Universidad Nacional De La Matanza
Especialización Ciencia de Datos

Tópicos de Bases de Datos
Profesor: Julio Bossero

Trabajo Práctico Integrador

Parte 3: Operaciones DML y Consultas Avanzadas DDL

Integrantes Grupo VI:
Fica Millán, Yesica - DNI 27624.956
Petraroia, Franco - DNI 27.161.862
Miranda Charca, Florencia - DNI 41.398.768

- Año 2025 -

Contenido

CONSIGNA

1. Objetivo

Aplicar sentencias de manipulación de datos (DML) y consultas SQL avanzadas sobre el modelo relacional implementado en la Parte II, con el fin de validar la integridad de los datos, explorar relaciones entre entidades y demostrar el uso de técnicas complejas de consulta en SQL Server.

2. Consigna

1. Inserción de Datos (INSERT)

Se desarrollarán scripts de inserción para poblar las tablas principales del modelo relacional. Los datos serán representativos y permitirán realizar consultas significativas.

- Se incluirán al menos 5 registros por tabla.
- Se respetarán las claves foráneas y las restricciones definidas en la Parte II.

2. Consultas con JOIN y Subconsultas

Se presentarán consultas que involucren:

- INNER JOIN, LEFT JOIN, y FULL JOIN entre entidades relacionadas.
- Subconsultas (Operadores IN,EXISTS, ANY y ALL).
- Cada consulta estará acompañada de su enunciado descriptivo. Esto es para todos los puntos.
- De ser posible, la imagen correspondiente a los datos devueltos. Esto es para todos los puntos.

3. Consultas con Funciones Comunes y de Agregados (Count(), Sum(), etc.

Se desarrollarán consultas que utilicen funciones comunes de SQL Server para resolver casos específicos del sistema.

- Funciones de texto.
- Funciones matemáticas.
- Funciones de fecha.

4. Funciones Definidas por el Usuario (UDF)

Se solicita la creación de al menos una función definida por el usuario que encapsule una lógica específica del sistema y pueda ser reutilizada en distintas consultas.

- Escalar
- De tabla en línea.
- De tabla multisentencia

5. Operaciones de Actualización y Eliminación (UPDATE, DELETE)

Se incluirán ejemplos de:

- Actualización de datos en función de condiciones específicas.
- Eliminación de registros con control de integridad referencial.

6. Consultas con Funciones de Ventana

Se utilizarán funciones como:

- OVER(PARTITION BY ...) para calcular agregados por grupo.
- COUNT(), SUM(), ROW_NUMBER(), RANK(), DENSE_RANK()

7. Expresiones Comunes de Tabla (CTE)

Se desarrollarán consultas que utilicen WITH para definir CTEs, incluyendo:

- CTEs recursivas para jerarquías o estructuras anidadas.
- CTEs para simplificar subconsultas complejas.

8. Transformaciones con PIVOT y UNPIVOT

Se incluirán ejemplos de:

- Transformación de datos verticales a horizontales (PIVOT).
- Reversión de estructuras (UNPIVOT) para análisis dinámico.

9. Vistas

Se solicita la creación de los **tipos de vistas** vistos, que reflejen distintos aspectos del sistema implementado. Cada vista debe estar acompañada de una breve descripción de su propósito y utilidad.

10. Formato de entrega

- Documento en PDF que incluya:
  - Un Diagrama de la BD realizado con SSMS.
- Scripts de las consultas, con sus enunciados y resultado esperado
  - Incluir todas las sentencias DDL utilizadas para crear la base de datos, esquemas, tablas

- Nombre de archivo: **TP3_GrupoXX.pdf.**
- Por favor, solo entregar un solo archivo .pdf.
- Fecha de entrega: Jueves 04 de septiembre.

DESARROLLO

1. Inserción de Datos

1.1 Insertar datos en produccion.Animal

Este script genera y carga de forma aleatoria y automática 100 registros de animales en la tabla `produccion.Animal`. Se implementa una lógica para garantizar la unicidad de los IDs de los animales generados, evitando conflictos con registros existentes en la tabla.

```
SET NOCOUNT ON;

DECLARE @contador INT = 0;
DECLARE @ids TABLE (id INT);    -- tabla temporal para almacenar IDs generados

WHILE @contador < 100           -- genera 100 animales
BEGIN
    DECLARE @nuevo_id INT = CAST(RAND()*9000 + 1000 AS INT);
    DECLARE @sexo CHAR(1);

    -- asegurar que no exista en la tabla ni en los IDs generados
    WHILE EXISTS (SELECT 1 FROM produccion.Animal WHERE id_animal = @nuevo_id)
          OR EXISTS (SELECT 1 FROM @ids WHERE id = @nuevo_id)
    BEGIN
        SET @nuevo_id = CAST(RAND()*9000 + 1000 AS INT);
    END;

    -- asignar sexo aleatorio
    SET @sexo = CASE CAST(RAND()*2+1 AS INT)
                    WHEN 1 THEN 'M'
                    ELSE 'H'
                END;

    INSERT INTO produccion.Animal
        (id_animal, raza, sexo, fecha_nacimiento, peso_nacimiento, categoria,
estado)
    VALUES
        (@nuevo_id,
         'Aberdeen Angus',                      -- raza fija
         @sexo,
         DATEADD(DAY, -CAST(RAND()*2000 AS INT), GETDATE()),  -- fecha nacimiento
aleatoria
         CAST(RAND()*500+200 AS DECIMAL(5,2)),   --  peso entre 200 y 700 aprox.
         CASE WHEN @sexo = 'M'
              THEN 'Toro'
              ELSE CASE CAST(RAND()*2+1 AS INT)
                        WHEN 1 THEN 'Vaca'
                        ELSE 'Vaquillona'
                   END
         END,
```

```
        'Activo');

    INSERT INTO @ids VALUES (@nuevo_id);     -- guardar ID generado
    SET @contador = @contador + 1;
END;
GO

-- Verificar los primeros 10 registros de Animal
SELECT TOP 10 *
FROM produccion.Animal
ORDER BY id_animal;
```

Resultados Mensajes

	id_animal	raza	sexo	fecha_nacimiento	peso_nacimiento	id_madre	categoria	estado
1	1068	Aberdeen Angus	H	2020-06-17	485.52	NULL	Vaquillona	Activo
2	1100	Aberdeen Angus	H	2020-11-14	684.21	NULL	Vaca	Activo
3	1315	Aberdeen Angus	M	2024-01-31	419.82	NULL	Toro	Activo
4	1370	Aberdeen Angus	H	2020-08-11	657.36	NULL	Vaquillona	Activo
5	1452	Aberdeen Angus	H	2023-04-25	572.62	NULL	Vaca	Activo
6	1503	Aberdeen Angus	H	2024-02-26	421.59	NULL	Vaquillona	Activo
7	1515	Aberdeen Angus	M	2022-06-30	389.78	NULL	Toro	Activo
8	1523	Aberdeen Angus	H	2025-03-13	402.00	NULL	Vaquillona	Activo
9	1530	Aberdeen Angus	M	2021-11-15	536.21	NULL	Toro	Activo
10	1534	Aberdeen Angus	M	2023-11-11	524.83	NULL	Toro	Activo

1.2 Insertar datos en produccion.Nacimiento

El siguiente script genera automáticamente entre 10 y 30 registros de nacimientos para 10 madres seleccionadas al azar de la tabla `produccion.Animal`. Cada madre tiene entre 1 y 3 partos, el primero en una fecha aleatoria entre el 15 de julio y el 31 de octubre de los últimos 5 años. Los partos siguientes se registran en orden cronológico, y el tipo de parto se asigna aleatoriamente entre: 'Normal', 'Cesárea', 'Asistido', 'Aborto' o 'No presenta ternero'.

```
SET NOCOUNT ON;

DECLARE @contador INT = 0;

-- Generar nacimientos para 10 madres distintas
WHILE @contador < 10
BEGIN
    DECLARE @madre_id INT;
    DECLARE @partos INT;
    DECLARE @anio INT;
    DECLARE @nro_nacimiento INT;
    DECLARE @fecha DATE;

    -- Seleccionar una madre (hembra) aleatoria existente
    SELECT TOP 1 @madre_id = id_animal
    FROM produccion.Animal
    WHERE sexo = 'H'
      AND id_animal NOT IN (SELECT madre_id FROM produccion.Nacimiento)
    ORDER BY NEWID();
```

```
    -- Número de partos entre 1 y 3 para esa madre
    SET @partos = CAST(RAND()*3 + 1 AS INT);

    -- Año base aleatorio entre los últimos 5 años
    SET @anio = YEAR(GETDATE()) - CAST(RAND()*5 AS INT);

    WHILE @partos > 0
    BEGIN
        -- Calcular nro_nacimiento correcto según la cantidad de partos previos
        SET @nro_nacimiento = ISNULL(
            (SELECT MAX(nro_nacimiento)
             FROM produccion.Nacimiento
             WHERE madre_id = @madre_id), 0
        ) + 1;

        -- Generar fecha entre 15/07 y 31/10 del año correspondiente
        SET @fecha = DATEADD(DAY, CAST(RAND()*108 AS INT), CAST(CAST(@anio AS
VARCHAR) + '-07-15' AS DATE));

        -- Asegurar que los partos posteriores no pasen del año actual
        IF @nro_nacimiento > 1
        BEGIN
            SET @fecha = DATEADD(DAY, 365 * (@nro_nacimiento - 1), @fecha);
            IF @fecha > GETDATE()
                SET @fecha = DATEADD(DAY, -CAST(RAND()*90 AS INT), GETDATE());
        END;

        -- Insertar nacimiento
        INSERT INTO produccion.Nacimiento (nro_nacimiento, madre_id, fecha,
tipo_parto)
        VALUES (
            @nro_nacimiento,
            @madre_id,
            @fecha,
            CASE CAST(RAND()*5+1 AS INT)
                WHEN 1 THEN 'Normal'
                WHEN 2 THEN 'Cesarea'
                WHEN 3 THEN 'Asistido'
                WHEN 4 THEN 'Aborto'
                ELSE 'No presenta ternero'
            END
        );

        SET @partos = @partos - 1;
    END;

    SET @contador = @contador + 1;
END;
GO

-- Verificar los Nacimiento creados
SELECT TOP 10 *
FROM produccion.Nacimiento
ORDER BY madre_id, nro_nacimiento;
```

