Universidad Nacional De La Matanza
Especialización Ciencia de Datos

Tópicos de Bases de Datos
Profesor: Julio Bossero

Trabajo Práctico Integrador

Parte 2: Implementación en SQL Server con 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

Objetivo

Implementar en SQL Server el modelo lógico relacional definido en la Parte I, aplicando las sentencias de definición de datos (DDL) vistas en la teoría.

Condición previa al desarrollo

El modelo relacional implementado debe respetar al menos la Tercera Forma Normal (3FN), asegurando la eliminación de redundancias.

Consigna

1. Creación de la Base de Datos y Esquema
   - Crear una base de datos (BD) para el proyecto.
   - El nombre de la BD será **GRUPOXX**
   - Definir al menos tres esquemas diferentes de **dbo** para organizar los objetos.
2. Creación de Tablas

- Crear todas las tablas del modelo relacional definido en la Parte I.
- Definir claves primarias en cada tabla.
- Definir claves foráneas que aseguren la integridad referencial.
- Especificar tipos de datos apropiados para cada columna. Utilice algún IDENTITY.

3. Definición de Restricciones
- Aplicar restricciones NOT NULL donde corresponda.
- Definir al menos una restricción UNIQUE.
- Definir al menos una restricción CHECK.
- Definir al menos una restricción DEFAULT.

4. Índices
- Crear al menos un índice **clustered** y uno **nonclustered** por tabla.
- Incorporar un índice de cobertura (covering index) para consultas frecuentes.

5. Modificación de Tablas (ALTER)
- Agregar una nueva columna a una tabla.
- Modificar el tipo de dato de una columna existente.
- Eliminar una columna de una tabla (que no sea clave primaria o foránea).

6. Eliminación de Objetos
- Eliminar al menos un objeto con DROP.
- Vaciar los datos de una tabla con TRUNCATE.

7. Estructuras Temporales
- Crear una tabla temporal local.
- Crear una tabla temporal global.
- Crear una variable de tabla.

Trate de proponer su uso potencial en el contexto de su sistema, justificando su utilidad.

8. Formato de entrega
- Documento en PDF que incluya:
  - Modelo Relacional Original.
- Scripts de implementación en SQL Server
  - Incluir todas las sentencias DDL utilizadas para crear la base de datos, esquemas, tablas, restricciones, índices y estructuras temporales.
  - Cada script debe estar identificado y comentado, indicando a qué punto de la consigna corresponde.
- Justificación técnica
  - Breve explicación del diseño elegido, incluyendo decisiones sobre tipos de datos, claves, restricciones e índices.
- Nombre de archivo: **TP2_GrupoXX.pdf**.
- Fecha de entrega: **Jueves 29 de agosto**.

DESARROLLO

Modelo Relacional Original

A continuación, se presenta el modelo relacional original desarrollado en el **Trabajo Práctico I**. En los anexos se incluye su representación gráfica, compuesta por:

- Modelo entidad–relación con notación Crow's Foot
- Diagrama DER con notación Chen.

TABLA ANIMAL

Columna	Tipo	Comentario
id_animal	PK	Este campo no es autoincremental, el valor proviene del número físico de la caravana colocada al nacer el animal.
raza		Raza del animal
sexo		Sexo del animal
fecha_nacimiento		Fecha de nacimiento del animal
peso_nacimiento		Peso al nacer del animal
madre_id	FK: ANIMAL(id_animal)	Identificador madre, relación reflexiva
categoria		Vaca, Vaquillona, Toro, T.macho y T.hembra
estado		ejemplo: sano, enfermo, preñado, vendido

Edad: es un atributo derivado, su valor se obtendría mediante una consulta y se calcula en tiempo real a partir de otro atributo (fecha_nacimiento).

Tabla PESO

COLUMNA	Tipo	Comentario
id_animal	PK, FK: ANIMAL(id_animal)	Parte de PK
fecha_medicion	PK	Parte de PK
valor		Peso medido en kilos

PK compuesta: fecha_medicion + id_animal

TABLA NACIMIENTO

Columna	Tipo	Comentario
madre_id	PK, FK: ANIMAL(id_animal)	Parte de PK
nro_nacimiento	PK	Parte de PK - 1º, 2º,3º o 4º parición
fecha		Fecha del nacimiento
tipo_parto		Normal, cesárea, etc.

