica('normal', 'vista_tareas_normales');
```