Resultados Mensajes

	nro_nacimiento	madre_id	fecha	tipo_parto
1	1	2914	2024-09-07	No presenta ternero
2	2	2914	2025-08-07	Normal
3	1	3306	2022-08-08	No presenta ternero
4	2	3306	2023-08-23	Asistido
5	3	3306	2024-09-14	Cesarea
6	1	3511	2021-09-18	Asistido
7	1	4044	2021-08-05	Cesárea
8	2	4044	2022-07-29	Normal
9	3	4044	2023-09-21	No presenta ternero
10	1	4112	2025-09-22	Asistido

```sql
-- Verificar que cada parto posterior sea mayor al anterior
SELECT madre_id, nro_nacimiento, fecha,
       LAG(fecha) OVER (PARTITION BY madre_id ORDER BY nro_nacimiento) AS
parto_anterior
FROM produccion.Nacimiento;
```

Resultados Mensajes

	madre_id	nro_nacimiento	fecha	parto_anterior
1	2914	1	2024-09-07	NULL
2	2914	2	2025-08-07	2024-09-07
3	3306	1	2022-08-08	NULL
4	3306	2	2023-08-23	2022-08-08
5	3306	3	2024-09-14	2023-08-23
6	3511	1	2021-09-18	NULL
7	4044	1	2021-08-05	NULL
8	4044	2	2022-07-29	2021-08-05
9	4044	3	2023-09-21	2022-07-29
10	4112	1	2025-09-22	NULL
11	4112	2	2025-07-05	2025-09-22
12	4112	3	2025-08-13	2025-07-05
13	4369	1	2022-09-16	NULL
14	4369	2	2023-07-22	2022-09-16
15	4538	1	2024-07-22	NULL
16	4580	1	2025-09-27	NULL
17	4580	2	2025-07-29	2025-09-27
18	5907	1	2024-09-03	NULL
19	6439	1	2023-07-29	NULL
20	6439	2	2024-09-12	2023-07-29
21	6439	3	2025-08-01	2024-09-12
22	7221	1	2022-09-18	NULL
23	7279	1	2024-08-03	NULL
24	7279	2	2025-08-18	2024-08-03
25	7279	3	2025-06-18	2025-08-18

1.3 Insertar datos en produccion.Peso

Para esta tabla, el script genera de forma automática y aleatoria entre 60 y 100 registros de peso en la tabla `produccion.Peso`. Este proceso simula múltiples mediciones para 20 animales seleccionados al azar de la tabla `produccion.Animal`.

```
SET NOCOUNT ON;

DECLARE @contador INT = 0;

-- Se generan pesos para 20 animales, simulando entre 3 y 5 mediciones por animal
WHILE @contador < 20
BEGIN
    DECLARE @id_animal INT;
    -- Se toma un animal existente de la tabla Animal
    SELECT TOP 1 @id_animal = id_animal
    FROM produccion.Animal
    ORDER BY NEWID();

    DECLARE @mediciones INT = CAST(RAND()*3 + 3 AS INT); -- entre 3 y 5
mediciones
    DECLARE @i INT = 0;

    WHILE @i < @mediciones
    BEGIN
        INSERT INTO produccion.Peso (fecha_medicion, id_animal, valor)
        VALUES (
            DATEADD(DAY, -CAST(RAND()*1000 AS INT), GETDATE()), -- fecha
aleatoria dentro de ~3 años
            @id_animal,
            CAST(RAND()*400 + 200 AS DECIMAL(5,2)) -- peso entre 200 y 600 kg
        );

        SET @i = @i + 1;
    END

    SET @contador = @contador + 1;
END;
GO

-- Verificar los primeros 10 registros de Peso
SELECT TOP 10 *
FROM produccion.Peso;
```

Resultados Mensajes

	fecha_medicion	id_animal	valor
1	2023-02-04	1068	404.02
2	2023-05-16	1068	567.24
3	2024-04-21	1068	339.28
4	2025-03-04	1068	315.69
5	2024-07-23	1315	246.54
6	2024-11-22	1315	557.26
7	2024-12-19	1315	527.65
8	2025-01-23	1315	340.82
9	2025-05-05	1315	313.55
10	2023-02-03	1503	222.11

1.4 Insertar datos en produccion.Potrero

Este script automatiza la población de la tabla `produccion.Potrero` con 15 registros generados de forma aleatoria, facilitando la creación de un conjunto de datos de prueba.

```
SET NOCOUNT ON;

DECLARE @contador INT = 0;

-- Generar 15 potreros de ejemplo
WHILE @contador < 15
BEGIN
    INSERT INTO produccion.Potrero (superficie, tipo)
    VALUES (
        CAST(RAND()*90 + 15 AS DECIMAL(6,2)), -- superficie entre 15 y 90
hectáreas
        CASE CAST(RAND()*3+1 AS INT)
            WHEN 1 THEN 'Pastura'
            WHEN 2 THEN 'Engorde'
            ELSE 'Reserva'
        END
    );

    SET @contador = @contador + 1;
END;
GO

-- Verificar los registros de Potrero
SELECT *
FROM produccion.Potrero;
```

Resultados Mensajes

	id_potrero	superficie	tipo
1	1	90.29	Reserva
2	2	21.03	Pastura
3	3	26.42	Engorde
4	4	103.71	Pastura
5	5	81.64	Pastura
6	6	52.41	Engorde
7	7	65.67	Pastura
8	8	16.22	Engorde
9	9	25.27	Pastura
10	10	83.16	Reserva
11	11	33.37	Reserva
12	12	47.81	Engorde
13	13	17.18	Reserva
14	14	104.12	Engorde
15	15	56.39	Pastura

1.5 Insertar datos en produccion.Animal_Potrero

Este script genera y carga 20 registros de asignación de animales a potreros en la tabla `produccion.Animal_Potrero`. Su propósito es simular de forma automática y aleatoria las ubicaciones históricas de los animales en el establecimiento.

```
SET NOCOUNT ON;

DECLARE @contador INT = 0;

-- Generar 20 asignaciones aleatorias de animales a potreros
WHILE @contador < 20
BEGIN
    DECLARE @id_animal INT;
    DECLARE @id_potrero INT;

    -- Seleccionar un animal aleatorio existente
    SELECT TOP 1 @id_animal = id_animal
    FROM produccion.Animal
    ORDER BY NEWID();

    -- Seleccionar un potrero aleatorio existente
    SELECT TOP 1 @id_potrero = id_potrero
    FROM produccion.Potrero
    ORDER BY NEWID();

    -- Insertar la asignación con fecha aleatoria dentro de los últimos 3 años
    INSERT INTO produccion.Animal_Potrero (fecha_asignacion, id_animal,
id_potrero)
    VALUES (
        DATEADD(DAY, -CAST(RAND()*1000 AS INT), GETDATE()), -- fecha aleatoria <=
hoy
        @id_animal,
        @id_potrero
    );
```

```
    SET @contador = @contador + 1;
END;
GO

-- Verificar los registros de Animal_Potrero
SELECT *
FROM produccion.Animal_Potrero;
```

Resultados Mensajes

	fecha_asignacion	id_animal	id_potrero
1	2023-07-21	7289	1
2	2023-08-11	9882	1
3	2023-06-06	8346	5
4	2023-09-09	9447	5
5	2025-06-10	3653	5
6	2024-09-21	5948	6
7	2024-11-03	5637	6
8	2024-12-17	4516	7
9	2025-02-23	7279	7
10	2025-07-24	9512	8
11	2025-07-02	2192	9
12	2024-04-28	6202	10
13	2022-12-19	7268	11
14	2024-06-16	1534	11
15	2024-07-05	8546	11
16	2024-11-03	9882	12
17	2024-09-16	9179	13
18	2023-03-30	4044	15
19	2023-06-11	7279	15
20	2025-07-07	5628	15

1.6 Insertar datos en produccion.Evento

En este script se generan y cargan 5 registros de eventos de manera automática y aleatoria en la tabla `produccion.Evento`. Su propósito es simular de forma sencilla y eficiente una serie de eventos para poblar la base de datos con datos de prueba.

```
SET NOCOUNT ON;

DECLARE @contador INT = 0;

-- Generar 5 eventos aleatorios
WHILE @contador < 5
BEGIN
    INSERT INTO produccion.Evento (tipo_evento, fecha, observaciones)
    VALUES (
        -- Tipo de evento aleatorio
        CASE CAST(RAND()*4+1 AS INT)
            WHEN 1 THEN 'Vacunacion'
            WHEN 2 THEN 'Remate'
```

```
            WHEN 3 THEN 'Inseminacion'
            ELSE 'General'
        END,
        -- Fecha aleatoria dentro de los últimos 2 años
        DATEADD(DAY, -CAST(RAND()*730 AS INT), GETDATE()),
        -- Observación breve generada
        CONCAT('Evento generado automáticamente nro ', @contador+1)
    );

    SET @contador = @contador + 1;
END;
GO

-- Verificar los registros de Evento
SELECT *
FROM produccion.Evento
ORDER BY fecha;
```

Resultados Mensajes

	id_evento	tipo_evento	fecha	observaciones
1	4	Remate	2023-11-25	Evento generado automáticamente nro 4
2	2	Inseminacion	2024-09-09	Evento generado automáticamente nro 2
3	3	Remate	2024-10-14	Evento generado automáticamente nro 3
4	5	General	2024-11-06	Evento generado automáticamente nro 5
5	1	Vacunacion	2025-02-06	Evento generado automáticamente nro 1
6	6	Siembra	2025-02-23	Verdeo de invierno

```
-- Contar eventos por tipo
SELECT tipo_evento, COUNT(*) AS cantidad
FROM produccion.Evento
GROUP BY tipo_evento;
```

Resultados Mensajes

	tipo_evento	cantidad
1	General	1
2	Inseminacion	1
3	Remate	2
4	Siembra	1
5	Vacunacion	1