PK compuesta: id_madre + nro_nacimiento

Tabla POTRERO

Columna	Tipo	Comentario
id_potrero	PK	Identificador de la potrero
superficie		Medida en hectáreas
tipo		Pastura, engorde, reserva

TABLA ANIMAL_POTRERO (RELACIÓN BINARIA N:M CON ATRIBUTO)

Columna	Tipo	Comentario
id_animal	PK, FK: ANIMAL(id_animal)	Parte de PK
id_potrero	PK, FK: POTRERO(id_potrero)	Parte de PK
fecha_asignacion	PK	Parte de PK

PK compuesta: id_animal + id_potrero + fecha_asignacion

TABLA EVENTO

Columna	Tipo	Comentario
id_evento	PK	Identificador único
tipo_evento		Vacunación, remate, inseminación
fecha		
observaciones		

TABLA PERSONAL (SUPERCLASE)

Columna	Tipo	Comentario
id_personal	PK	Identificador único
nombre		Parte del atributo compuesto
apellido		Parte del atributo compuesto
contacto_emerg		Familiar cercano
celular		

Para mantener la integridad referencial, en las subtablas del personal, se elige la estrategia de ON DELETE CASCADE para mantener consistencia si se borra un registro de Personal.

SUBTABLA PEON (especialización disjunta, participación total)

Columna	Tipo	Comentario
id_personal	PK, FK: PERSONAL (id_personal)	
sector_asignado		
turno		

SUBTABLA CAPATAZ (especialización disjunta, participación total)

Columna	Tipo	Comentario
id_personal	PK, FK: PERSONAL (id_personal)	
anios_experiencia		

SUBTABLA TRACTORISTA (especialización disjunta, participación total)

Columna	Tipo	Comentario
id_personal	PK, FK: PERSONAL (id_personal)	
licencia_nro		
tipo_maquinaria		Tractor, Sembradora, Disco, Rolo, etc.

SUBTABLA VETERINARIO (especialización disjunta, participación total)

Columna	Tipo	Comentario
id_personal	PK, FK: PERSONAL (id_personal)	
mat_profesional		Matricula del profesional
especialidad		Inseminación, Reproducción, Clínica, etc

SUBTABLA ENCARGADO

Columna	Tipo	Comentario
id_personal	PK, FK: PERSONAL (id_personal)	
ppto_asignado		Presupuesto asignado
cant_personal		Cantidad de personal a cargo

TABLA ANIMAL_PERSONAL_EVENTO (RELACIÓN TERNARIA CON ATRIBUTO)

Columna	Tipo	Comentario
id_animal	PK, FK: ANIMAL(id_animal)	Parte de PK
id_personal	PK, FK: PERSONAL(id_personal)	Parte de PK
id_evento	PK, FK: EVENTO(id_evento)	Parte de PK
rol_en_evento		Rol del personal en ese evento

PK compuesta: id_animal + id_personal + id_evento

TABLA POTRERO_EVENTO

Columna	Tipo	Comentario
id_potrero	PK, FK: POTRERO(id_potrero)	Parte de PK
id_evento	PK, FK: EVENTO(id_evento)	Parte de PK

PK compuesta: id_potrero + id_evento

Script Completo

A lo largo de este trabajo se irá detallando el código utilizado para resolver cada consigna. El script completo puede encontrarse en el archivo **TP2_GrupoVI.sql,** adjunto junto con el presente PDF en la plataforma MIeL.

1. Creación de la Base de Datos y Esquema

Para comenzar con la implementación, se procedió a crear la base de datos denominada **GRUPOVI** y a seleccionarla como la base de trabajo.

Las sentencias utilizadas fueron:

```
CREATE DATABASE GRUPOVI;
GO

USE GRUPOVI;
GO
```

Posteriormente, se procedió a definir tres **esquemas dentro de la base de datos** GRUPOVI, correspondientes a las áreas de producción, recursos humanos y gestión.