1.7 Insertar datos en rrhh.Personal

En este caso el script automatiza la carga de 7 registros de personal en la tabla `rrhh.Personal` con datos ficticios generados de forma aleatoria. Su objetivo es crear rápidamente un conjunto de datos de prueba para el módulo de recursos humanos.

```
SET NOCOUNT ON;

DECLARE @contador INT = 0;

-- Generar 7 registros de personal
WHILE @contador < 7
```

```
BEGIN
    INSERT INTO rrhh.Personal (nombre, apellido, contacto_emerg, celular)
    VALUES (
        -- Nombre aleatorio
        CASE CAST(RAND()*6+1 AS INT)
            WHEN 1 THEN 'Juan'
            WHEN 2 THEN 'Pedro'
            WHEN 3 THEN 'Mario'
            WHEN 4 THEN 'Jose'
            WHEN 5 THEN 'Carlos'
            WHEN 6 THEN 'Carlos'
            ELSE 'Julian'
        END,
        -- Apellido aleatorio
        CASE CAST(RAND()*6+1 AS INT)
            WHEN 1 THEN 'Gómez'
            WHEN 2 THEN 'Pérez'
            WHEN 3 THEN 'Rodríguez'
            WHEN 4 THEN 'Fernández'
            WHEN 5 THEN 'Martínez'
            WHEN 6 THEN 'Gonzalez'
            ELSE 'López'
        END,
        -- Contacto de emergencia (número de celular)
        CAST(9000000000 + CAST(RAND() * 999999999 AS BIGINT) AS VARCHAR(15)),
        -- Celular ficticio (número válido)
        CAST(9000000000 + CAST(RAND() * 999999999 AS BIGINT) AS VARCHAR(15))
    );

    SET @contador = @contador + 1;
END;
GO

-- Ver todos los registros cargados
SELECT *
FROM rrhh.Personal;
```

Resultados Mensajes

	id_personal	nombre	apellido	contacto_emerg	celular
1	1	Carlos	Fernández	9783788755	9355360770
2	2	Jose	Gómez	9542005710	9916789912
3	3	Julian	Gómez	9340289214	9807237118
4	4	Julian	Fernández	9572943786	9849072418
5	5	Juan	Fernández	9081693994	9839127798
6	6	Julian	Gómez	9484169022	9729745787
7	7	Mario	López	9446819496	9992284056

1.8 Insertar datos en rrhh.Peon

Este código inserta tres registros específicos de peones en la tabla `rrhh.Peon`. A diferencia de los scripts anteriores que generaban datos aleatorios, este es un ejemplo de inserción manual de datos.

En primer lugar se realizó una consulta de la tabla rrhh.Personal para verificar el id_personal y asi poder hacer la asignación.

```
-- Verificar los IDs existentes en rrhh.Personal
SELECT id_personal, nombre, apellido
FROM rrhh.Personal;

-- Insertar manualmente en la tabla 3 peones
INSERT INTO rrhh.Peon (id_personal, sector_asignado, turno)
VALUES
    (1, 'Mantenimiento', 'Mañana'), -- Peón 1, sector de mantenimiento, turno
mañana
    (2, 'Ganadería', 'Tarde'),      -- Peón 2, sector de ganadería, turno tarde
    (3, 'Agricultura', 'Noche');    -- Peón 3, sector de agricultura, turno noche
GO

-- Ver todos los registros de Peon
SELECT *
FROM rrhh.Peon;
```

Resultados Mensajes

	id_personal	sector_asignado	turno
1	1	Mantenimiento	Mañana
2	2	Ganadería	Tarde
3	3	Agricultura	Noche

1.9 Insertar datos en rrhh.Capataz

Este código inserta un registro específico en la tabla `rrhh.Capataz`, Es otro ejemplo de carga de datos manual, donde cada valor es definido de forma explícita.

```
INSERT INTO rrhh.Capataz (id_personal, anios_experiencia)
VALUES
    (4, 10);   -- Capataz con 10 años de experiencia
GO

-- Ver todos los registros de Capataz
SELECT *
FROM rrhh.Capataz;
```

Resultados Mensajes

	id_personal	anios_experiencia
1	4	10

1.10 Insertar datos en rrhh.Tractorista

Este código inserta un registro individual en la tabla `rrhh.Tractorista`. Representa una carga de datos específica y manual, a diferencia de la generación aleatoria que se vería en scripts para datos de prueba.

```
INSERT INTO rrhh.Tractorista (id_personal, licencia_nro, tipo_maquinaria)
VALUES
    (5, '35123789', 'Tractor');  -- Tractorista con licencia y tipo de maquinaria
GO
```

```sql
-- Ver todos los tractoristas cargados
SELECT *
FROM rrhh.Tractorista;
```

Resultados Mensajes

	id_personal	licencia_nro	tipo_maquinaria
1	5	35123789	Tractor

1.11 Insertar datos en rrhh.Encargado

Este es otro ejemplo de carga de datos manual, donde cada valor es definido de forma explícita. El código inserta un registro específico en la tabla `rrhh.Encargado`.

```sql
INSERT INTO rrhh.Encargado (id_personal, ppto_asignado, cant_personal)
VALUES
    (6, 250000.00, 5);   -- Encargado con presupuesto y 5 personas a cargo
GO

-- Ver todos los encargados cargados
SELECT *
FROM rrhh.Encargado;
```

Resultados Mensajes

	id_personal	ppto_asignado	cant_personal
1	6	250000.00	5

1.12 Insertar datos en rrhh.Veterinario

Este código también inserta un registro específico en la tabla `rrhh.Veterinario`. Es otro ejemplo de carga de datos manual, donde cada valor es definido de forma explícita.

```sql
-- Insertar manualmente un veterinario
INSERT INTO rrhh.Veterinario (id_personal, mat_profesional, especialidad)
VALUES
    (7, 'MAT-AR-2025-0014', 'Reproducción Bovina');
GO

-- Ver todos los veterinarios cargados
SELECT *
FROM rrhh.Veterinario;
```

Resultados Mensajes

	id_personal	mat_profesional	especialidad
1	7	MAT-AR-2025-0014	Reproducción Bovina

1.13 Insertar datos en gestion.Animal_Personal_Evento

Este código inserta tres registros específicos en la tabla `gestion.Animal_Personal_Evento`, una tabla de relación que vincula animales, personal y eventos. El script es un ejemplo de carga de datos manual, donde cada registro define de forma explícita la interacción entre estas tres entidades.

```
INSERT INTO gestion.Animal_Personal_Evento (id_animal, id_personal, id_evento,
rol_evento)
VALUES
    (1068, 7, 1, 'Veterinario a cargo'),   -- veterinario en revisión sanitaria
    (1370, 2, 1, 'Peón auxiliar'),         -- peón que ayuda
    (1503, 6, 1, 'Encargado supervisor');  -- encargado que supervisa un evento
GO

-- Ver registro cargado
SELECT *
FROM gestion.Animal_Personal_Evento
WHERE id_evento = 1;
```

Resultados | Mensajes

	id_animal	id_personal	id_evento	rol_evento
1	1370	2	1	Peón auxiliar
2	1503	6	1	Encargado supervisor
3	1068	7	1	Veterinario a cargo

1.14 Insertar datos en gestion.Potrero_Evento

En este caso: primero se inserta un registro específico de un evento en la tabla `produccion.Evento`; luego, verifica el ID del evento recién creado para su uso posterior; y finalmente, se asocia ese evento a múltiples potreros en la tabla de relación `gestion.Potrero_Evento`.

```
-- Se inserta manualmente un evento específico
INSERT INTO produccion.Evento (tipo_evento, fecha, observaciones)
VALUES
    ('Siembra', '2025-02-23', 'Verdeo de invierno');
GO

-- Se verifica el número del ID del evento creado para este fin
SELECT TOP 1 *
FROM produccion.Evento
ORDER BY id_evento DESC;
```

Resultados | Mensajes

	id_evento	tipo_evento	fecha	observaciones
1	6	Siembra	2025-02-23	Verdeo de invierno

```
-- Ejemplo de evento en distintos potreros
INSERT INTO gestion.Potrero_Evento (id_potrero, id_evento)
VALUES
    (1, 6),   -- Evento 6 en Potrero 1
```

```
    (2, 6),   -- Evento 6 en Potrero 2
    (3, 6);   -- Evento 6 en Potrero 3
GO

-- Ver registro cargado
SELECT *
FROM gestion.Potrero_Evento
WHERE id_evento = 6;
```

Resultados | Mensajes

	id_potrero	id_evento
1	1	6
2	2	6
3	3	6

2. Consultas con JOIN y Subconsultas

2.1 INNER JOIN

El código realiza las siguientes uniones lógicas para construir el informe:

- Se une la tabla de eventos (`Evento`) con la tabla de relación (`Animal_Personal_Evento`) a través del `id_evento`.
- Se une la tabla de personal (`Personal`) con la tabla de relación a través del `id_personal`.
- Se une la tabla de animales (`Animal`) con la tabla de relación a través del `id_animal`.

Al usar `INNER JOIN`, la consulta garantiza que solo se incluyan en el resultado los registros que tienen una correspondencia en todas las tablas, es decir, solo se mostrarán las participaciones confirmadas de animales, personal y eventos. El resultado final se presenta ordenado por el ID del animal y el ID del evento, lo que facilita la lectura y el análisis de las participaciones.

```
-- Enunciado: Mostrar los animales que participaron en algún evento junto con el
nombre del personal y el tipo de evento.
-- INNER JOIN entre Animal_Personal_Evento, Animal, Personal y Evento
-- Muestra solo combinaciones existentes (participaciones confirmadas)

SELECT
    ape.id_animal,
    a.id_animal AS animal_id,
    per.nombre + ' ' + per.apellido AS personal,
    e.tipo_evento,
    ape.rol_evento
FROM gestion.Animal_Personal_Evento ape
INNER JOIN produccion.Animal a ON ape.id_animal = a.id_animal
INNER JOIN rrhh.Personal per ON ape.id_personal = per.id_personal
INNER JOIN produccion.Evento e ON ape.id_evento = e.id_evento
ORDER BY ape.id_animal, ape.id_evento;
```

Resultados Mensajes

	id_animal	animal_id	personal	tipo_evento	rol_evento
1	1068	1068	Mario López	Vacunacion	Veterinario a cargo
2	1370	1370	Jose Gómez	Vacunacion	Peón auxiliar
3	1503	1503	Julian Gómez	Vacunacion	Encargado supervisor

2.2 LEFT JOIN

El código utiliza la instrucción `SELECT` para elegir qué información mostrar y la instrucción `LEFT JOIN` para combinar los datos de dos tablas.

- Punto de Partida (`FROM produccion.Evento e`): La consulta comienza en la tabla de `Evento`. Esta es la tabla principal, lo que significa que todas sus filas serán incluidas en el resultado final.
- La Unión (`LEFT JOIN`): El comando `LEFT JOIN` une la tabla de `Evento` con la tabla `gestion.Potrero_Evento`. La tabla `Potrero_Evento` es la que almacena las relaciones entre los eventos y los potreros. La unión se realiza usando el `id_evento` como un conector entre ambas tablas. Para cada evento en la tabla principal, la consulta busca si hay una fila que coincida en la tabla de `Potrero_Evento`.
- El Resultado:
    - Si hay una coincidencia, la consulta muestra los detalles del evento junto con el ID del potrero.
    - Si no hay una coincidencia (es decir, el evento no se ha asignado a ningún potrero), el campo `id_potrero` del resultado muestra `NULL`, lo que indica que no hay datos relacionados.

```sql
-- Enunciado: Listar todos los eventos y, si corresponde, los potreros donde se
realizaron. Mostrar NULL si no hay potrero asignado.
-- LEFT JOIN entre Evento y Potrero_Evento
-- Incluye todos los eventos, aunque no tengan potrero asignado

SELECT
    e.id_evento,
    e.tipo_evento,
    e.fecha,
    pe.id_potrero
FROM produccion.Evento e
LEFT JOIN gestion.Potrero_Evento pe ON e.id_evento = pe.id_evento
ORDER BY e.fecha;
```

Resultados Mensajes

	id_evento	tipo_evento	fecha	id_potrero
1	4	Remate	2023-11-25	NULL
2	2	Inseminacion	2024-09-09	NULL
3	3	Remate	2024-10-14	NULL
4	5	General	2024-11-06	NULL
5	1	Vacunacion	2025-02-06	NULL
6	6	Siembra	2025-02-23	1
7	6	Siembra	2025-02-23	2
8	6	Siembra	2025-02-23	3

2.3 FULL OUTER JOIN

La operación `FULL OUTER JOIN` se utiliza para obtener una vista completa y detallada de la interconexión entre dos entidades, en este caso: Animales y Eventos. El propósito es recuperar todos los registros de la tabla `produccion.Animal` y todos los registros de la tabla `gestion.Animal_Personal_Evento`, mostrando la relación entre estos.

El código utiliza la instrucción `SELECT` para elegir qué información mostrar y el comando `FULL OUTER JOIN` para unir los datos de las dos tablas.

- Punto de Partida: La consulta inicia en la tabla `produccion.Animal`.
- La Unión (`FULL OUTER JOIN`): El comando `FULL OUTER JOIN` une la tabla `Animal` con la tabla `gestion.Animal_Personal_Evento`, usando el `id_animal` como conector. Esta operación incluye:
    - Todos los animales que tienen una participación en un evento.
    - Todos los animales que no tienen ninguna participación en un evento.
    - Todos los eventos que no tienen ningún animal asignado.

Para los registros que no tienen una coincidencia en la tabla de relación (`gestion.Animal_Personal_Evento`), los campos correspondientes (`id_evento`, `id_personal`, `rol_evento`) se muestran con el valor `NULL`, lo que indica que no hay datos relacionados.

```sql
-- Enunciado: Mostrar todos los animales y todos los eventos, indicando si cada
animal participó o no de cada evento.
-- FULL OUTER JOIN entre Animal y Animal_Personal_Evento
-- Permite ver animales sin eventos y eventos sin animales asignados
SELECT
    a.id_animal,
    ape.id_evento,
    ape.id_personal,
    ape.rol_evento
FROM produccion.Animal a
FULL OUTER JOIN gestion.Animal_Personal_Evento ape ON a.id_animal = ape.id_animal
ORDER BY a.id_animal, ape.id_evento;
```

	id_animal	id_evento	id_personal	rol_evento
1	1068	1	7	Veterinario a cargo
2	1100	NULL	NULL	NULL
3	1315	NULL	NULL	NULL
4	1370	1	2	Peón auxiliar
5	1452	NULL	NULL	NULL
6	1503	1	6	Encargado super...
7	1515	NULL	NULL	NULL
8	1523	NULL	NULL	NULL
9	1530	NULL	NULL	NULL
10	1534	NULL	NULL	NULL
11	1879	NULL	NULL	NULL
12	2014	NULL	NULL	NULL
13	2016	NULL	NULL	NULL
14	2192	NULL	NULL	NULL
15	2739	NULL	NULL	NULL
16	2807	NULL	NULL	NULL
17	2816	NULL	NULL	NULL
18	2854	NULL	NULL	NULL
19	2914	NULL	NULL	NULL
20	3306	NULL	NULL	NULL

En la tabla se puede observar que los únicos datos que están cargados corresponden al evento 1; para los demás eventos, como no tienen potreros asignados, el campo `id_potrero`, `id_personal` y `rol_evento` aparecen con valor NULL.

2.4 Subconsulta con IN

En esta consulta se utiliza una subconsulta con el operador `IN` para responder de manera clara a una pregunta compleja: identificar al personal que participó en eventos de tipo *"Vacunacion"*.

La instrucción se compone de dos partes principales. La subconsulta interna se ejecuta en primer lugar y tiene como finalidad obtener los identificadores de las personas (`id_personal`) que intervinieron en eventos de inseminación. Para ello, combina la tabla de relación `gestion.Animal_Personal_Evento` con la tabla `produccion.Evento`, aplicando un filtro sobre el atributo `tipo_evento`. El resultado es un conjunto de IDs de personal vinculados a ese tipo de procedimiento.

La consulta externa, en segundo lugar, utiliza ese conjunto como referencia en la cláusula `WHERE id_personal IN (...)`. De esta manera, se seleccionan únicamente las filas de la tabla `rrhh.Personal` cuyo identificador coincida con los devueltos por la subconsulta, mostrando finalmente los atributos `nombre` y `apellido`.

En conclusión, la consulta devuelve de manera eficiente un listado del personal que participó en eventos de inseminación, aprovechando la potencia de las subconsultas para filtrar resultados específicos.

```sql
-- Enunciado: Mostrar el personal que participó en eventos de tipo 'Vacunacion'.
SELECT nombre, apellido
FROM rrhh.Personal
WHERE id_personal IN (
    SELECT id_personal
    FROM gestion.Animal_Personal_Evento ape
    INNER JOIN produccion.Evento e ON ape.id_evento = e.id_evento
    WHERE e.tipo_evento = 'Vacunacion'

);
```

Resultados Mensajes

	nombre	apellido
1	Jose	Gómez
2	Julian	Gómez
3	Mario	López

2.5 Subconsulta con EXISTS

En esta consulta se utiliza una subconsulta correlacionada junto con el operador `EXISTS`. La tabla principal es `produccion.Animal`, que contiene el listado de animales.

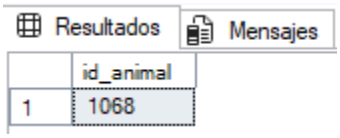
Por cada animal, el sistema verifica si existe un registro en la tabla intermedia `gestion.Animal_Personal_Evento`, donde se almacenan las participaciones de los animales, el personal y los eventos. A su vez, se hace un `INNER JOIN` con la tabla `rrhh.Veterinario`, lo que permite filtrar únicamente los casos en que el personal involucrado sea de tipo veterinario.

La condición `ape.id_animal = a.id_animal` establece la relación entre el animal de la consulta externa y el de la subconsulta.

El uso de `EXISTS` significa que la consulta no necesita traer datos detallados de los eventos, sino simplemente comprobar si existe al menos una coincidencia. De este modo, el resultado final es un listado de animales que han tenido al menos un veterinario a cargo en algún evento, cumpliendo con el criterio de existencia sin necesidad de duplicar registros.

```sql
-- Enunciado: Mostrar los animales que tuvieron al menos un veterinario a cargo
en algún evento.
SELECT a.id_animal
FROM produccion.Animal a
WHERE EXISTS (
    SELECT 1
    FROM gestion.Animal_Personal_Evento ape
    INNER JOIN rrhh.Veterinario v ON ape.id_personal = v.id_personal
    WHERE ape.id_animal = a.id_animal
```

```
);
```



2.6 Subconsulta con ANY

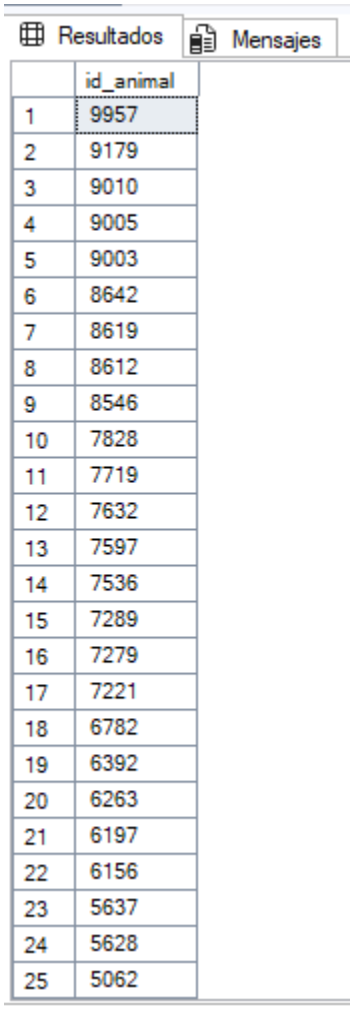
La consulta obtiene los animales cuyo último peso registrado es mayor que al menos uno de los pesos de los animales de categoría '`Vaquillona`'.