La elección de estos esquemas responde a una necesidad de organización lógica y modularidad dentro de la base de datos. Al asignar objetos a diferentes esquemas:

- **producción**: concentra las tablas, vistas e índices vinculados al registro y seguimiento de la actividad productiva (ej. control de animales, potreros, pesos, etc.).
- **rrhh (recursos humanos)**: reúne la información relacionada con el personal, roles y su participación en los procesos productivos.
- **gestión**: funciona como un espacio de integración, donde se administran las relaciones entre recursos humanos, animales y eventos productivos, sirviendo de nexo entre los otros dos esquemas.

Las sentencias utilizadas fueron:

```
CREATE SCHEMA produccion;
GO

CREATE SCHEMA rrhh;
GO

CREATE SCHEMA gestion;
GO
```

2. Creación de Tablas

Una vez definidos los esquemas, se procedió a la creación de las tablas correspondientes a cada uno de ellos (`produccion`, `rrhh` y `gestion`).

Los tipos de datos utilizados son:

- `INT`: número entero, sin decimales.
- `DECIMAL (p,s)`: número con decimales, donde p es la precisión total y s los decimales.
- `VARCHAR(n)`: cadena de texto de longitud variable, hasta n caracteres.
- `CHAR(n)`: carácter fijo de longitud n.
- `DATE`: almacena una fecha (día, mes, año).

Las restricciones como claves primarias, foráneas, valores por defecto y validaciones específicas se describen en detalle en el apartado Definición de Restricciones.

2.1 Esquema Producción

En este esquema se modelan las entidades vinculadas a los animales, sus nacimientos, pesos, potreros y eventos asociados. Indicando el tipo de dato de cada columna y su rol dentro de la tabla.

2.1.1 Tabla Animal

La tabla `produccion.Animal` guarda los datos generales de cada ejemplar, incluyendo raza, sexo, fecha y peso de nacimiento, categoría y estado, además de una autorreferencia que permite identificar a la madre.

```
CREATE TABLE produccion.Animal (
    id_animal INT PRIMARY KEY,
    raza VARCHAR(45) NOT NULL,
    sexo CHAR(1) NOT NULL CHECK (sexo IN ('M','H')),
    fecha_nacimiento DATE NOT NULL,
    peso_nacimiento DECIMAL(5,2) NULL CHECK (peso_nacimiento > 0),
    id_madre INT NULL,
    categoria VARCHAR(45) NOT NULL,
    estado VARCHAR(45) NULL DEFAULT 'Activo',
    CONSTRAINT FK_Animal_Madre FOREIGN KEY (id_madre) REFERENCES
produccion.Animal(id_animal)
);
GO
```

Nótese que el atributo `edad`, si bien fue representado en el DER por motivos conceptuales, no se materializa en la base de datos al tratarse de un atributo derivado. Su valor se obtiene dinámicamente a partir de `fecha_nacimiento` mediante consultas SQL.

2.1.2 Tabla Nacimiento

La tabla `produccion.Nacimiento` registra los distintos partos de cada madre, indicando número, fecha y tipo de parto, asegurando que una misma madre no pueda tener duplicados en el orden de nacimientos.

```
CREATE TABLE produccion.Nacimiento (

    nro_nacimiento INT NOT NULL CHECK (nro_nacimiento BETWEEN 1 AND 15),
    madre_id INT NOT NULL,
    fecha DATE NOT NULL DEFAULT GETDATE(),
    tipo_parto VARCHAR(20) NOT NULL
        CHECK (tipo_parto IN ('Normal', 'Cesárea', 'Asistido', 'Aborto', 'No
presenta ternero')),
    CONSTRAINT PK_Nacimiento PRIMARY KEY (madre_id, nro_nacimiento),
    CONSTRAINT FK_Nacimiento_Madre FOREIGN KEY (madre_id) REFERENCES
produccion.Animal(id_animal)
);
GO
```

2.1.3 Tabla Peso

La tabla `produccion.Peso` almacena mediciones periódicas del peso de los animales, permitiendo un historial cronológico.

```
CREATE TABLE produccion.Peso (
    fecha_medicion DATE NOT NULL DEFAULT GETDATE(),
    id_animal INT NOT NULL,
```

```
    valor DECIMAL(5,2) NOT NULL CHECK (valor > 0),
    CONSTRAINT PK_Peso PRIMARY KEY (fecha_medicion, id_animal),
    CONSTRAINT FK_Peso_Animal FOREIGN KEY (id_animal) REFERENCES
produccion.Animal(id_animal)
);
GO
```