- Primero, se calcula la última fecha de medición de peso de cada animal.
- Luego, se une con la tabla `Peso` para obtener el valor del peso más reciente.
- Finalmente, se filtran los resultados usando `WHERE p.valor > ANY (...)`, lo que garantiza que se incluyan aquellos animales cuyo último peso supere al menos uno de los pesos históricos de vaquillonas.

```
-- Enunciado: Listar los animales cuyo último peso supera al menos uno de los
pesos registrados en vaquillonas.
-- Selecciona los animales cuyo último peso es mayor que alguno de los pesos de
vaquillonas
SELECT a.id_animal
FROM produccion.Animal a
-- Subconsulta: obtener la última fecha de medición de peso de cada animal
JOIN (
    SELECT id_animal, MAX(fecha_medicion) AS ultima_fecha
    FROM produccion.Peso
    GROUP BY id_animal
) ult ON a.id_animal = ult.id_animal

-- Obtener el peso correspondiente a la última fecha
JOIN produccion.Peso p
    ON p.id_animal = a.id_animal AND p.fecha_medicion = ult.ultima_fecha

-- Comparar con todos los pesos históricos de vaquillonas, usando ANY
WHERE p.valor > ANY (
    SELECT valor
    FROM produccion.Peso p2
    JOIN produccion.Animal a2 ON p2.id_animal = a2.id_animal
    WHERE a2.categoria = 'Vaquillona'
);
```



Resultados | Mensajes

	id_animal
1	9957
2	9179
3	9010
4	9005
5	9003
6	8642
7	8619
8	8612
9	8546
10	7828
11	7719
12	7632
13	7597
14	7536
15	7289
16	7279
17	7221
18	6782
19	6392
20	6263
21	6197
22	6156
23	5637
24	5628
25	5062

2.7 Subconsulta con ALL

Esta consulta tiene como objetivo identificar al personal que participó en todos los eventos de tipo "`Vacunación`" registrados en la base de datos. Para lograrlo, se combina la información de tres tablas: `rrhh.Personal` (que contiene los datos del personal), `gestion.Animal_Personal_Evento` (que relaciona personal, animales y eventos), y `produccion.Evento` (que almacena los eventos realizados).

La consulta se estructura en dos niveles:

1º. Consulta externa:
   - Selecciona los campos `id_personal`, nombre y apellido de la tabla `rrhh.Personal`.
   - Filtra los resultados usando la cláusula `WHERE` junto con una subconsulta que define los criterios de selección.

2º. Subconsulta con `IN` y `ALL`:

- La subconsulta primero une (`INNER JOIN`) la tabla de eventos con la tabla de relaciones de personal y eventos, de manera que cada registro de participación pueda ser asociado con su tipo de evento.
- Se filtra para considerar únicamente los eventos de tipo `'Vacunacion'`.
- Luego, se agrupa por `id_personal` para contar cuántos eventos de vacunación ha participado cada persona.
- La cláusula `HAVING` compara este conteo con el total de eventos de vacunación existentes utilizando `ALL`. Esto asegura que solo se seleccionen aquellos individuos que hayan participado en todos los eventos de vacunación registrados.

```
-- Enunciado: Mostrar personal que participó en todos los eventos del tipo
'Vacunacion'.
SELECT p.id_personal, p.nombre, p.apellido
FROM rrhh.Personal p
WHERE p.id_personal IN (
    SELECT ape.id_personal
    FROM gestion.Animal_Personal_Evento ape
    INNER JOIN produccion.Evento e ON ape.id_evento = e.id_evento
    WHERE e.tipo_evento = 'Vacunacion'
    GROUP BY ape.id_personal
    HAVING COUNT(DISTINCT ape.id_evento) >= ALL (
        SELECT COUNT(*)
        FROM produccion.Evento
        WHERE tipo_evento = 'Vacunacion'
    )

);
```

Resultados | Mensajes

	id_personal	nombre	apellido
1	2	Jose	Gómez
2	6	Julian	Gómez
3	7	Mario	López

3. Consultas con Funciones Comunes y de Agregados

3.1 Funciones de Texto

Se utilizó el siguiente codigo para obtener las tres primeras letras de la raza en mayúsculas para cada animal (ID 1000 a 1800).

```
-- Ver iniciales de la raza en mayúsculas
SELECT id_animal, UPPER(LEFT(raza,3)) AS inicial_raza, sexo
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	id_animal	inicial_raza	sexo
1	1068	ABE	H
2	1100	ABE	H
3	1315	ABE	M
4	1370	ABE	H
5	1452	ABE	H
6	1503	ABE	H
7	1515	ABE	M
8	1523	ABE	H
9	1530	ABE	M
10	1534	ABE	M

El siguiente codigo se utilizó para concatenar el `id_Animal` con su sexo para generar un detalle descriptivo (ID 1000 a 1800).

```
-- Concatenar caravana (id_animal) + sexo
SELECT id_animal, CONCAT('Caravana: ', id_animal, ' - Sexo: ', sexo) AS detalle
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	id_animal	detalle
1	1068	Caravana: 1068 - Sexo: H
2	1100	Caravana: 1100 - Sexo: H
3	1315	Caravana: 1315 - Sexo: M
4	1370	Caravana: 1370 - Sexo: H
5	1452	Caravana: 1452 - Sexo: H
6	1503	Caravana: 1503 - Sexo: H
7	1515	Caravana: 1515 - Sexo: M
8	1523	Caravana: 1523 - Sexo: H
9	1530	Caravana: 1530 - Sexo: M
10	1534	Caravana: 1534 - Sexo: M

Codigo para obtener la longitud (cantidad de caracteres) de la categoría de cada animal con `id_animal` entre 1000 y 1800.

```
-- Longitud de la categoría (cuántos caracteres tiene la palabra)
SELECT id_animal, categoria, LEN(categoria) AS largo_categoria
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	id_animal	categoria	largo_categoria
1	1068	Vaquillona	10
2	1100	Vaca	4
3	1315	Toro	4
4	1370	Vaquillona	10
5	1452	Vaca	4
6	1503	Vaquillona	10
7	1515	Toro	4
8	1523	Vaquillona	10
9	1530	Toro	4
10	1534	Toro	4

3.2 Funciones Matemáticas

Se utilizó el siguiente codigo para calcular el promedio del peso de los animales con `id_animal` entre 1000 y 1800, mostrando el resultado con dos decimales.

```
-- Promedio del peso de los animales insertados
SELECT CAST(AVG(peso_nacimiento) AS DECIMAL(10,2)) AS promedio_peso
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	promedio_peso
1	509.39

Codigo para calcular el peso total de los animales con `id_animal` entre 1000 y 1800.

```
-- Peso total de los animales insertados
SELECT SUM(peso_nacimiento) AS peso_total
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	peso_total
1	5093.94

Codigo para contar la cantidad de animales por sexo para los animales con `id_animal` entre 1000 y 1800.

```
-- Distribución por sexo
SELECT sexo, COUNT(*) AS cantidad
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800
GROUP BY sexo;
```

Resultados Mensajes

	sexo	cantidad
1	H	6
2	M	4

3.3 Funciones de Fecha

El siguiente codigo extrae el año, mes y día de nacimiento de los animales con `id_animal` entre 1000 y 1800.

```
SELECT id_animal,
       YEAR(fecha_nacimiento) AS anio,
       MONTH(fecha_nacimiento) AS mes,
       DAY(fecha_nacimiento) AS dia
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	id_animal	anio	mes	dia
1	1068	2020	6	17
2	1100	2020	11	14
3	1315	2024	1	31
4	1370	2020	8	11
5	1452	2023	4	25
6	1503	2024	2	26
7	1515	2022	6	30
8	1523	2025	3	13
9	1530	2021	11	15
10	1534	2023	11	11

El siguiente codigo calcula la edad de cada animal en días a partir de su fecha de nacimiento hasta la fecha actual de los animales con `id_animal` entre 1000 y 1800.

```
SELECT id_animal,
       DATEDIFF(DAY, fecha_nacimiento, GETDATE()) AS edad_dias
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	id_animal	edad_dias
1	1068	1902
2	1100	1752
3	1315	579
4	1370	1847
5	1452	860
6	1503	553
7	1515	1159
8	1523	172
9	1530	1386
10	1534	660

El siguiente codigo calcula la fecha estimada de entore agregando 2 años a la fecha de nacimiento de los animales seleccionados de los animales con `id_animal` entre 1000 y 1800.

```
SELECT id_animal,
       fecha_nacimiento,
       DATEADD(YEAR, 2, fecha_nacimiento) AS fecha_entore
FROM produccion.Animal
WHERE id_animal BETWEEN 1000 AND 1800;
```

Resultados Mensajes

	id_animal	fecha_nacimiento	fecha_entore
1	1068	2020-06-17	2022-06-17
2	1100	2020-11-14	2022-11-14
3	1315	2024-01-31	2026-01-31
4	1370	2020-08-11	2022-08-11
5	1452	2023-04-25	2025-04-25
6	1503	2024-02-26	2026-02-26
7	1515	2022-06-30	2024-06-30
8	1523	2025-03-13	2027-03-13
9	1530	2021-11-15	2023-11-15
10	1534	2023-11-11	2025-11-11

4. Funciones Definidas por el Usuario

4.1 Función Escalar

La siguiente función calcula la edad en años de un animal en específico, para ello la función recibe el parámetro `id_animal`.

```
CREATE FUNCTION produccion.fn_EdadAnimal(@id_animal INT)
RETURNS INT
AS
BEGIN
    DECLARE @edad INT;
```

```
    DECLARE @fechaNacimiento DATE;
    DECLARE @hoy DATE = CAST(GETDATE() AS DATE);

    SELECT @fechaNacimiento = fecha_nacimiento
    FROM produccion.Animal
    WHERE id_animal = @id_animal;

    IF @fechaNacimiento IS NULL
        RETURN NULL;

    SET @edad = DATEDIFF(YEAR, @fechaNacimiento, @hoy)
              - CASE
                  WHEN (MONTH(@hoy) < MONTH(@fechaNacimiento))
                       OR (MONTH(@hoy) = MONTH(@fechaNacimiento)
                           AND DAY(@hoy) < DAY(@fechaNacimiento))
                  THEN 1 ELSE 0
                END;

    RETURN @edad;
END;
```

Como ejemplo, la siguiente consulta ejecuta la función:

```
SELECT gestion.fn_EdadAnimal(1530) AS Edad;
```

	Edad
1	3

4.2 Función de tabla en línea

A continuación, se crea la función `produccion.fn_HistoricoPesoAnimal`, que devuelve una tabla ingresando como parámetro el `id_animal`. La función nos permite ver la evolución del peso de un animal específico, brindando la primera y última fecha de medición junto con sus pesos y su categoría.

```
CREATE FUNCTION produccion.fn_HistoricoPesoAnimal (@id_animal INT)

RETURNS TABLE
AS
RETURN
(
    SELECT
        a.id_animal,
        a.categoria,
        DATEDIFF(YEAR, a.fecha_nacimiento, GETDATE()) AS edad,
        MIN(p.fecha_medicion) AS primera_fecha,
        (
            SELECT TOP 1 p2.valor
            FROM produccion.Peso p2
            WHERE p2.id_animal = @id_animal
            ORDER BY p2.fecha_medicion ASC
        ) AS primer_peso,
        MAX(p.fecha_medicion) AS ultima_fecha,
        (
            SELECT TOP 1 p3.valor
```

```
            FROM produccion.Peso p3
            WHERE p3.id_animal = @id_animal
            ORDER BY p3.fecha_medicion DESC
        ) AS ultimo_peso
    FROM produccion.Peso p
    INNER JOIN produccion.Animal a
        ON p.id_animal = a.id_animal
    WHERE p.id_animal = @id_animal
    GROUP BY a.id_animal, a.categoria, a.fecha_nacimiento
);
```

La siguiente consulta ejecuta la función tabla en línea `produccion.fn_HistoricoPesoAnimal` con parámetro @id_animal 8642.

```
SELECT * FROM produccion.fn_HistoricoPesoAnimal(8642);
```

	id_animal	categoria	edad	primera_fecha	primer_peso	ultima_fecha	ultimo_peso
1	8642	Vaca	2	2023-04-04	261.55	2025-05-20	398.58

4.3 Función de tabla multisentencia

Para un control y seguimiento del ganado se crea la función `produccion.fn_AnimalesActivosPorCategoria`, que devuelve una tabla con todos los animales activos ingresando en parámetro una categoría específica, las cuales pueden ser Vaquillona, Vaca o Toro. Devuelve datos como `id_animal, raza, edad` y `ultimo_peso`.

```
CREATE FUNCTION produccion.fn_AnimalesActivosPorCategoria (@categoria
NVARCHAR(50))

RETURNS @Resultado TABLE
(
    id_animal INT,
    raza NVARCHAR(50),
    categoria NVARCHAR(50),
    edad INT,
    ultimo_peso DECIMAL(10,2)
)
AS
BEGIN

    DECLARE @UltimoPeso TABLE
    (
        id_animal INT,
        ultima_fecha DATE
    );

    INSERT INTO @UltimoPeso (id_animal, ultima_fecha)
    SELECT
        p.id_animal,
        MAX(p.fecha_medicion) AS ultima_fecha
    FROM produccion.Peso p
    GROUP BY p.id_animal;

    INSERT INTO @Resultado (id_animal, raza, categoria, edad, ultimo_peso)
    SELECT
        a.id_animal,
```

```
        a.raza,
        a.categoria,
        produccion.fn_EdadAnimal(a.id_animal) AS edad,
        p.valor AS ultimo_peso
    FROM produccion.Animal a
    INNER JOIN @UltimoPeso u
        ON a.id_animal = u.id_animal
    INNER JOIN produccion.Peso p
        ON p.id_animal = u.id_animal
       AND p.fecha_medicion = u.ultima_fecha
    WHERE a.categoria = @categoria
      AND a.estado = 'Activo';

    RETURN;

END;
```

La siguiente consulta ejecuta la función tabla con cuerpo `produccion.fn_AnimalesActivosPorCategoria` con parámetro `@categoria` `vaquillona`.

```
SELECT * FROM produccion.fn_AnimalesActivosPorCategoria('Vaquillona')
ORDER BY edad;
```

	id_animal	raza	categoria	edad	ultimo_peso
1	6197	Aberdeen Angus	Vaquillona	0	250.00
2	1503	Aberdeen Angus	Vaquillona	1	598.28
3	7221	Aberdeen Angus	Vaquillona	1	414.29
4	7279	Aberdeen Angus	Vaquillona	2	488.22
5	3306	Aberdeen Angus	Vaquillona	2	253.36
6	6263	Aberdeen Angus	Vaquillona	2	272.39
7	6392	Aberdeen Angus	Vaquillona	3	220.68
8	5628	Aberdeen Angus	Vaquillona	3	531.35
9	2807	Aberdeen Angus	Vaquillona	3	511.60
10	8612	Aberdeen Angus	Vaquillona	4	324.61
11	9957	Aberdeen Angus	Vaquillona	4	508.89

5. Operaciones de Actualización y Eliminación

5.1 UPDATE

La siguiente consulta actualiza el estado de los animales en la tabla `producción.Animal` a Inactivo si son categoría vaquillona y tienen más de 4 años. Las mismas pasaran al matadero.

```
UPDATE produccion.Animal
SET estado = 'Inactivo'
WHERE DATEDIFF(YEAR, fecha_nacimiento, GETDATE()) > 4
    AND categoria = 'Vaquillona';
```

Para verificar si se realizó algún cambio se ejecuta la siguiente consulta. Se realizaron 6 cambios

```
SELECT id_animal, fecha_nacimiento, estado, categoria
FROM produccion.Animal
```

```
WHERE estado = 'Inactivo';
```

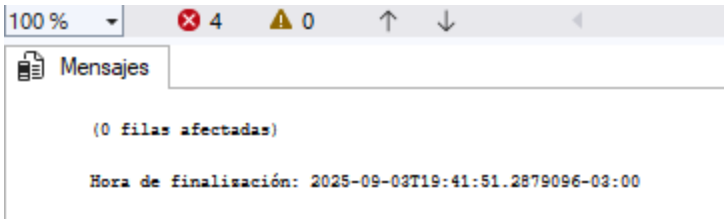
	id_animal	fecha_nacimiento	estado	categoria
1	1068	2020-06-17	Inactivo	Vaquillona
2	1370	2020-08-11	Inactivo	Vaquillona
3	2914	2020-12-07	Inactivo	Vaquillona
4	5170	2020-05-12	Inactivo	Vaquillona
5	7289	2020-12-25	Inactivo	Vaquillona
6	9520	2020-07-22	Inactivo	Vaquillona

5.3 DELETE

Para garantizar que la información no se duplique, se eliminan las mediciones de peso duplicadas que corresponden a un mismo animal y fecha. Se enumera la cantidad de registros definidos en la combinación `id_animal` y `fecha_medición`.

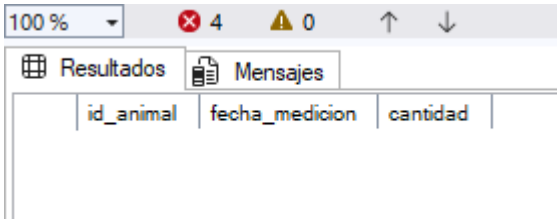
```
;WITH CTE_Duplicados AS (
    SELECT
        *,
        ROW_NUMBER() OVER(PARTITION BY id_animal, fecha_medicion
                          ORDER BY fecha_medicion, id_animal) AS repetido
    FROM produccion.Peso
)
DELETE FROM CTE_Duplicados
WHERE repetido > 1;
```

Al ejecutar la instrucción DELETE, se observa que la consola devuelve “0 filas afectadas”, lo que indica que no existían registros duplicados de pesaje en este caso.



Para confirmarlo se ejecuta la siguiente consulta, confirmando que no hay filas repetidas.

```
SELECT id_animal, fecha_medicion, COUNT(*) AS cantidad
FROM produccion.Peso
GROUP BY id_animal, fecha_medicion
HAVING COUNT(*) > 1;
```



6. Consultas con Funciones de Ventana

6.1 COUNT OVER

Para conocer la cantidad de partos asociados a cada madre se realiza la siguiente consulta con `COUNT(*) OVER(PARTITION BY madre_id`.

```
SELECT
    madre_id, COUNT(*) OVER(PARTITION BY madre_id) AS total_partos
FROM produccion.Nacimiento
ORDER BY madre_id, nro_nacimiento;
```

Su salida es la siguiente:

	madre_id	total_partos
1	2914	2
2	2914	2
3	3306	3
4	3306	3
5	3306	3
6	3511	1
7	4044	3
8	4044	3
9	4044	3
10	4112	3
11	4112	3
12	4112	3
13	4369	2
14	4369	2
15	4538	1
16	4580	2
17	4580	2

6.2 AVG

En la siguiente consulta se calcula el peso promedio de cada animal utilizando `AVG(valor) OVER(PARTITION BY id_animal)`. De esta manera, en cada medición individual aparece el peso promedio correspondiente a cada uno.

```
SELECT
    id_animal,
    fecha_medicion,
    valor,
    ROUND(AVG(valor) OVER(PARTITION BY id_animal), 2) AS peso_promedio
FROM produccion.Peso
ORDER BY id_animal, fecha_medicion;
```

Resultados Mensajes

	id_animal	fecha_medicion	valor	peso_promedio
1	1068	2023-02-04	404.02	406.56
2	1068	2023-05-16	567.24	406.56
3	1068	2024-04-21	339.28	406.56
4	1068	2025-03-04	315.69	406.56
5	1315	2024-07-23	246.54	397.16
6	1315	2024-11-22	557.26	397.16
7	1315	2024-12-19	527.65	397.16
8	1315	2025-01-23	340.82	397.16
9	1315	2025-05-05	313.55	397.16
10	1503	2023-02-03	222.11	422.60
11	1503	2023-05-27	535.12	422.60
12	1503	2023-10-27	355.53	422.60
13	1503	2024-06-04	401.95	422.60
14	1503	2025-07-25	598.28	422.60
15	1515	2023-02-26	396.70	308.28
16	1515	2023-09-28	203.47	308.28
17	1515	2023-10-16	372.33	308.28
18	1515	2025-03-31	260.61	308.28
19	2014	2024-01-22	572.37	489.55
20	2014	2024-09-10	407.08	489.55
21	2014	2024-11-21	489.20	489.55

6.3 ROW_NUMBER

En la siguiente consulta asignamos un número de fila a cada animal en función de la fecha de nacimiento con `ROW_NUMBER() OVER(PARTITION BY categoria)`. Nos permite asignar un número secuencial en función de la antigüedad del animal y no solo ordenar por fecha.