2.1.4 Tabla Potrero

La tabla `produccion.Potrero` define los potreros disponibles con su superficie y tipo.

```
CREATE TABLE produccion.Potrero (
    id_potrero INT IDENTITY(1,1) PRIMARY KEY,
    superficie DECIMAL(6,2) NOT NULL CHECK (superficie > 0),
    tipo VARCHAR(100) NOT NULL,
);
GO
```

2.1.5 Tabla Animal_Potrero

La tabla `produccion.Animal_Potrero` establece la asignación de animales a potreros con control de fechas válidas.

```
CREATE TABLE produccion.Animal_Potrero (
    fecha_asignacion DATE NOT NULL DEFAULT GETDATE(),
    id_animal INT NOT NULL,
    id_potrero INT NOT NULL,

    CONSTRAINT PK_Animal_Potrero PRIMARY KEY (fecha_asignacion, id_animal,
id_potrero),

    CONSTRAINT FK_AnimalPotrero_Animal FOREIGN KEY (id_animal) REFERENCES
produccion.Animal(id_animal),

    CONSTRAINT FK_AnimalPotrero_Potrero FOREIGN KEY (id_potrero) REFERENCES
produccion.Potrero(id_potrero),

    CONSTRAINT CHK_Fecha_Asignacion CHECK (fecha_asignacion <= GETDATE())
);
GO
```

2.1.6 Tabla Evento

La tabla `produccion.Evento` registra sucesos relacionados con la gestión productiva, cuestiones sanitarias, procesos reproductivos o administrativos, junto con observaciones.

```
CREATE TABLE produccion.Evento (
    id_evento INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
    tipo_evento VARCHAR(100) NOT NULL DEFAULT 'General',
    fecha DATE NOT NULL DEFAULT GETDATE(),
    observaciones VARCHAR(300) NOT NULL
);
GO
```

2.2 Esquema RRHH

Este esquema concentra la información sobre el personal y sus roles específicos: Peón, Capataz, Tractorista, Encargado y Veterinario.

2.2.1 Tabla Personal

La tabla `rrhh.Personal` centraliza los datos básicos de identificación, contacto y celular.

```
CREATE TABLE rrhh.Personal (
    id_personal INT IDENTITY(1,1) PRIMARY KEY CLUSTERED,
    nombre VARCHAR(50) NOT NULL,
    apellido VARCHAR(50) NOT NULL,
    contacto_emerg VARCHAR(80) NOT NULL,
    celular VARCHAR(18) NOT NULL,

    CONSTRAINT CHK_Celular CHECK (LEN(celular) >= 8)
);
GO
```

A partir de ella se derivan las tablas de roles específicos: `Peón`, `Capataz`, `Tractorista`, `Encargado` y `Veterinario`, cada una con los atributos particulares de su función (turno, experiencia, licencias, presupuesto, matrícula, especialidad, entre otros).

2.2.2 Tabla Peón

```
CREATE TABLE rrhh.Peon (
    id_personal INT PRIMARY KEY CLUSTERED,
    sector_asignado VARCHAR(50) NULL,
    turno VARCHAR(20) NULL,

    CONSTRAINT FK_Peon_Personal FOREIGN KEY (id_personal)
        REFERENCES rrhh.Personal(id_personal)
        ON DELETE CASCADE,

    CONSTRAINT CHK_Turno CHECK (turno IN ('Mañana','Tarde','Noche') OR turno IS
NULL)
);
GO
```

2.2.3 Tabla Capataz

```
CREATE TABLE rrhh.Capataz (
    id_personal INT PRIMARY KEY CLUSTERED,
    anios_experiencia INT NULL,
    CONSTRAINT FK_capataz_personal
        FOREIGN KEY (id_personal)
        REFERENCES rrhh.Personal (id_personal)
        ON DELETE CASCADE
);
```