```
SELECT
    id_animal,
    fecha_nacimiento,
    categoria,
    ROW_NUMBER() OVER(PARTITION BY categoria ORDER BY fecha_nacimiento) AS
nro_fila
FROM produccion.Animal;
```

	id_animal	fecha_nacimiento	categoria	nro_fila
1	9006	2022-07-29	T.Hembra	1
2	9010	2022-09-16	T.Hembra	2
3	9013	2023-07-29	T.Hembra	3
4	9017	2023-08-22	T.Hembra	4
5	9014	2024-09-12	T.Hembra	5
6	9003	2024-09-14	T.Hembra	6
7	9008	2025-07-05	T.Hembra	7
8	9001	2025-08-07	T.Hembra	8
9	9019	2025-08-28	T.Hembra	9
10	9011	2025-09-27	T.Hembra	10
11	9020	2021-08-03	T.Macho	1
12	9005	2021-08-05	T.Macho	2
13	9016	2021-09-07	T.Macho	3
14	9004	2021-09-18	T.Macho	4
15	9015	2022-09-18	T.Macho	5
16	9002	2023-08-23	T.Macho	6
17	9018	2024-07-31	T.Macho	7
18	9012	2024-09-03	T.Macho	8
19	9009	2025-08-13	T.Macho	9
20	9007	2025-09-22	T.Macho	10
21	2014	2020-03-12	Toro	1

6.4 RANK Y DENSE RANK

El objetivo de la siguiente consulta es obtener el último registro de cada animal y asignarle un ranking de peso dentro de su categoría. Se aplican `RANK()` y `DENSE_RANK()` para ordenar los animales por peso dentro de cada categoría. La diferencia entre `RANK()` si dos filas tienen valores iguales, se les asigna el mismo rango, y se salta un número en el siguiente, mientras que con `DENSE_RANK()` no salta números cuando hay empates.

```
;WITH UltimoPeso AS (
    SELECT
        id_animal,
        MAX(fecha_medicion) AS ultima_fecha
    FROM produccion.Peso
    GROUP BY id_animal
)
SELECT
    a.categoria,
    a.id_animal,
    p.valor AS peso,
    RANK() OVER(PARTITION BY a.categoria ORDER BY p.valor DESC) AS ranking,
    DENSE_RANK() OVER(PARTITION BY a.categoria ORDER BY p.valor DESC) AS
ranking_denso
FROM UltimoPeso u
JOIN produccion.Peso p
     ON u.id_animal = p.id_animal
    AND u.ultima_fecha = p.fecha_medicion
JOIN produccion.Animal a

     ON u.id_animal = a.id_animal;
```

	categoria	id_animal	peso	ranking	ranking_denso
1	T.Hembra	9010	485.18	1	1
2	T.Hembra	9003	323.20	2	2
3	T.Macho	9005	384.73	1	1
4	Toro	6782	580.40	1	1
5	Toro	7536	565.30	2	2
6	Toro	7828	558.61	3	3
7	Toro	7632	517.44	4	4
8	Toro	2014	489.20	5	5
9	Toro	8619	481.70	6	6
10	Toro	6156	356.40	7	7
11	Toro	3456	344.31	8	8
12	Toro	1315	313.55	9	9
13	Toro	5062	279.02	10	10
14	Toro	1515	260.61	11	11
15	Toro	5637	255.21	12	12
16	Vaca	4369	539.73	1	1
17	Vaca	8642	398.58	2	2
18	Vaca	8546	359.05	3	3
19	Vaca	9179	342.57	4	4
20	Vaca	7597	322.13	5	5
21	Vaca	4538	311.38	6	6
22	Vaca	7719	230.75	7	7

7. Expresiones Comunes de Tabla

7.1 CTE Recursiva (jerarquía madre–hija en animales)

El objetivo es mostrar la genealogía de los animales en distintos niveles de generación, partiendo de las madres registradas.

La tabla `produccion.Animal` tiene una relación reflexiva: cada registro tiene un `id_animal` y opcionalmente un `id_madre` (clave foránea que apunta al `id_animal` de otra fila).

La CTE comienza con los animales que no tienen madre registrada (`id_madre IS NULL`), considerándolos como nivel 0.

Luego, de manera recursiva, se van uniendo los hijos de cada madre encontrada, sumando +1 al nivel en cada paso.

Finalmente, la consulta muestra toda la genealogía ordenada por nivel y por ID de animal.

De esta forma, se puede recorrer toda la jerarquía: madre ➔ hija ➔ nieta ➔ bisnieta, etc.

```
WITH CTE_Familia AS (
    -- Nivel base: madres
    SELECT
        a.id_animal,
        a.raza,
```

```
        a.id_madre,
        0 AS nivel
    FROM produccion.Animal a
    WHERE a.id_madre IS NULL

    UNION ALL

    -- Recursión: hijos de cada madre
    SELECT
        h.id_animal,
        h.raza,
        h.id_madre,
        c.nivel + 1
    FROM produccion.Animal h
    INNER JOIN CTE_Familia c ON h.id_madre = c.id_animal
)
SELECT *
FROM CTE_Familia
ORDER BY nivel, id_animal;
```

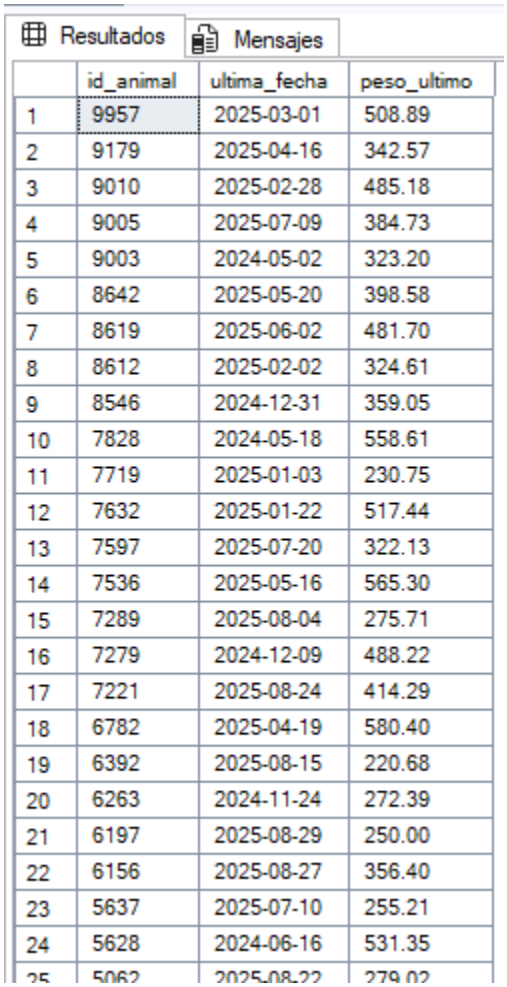
Resultados | Mensajes

	id_animal	raza	id_madre	nivel
96	9512	Aberdeen Angus	NULL	0
97	9520	Aberdeen Angus	NULL	0
98	9710	Aberdeen Angus	NULL	0
99	9882	Aberdeen Angus	NULL	0
100	9957	Aberdeen Angus	NULL	0
101	9001	Aberdeen Angus	2914	1
102	9002	Aberdeen Angus	3306	1
103	9003	Aberdeen Angus	3306	1
104	9004	Aberdeen Angus	3511	1
105	9005	Aberdeen Angus	4044	1
106	9006	Aberdeen Angus	4044	1
107	9007	Aberdeen Angus	4112	1
108	9008	Aberdeen Angus	4112	1
109	9009	Aberdeen Angus	4112	1
110	9010	Aberdeen Angus	4369	1
111	9011	Aberdeen Angus	4580	1
112	9012	Aberdeen Angus	5907	1
113	9013	Aberdeen Angus	6439	1
114	9014	Aberdeen Angus	6439	1
115	9015	Aberdeen Angus	7221	1
116	9016	Aberdeen Angus	7359	1
117	9017	Aberdeen Angus	7359	1
118	9018	Aberdeen Angus	9179	1
119	9019	Aberdeen Angus	9179	1
120	9020	Aberdeen Angus	9447	1

7.2 CTE para simplificar subconsulta (último peso por animal)

El objetivo es obtener el último peso registrado de cada animal de forma clara, evitando subconsultas anidadas. La tabla `produccion.Peso` guarda los registros de peso de cada animal, identificados por `id_animal` y `fecha_medicion` (que forman una clave primaria compuesta). En la CTE `UltimoPeso`, se agrupan los registros por animal y se obtiene la última fecha de medición con `MAX(fecha_medicion)`. Luego, se hace un JOIN con la tabla original para recuperar el valor del peso correspondiente a esa última fecha. El resultado muestra: `id_animal`, la `ultima_fecha` y el `peso_ultimo`.

```
WITH UltimoPeso AS (
    SELECT
        p.id_animal,
        MAX(p.fecha_medicion) AS ultima_fecha
    FROM produccion.Peso p
    GROUP BY p.id_animal
)
SELECT
    u.id_animal,
    u.ultima_fecha,
    p.valor AS peso_ultimo
FROM UltimoPeso u
JOIN produccion.Peso p
  ON u.id_animal = p.id_animal
 AND u.ultima_fecha = p.fecha_medicion;
```



Resultados | Mensajes

	id_animal	ultima_fecha	peso_ultimo
1	9957	2025-03-01	508.89
2	9179	2025-04-16	342.57
3	9010	2025-02-28	485.18
4	9005	2025-07-09	384.73
5	9003	2024-05-02	323.20
6	8642	2025-05-20	398.58
7	8619	2025-06-02	481.70
8	8612	2025-02-02	324.61
9	8546	2024-12-31	359.05
10	7828	2024-05-18	558.61
11	7719	2025-01-03	230.75
12	7632	2025-01-22	517.44
13	7597	2025-07-20	322.13
14	7536	2025-05-16	565.30
15	7289	2025-08-04	275.71
16	7279	2024-12-09	488.22
17	7221	2025-08-24	414.29
18	6782	2025-04-19	580.40
19	6392	2025-08-15	220.68
20	6263	2024-11-24	272.39
21	6197	2025-08-29	250.00
22	6156	2025-08-27	356.40
23	5637	2025-07-10	255.21
24	5628	2024-06-16	531.35
25	5062	2025-08-22	279.02

8. Transformaciones

8.1 PIVOT (cantidad de partos por tipo)

El objetivo es transformar los nacimientos registrados por madre en un formato horizontal, mostrando la cantidad de partos discriminados por tipo. La tabla `produccion.Nacimiento` contiene, entre otros, los campos `madre_id` y `tipo_parto`.

Primero, se seleccionan esas columnas en una subconsulta (`src`). Luego, mediante la cláusula `PIVOT`, se cuentan los registros de cada tipo de parto con `COUNT(tipo_parto)` y se distribuyen en columnas separadas: `[Normal]`, `[Cesárea]`, `[Asistido]`, `[Aborto]` y `[No presenta ternero]`.

El resultado final es una tabla donde cada fila corresponde a una madre y las columnas indican la cantidad de partos de cada tipo.

```
SELECT madre_id, [Normal], [Cesárea], [Asistido], [Aborto], [No presenta ternero]
FROM (
    SELECT madre_id, tipo_parto
```

```
    FROM produccion.Nacimiento
) AS src
PIVOT (
    COUNT(tipo_parto)
    FOR tipo_parto IN ([Normal], [Cesárea], [Asistido], [Aborto], [No presenta
ternero])
) AS p;
```

Resultados Mensajes

	madre_id	Normal	Cesárea	Asistido	Aborto	No presenta ternero
1	2914	1	0	0	0	1
2	3306	0	1	1	0	1
3	3511	0	0	1	0	0
4	4044	1	1	0	0	1
5	4112	1	1	1	0	0
6	4369	0	0	1	0	1
7	4538	0	0	0	0	1
8	4580	0	0	0	1	1
9	5907	0	1	0	0	0
10	6439	1	1	0	0	1
11	7221	0	0	0	1	0
12	7279	0	0	0	0	3
13	7359	0	2	0	0	1
14	8215	0	0	0	0	2
15	8428	0	0	0	0	1
16	8651	0	0	0	0	2
17	9179	1	0	0	1	1
18	9447	1	0	0	0	1
19	9520	0	0	0	0	2

8.2 UNPIVOT (categorías de animales en filas)

El objetivo es convertir columnas fijas en filas, facilitando el análisis dinámico de las categorías de animales según su sexo. La tabla `produccion.Animal` contiene información de cada animal, incluyendo los campos `categoria` y `sexo`. Primero, en una subconsulta (`src`), se cuentan los animales de cada categoría discriminados por sexo:

- `Machos`: usando `COUNT(CASE WHEN sexo = 'M' THEN 1 END)`
- `Hembras`: usando `COUNT(CASE WHEN sexo = 'H' THEN 1 END)`

Luego, con la cláusula `UNPIVOT`, esas columnas (`Machos` y `Hembras`) se transforman en filas, generando dos atributos:

- `atributo` (indica si corresponde a Machos o Hembras)
- `cantidad` (el total en cada caso)

El resultado final muestra por cada `categoria` dos registros:

- Uno con la cantidad de machos
- Otro con la cantidad de hembras.

```
SELECT categoria, atributo, cantidad
FROM (
    SELECT categoria,
           COUNT(CASE WHEN sexo = 'M' THEN 1 END) AS Machos,
           COUNT(CASE WHEN sexo = 'H' THEN 1 END) AS Hembras
    FROM produccion.Animal
    GROUP BY categoria
) AS src
UNPIVOT (
    cantidad FOR atributo IN ([Machos], [Hembras])
) AS unp;
```

Resultados Mensajes

	categoria	atributo	cantidad
1	T.Hembra	Machos	0
2	T.Hembra	Hembras	10
3	T.Macho	Machos	10
4	T.Macho	Hembras	0
5	Toro	Machos	37
6	Toro	Hembras	0
7	Vaca	Machos	0
8	Vaca	Hembras	30
9	Vaquillo...	Machos	0
10	Vaquillo...	Hembras	33

9. Vistas

9.1 Vista de animales activos con último peso

El objetivo es crear una vista que muestre los animales activos junto con su último peso registrado. Se utiliza la tabla `produccion.Animal` (información general del animal) y la tabla `produccion.Peso` (histórico de mediciones de peso). Dentro de la vista, se define la CTE `UltimoPeso`, que obtiene la última fecha de medición para cada animal mediante `MAX(fecha_medicion)`. Luego, la consulta principal combina:

- Los datos de cada animal (`id_animal`, `raza`, `categoria`, `estado`)
- El valor del peso correspondiente a la última fecha (`peso_actual`)
- Y la fecha de esa última medición (`ultima_fecha`).

Finalmente, se filtran únicamente los animales con `estado = 'Activo'`.

```
CREATE VIEW gestion.vw_Animales_Activos_Peso AS
WITH UltimoPeso AS (
    SELECT id_animal, MAX(fecha_medicion) AS ultima_fecha
    FROM produccion.Peso
```

```
    GROUP BY id_animal
)
SELECT
    a.id_animal,
    a.raza,
    a.categoria,
    a.estado,
    p.valor AS peso_actual,
    u.ultima_fecha
FROM produccion.Animal a
JOIN UltimoPeso u ON a.id_animal = u.id_animal
JOIN produccion.Peso p
     ON u.id_animal = p.id_animal AND u.ultima_fecha = p.fecha_medicion
WHERE a.estado = 'Activo';

-- Verificación
SELECT *
FROM gestion.vw_Animales_Activos_Peso;
```

Resultados Mensajes

	id_animal	raza	categoria	estado	peso_actual	ultima_fecha
1	1068	Aberdeen Angus	Vaquillona	Activo	315.69	2025-03-04
2	1315	Aberdeen Angus	Toro	Activo	313.55	2025-05-05
3	1503	Aberdeen Angus	Vaquillona	Activo	598.28	2025-07-25
4	1515	Aberdeen Angus	Toro	Activo	260.61	2025-03-31
5	2014	Aberdeen Angus	Toro	Activo	489.20	2024-11-21
6	2807	Aberdeen Angus	Vaquillona	Activo	511.60	2024-12-11
7	2914	Aberdeen Angus	Vaquillona	Activo	439.67	2024-03-27
8	3306	Aberdeen Angus	Vaquillona	Activo	253.36	2025-05-28
9	3456	Aberdeen Angus	Toro	Activo	344.31	2025-06-09
10	4311	Aberdeen Angus	Vaca	Activo	214.35	2023-10-22
11	4369	Aberdeen Angus	Vaca	Activo	539.73	2025-08-03
12	4538	Aberdeen Angus	Vaca	Activo	311.38	2025-08-17
13	5062	Aberdeen Angus	Toro	Activo	279.02	2025-08-22
14	5628	Aberdeen Angus	Vaquillona	Activo	531.35	2024-06-16
15	5637	Aberdeen Angus	Toro	Activo	255.21	2025-07-10
16	6156	Aberdeen Angus	Toro	Activo	356.40	2025-08-27
17	6197	Aberdeen Angus	Vaquillona	Activo	250.00	2025-08-29
18	6263	Aberdeen Angus	Vaquillona	Activo	272.39	2024-11-24
19	6392	Aberdeen Angus	Vaquillona	Activo	220.68	2025-08-15
20	6782	Aberdeen Angus	Toro	Activo	580.40	2025-04-19
21	7221	Aberdeen Angus	Vaquillona	Activo	414.29	2025-08-24
22	7279	Aberdeen Angus	Vaquillona	Activo	488.22	2024-12-09
23	7289	Aberdeen Angus	Vaquillona	Activo	275.71	2025-08-04
24	7536	Aberdeen Angus	Toro	Activo	565.30	2025-05-16
25	7597	Aberdeen Angus	Vaca	Activo	322.13	2025-07-20

9.2 Vista de eventos con participantes

El objetivo es crear una vista que centralice la información de los eventos realizados, junto con los animales y el personal que participaron en cada uno. La vista relaciona varias tablas:

- `produccion.Evento`, contiene los datos principales del evento (ID, tipo y fecha).
- `gestion.Animal_Personal_Evento`, tabla intermedia que vincula animales y personal con el evento, incluyendo el rol de cada participante.
- `produccion.Animal`, información de los animales involucrados.
- `rrhh.Personal`, datos del personal, combinando nombre y apellido.

La consulta selecciona:

- El identificador, tipo y fecha del evento.
- El ID del animal participante.
- El nombre completo del personal asociado.
- El rol del participante dentro del evento (`rol_evento`).

```sql
CREATE VIEW gestion.vw_Eventos_Con_Participantes AS
SELECT
    e.id_evento,
    e.tipo_evento,
    e.fecha,
    a.id_animal,
    p.nombre + ' ' + p.apellido AS personal,
    ape.rol_evento
FROM produccion.Evento e
JOIN gestion.Animal_Personal_Evento ape ON e.id_evento = ape.id_evento
JOIN produccion.Animal a ON ape.id_animal = a.id_animal
JOIN rrhh.Personal p ON ape.id_personal = p.id_personal;

-- Verificación
SELECT *
FROM gestion.vw_Eventos_Con_Participantes;
```

Resultados Mensajes

	id_evento	tipo_evento	fecha	id_animal	personal	rol_evento
1	1	Vacunacion	2025-02-06	1370	Jose Gómez	Peón auxiliar
2	1	Vacunacion	2025-02-06	1503	Julian Gómez	Encargado supervisor
3	1	Vacunacion	2025-02-06	1068	Mario López	Veterinario a cargo

9.3 Vista de potreros y superficie disponible

El objetivo es crear una vista que muestre únicamente los potreros disponibles, aquellos que cuentan con superficie utilizable. La consulta se realiza sobre la tabla `produccion.Potrero`, que contiene información de cada potrero (identificador, superficie y tipo). Se aplicó un filtro en la cláusula `WHERE` para incluir solamente los registros con `superficie > 0`, descartando así los potreros sin disponibilidad. El resultado de la vista `gestion.vw_Potreros_Disponibles` devuelve:

- `id_potrero`, identificador del potrero.
- `superficie`, extensión disponible.
- `tipo`, clasificación del potrero.

```sql
CREATE VIEW gestion.vw_Potreros_Disponibles AS
SELECT
    id_potrero,
    superficie,
    tipo
FROM produccion.Potrero
WHERE superficie > 0;

-- Verificación
SELECT *
FROM gestion.vw_Potreros_Disponibles;
```

Resultados Mensajes

	id_potrero	superficie	tipo
1	1	90.29	Reserva
2	2	21.03	Pastura
3	3	26.42	Engorde
4	4	103.71	Pastura
5	5	81.64	Pastura
6	6	52.41	Engorde
7	7	65.67	Pastura
8	8	16.22	Engorde
9	9	25.27	Pastura
10	10	83.16	Reserva
11	11	33.37	Reserva
12	12	47.81	Engorde
13	13	17.18	Reserva
14	14	104.12	Engorde
15	15	56.39	Pastura

10. Diagrama de la BD