2.2.4 Tabla Tractorista

```
CREATE TABLE rrhh.Tractorista (
    id_personal INT NOT NULL PRIMARY KEY CLUSTERED,
    licencia_nro VARCHAR(45) NOT NULL,
    tipo_maquinaria VARCHAR(100) NOT NULL,

    CONSTRAINT FK_Tractorista_Personal FOREIGN KEY (id_personal)
        REFERENCES rrhh.personal (id_personal)
        ON DELETE CASCADE,

    CONSTRAINT UQ_Tractorista_Licencia UNIQUE (licencia_nro),

    CONSTRAINT CK_Tractorista_Licencia CHECK (LEN(licencia_nro) >= 8)
);
GO
```

2.2.5 Tabla Encargado

```
CREATE TABLE rrhh.Encargado (
    id_personal INT NOT NULL PRIMARY KEY CLUSTERED,
    ppto_asignado DECIMAL(10,2) NULL,
    cant_personal INT NOT NULL,

    CONSTRAINT FK_Encargado_Personal FOREIGN KEY (id_personal)
        REFERENCES rrhh.personal (id_personal)
        ON DELETE CASCADE,

    CONSTRAINT CK_Encargado_Cant_Personal
    CHECK (cant_personal >= 0),

    CONSTRAINT CK_Encargado_Ppto_Asignado
    CHECK (ppto_asignado IS NULL OR ppto_asignado >= 0)
);
GO
```

2.2.6 Tabla Veterinario

```
CREATE TABLE rrhh.Veterinario (
    id_personal INT PRIMARY KEY CLUSTERED,
    mat_profesional VARCHAR(60) NOT NULL,
    especialidad VARCHAR(60) NOT NULL,

    CONSTRAINT FK_Veterinario_Personal
        FOREIGN KEY (id_personal)
        REFERENCES rrhh.Personal(id_personal)
        ON DELETE CASCADE,

    CONSTRAINT UQ_Veterinario_Matricula UNIQUE(mat_profesional)
);
GO
```

2.3 Esquema Gestión

En el esquema `Gestión` se definen las tablas que vinculan los recursos productivos, humanos y eventos.

2.3.1 Tabla Animal_Personal_Evento

La tabla `gestión.Animal_Personal_Evento` permite registrar qué personas participaron en determinados eventos y sobre qué animal, indicando el rol desempeñado.

```
CREATE TABLE gestion.Animal_Personal_Evento (
    id_animal INT NOT NULL,
    id_personal INT NOT NULL,
    id_evento INT NOT NULL,
    rol_evento VARCHAR(50) NOT NULL,
    CONSTRAINT PK_Animal_Personal_Evento PRIMARY KEY CLUSTERED (id_animal,
id_personal, id_evento),

    CONSTRAINT FK_APE_Animal FOREIGN KEY (id_animal)
        REFERENCES produccion.Animal(id_animal)
        ON DELETE NO ACTION,

    CONSTRAINT FK_APE_Personal FOREIGN KEY (id_personal)
        REFERENCES rrhh.Personal(id_personal)
        ON DELETE NO ACTION,

    CONSTRAINT FK_APE_Evento FOREIGN KEY (id_evento)
        REFERENCES produccion.Evento(id_evento)
        ON DELETE NO ACTION
);
GO
```

2.3.2 Tabla Potrero_Evento

La tabla `gestión.Potrero_Evento` relaciona los potreros con los eventos ocurridos en ellos.

```
CREATE TABLE gestion.Potrero_Evento (
    id_potrero INT NOT NULL,
    id_evento INT NOT NULL,
    CONSTRAINT PK_Potrero_Evento PRIMARY KEY CLUSTERED (id_potrero, id_evento),

    CONSTRAINT FK_PE_Potrero FOREIGN KEY (id_potrero)
        REFERENCES produccion.Potrero(id_potrero)
        ON DELETE NO ACTION,

    CONSTRAINT FK_PE_Evento FOREIGN KEY (id_evento)
        REFERENCES produccion.Evento(id_evento)
        ON DELETE NO ACTION
);
GO
```

3. Definición de Restricciones

En este apartado se detallan las restricciones implementadas en las distintas tablas para garantizar la integridad y consistencia de los datos.

3.1 Esquema Producción

3.1.1 Animal

- `PRIMARY KEY (id_animal)`: Identificador único de cada animal.
- `CHECK (sexo IN ('M','H'))`: Controla que el sexo solo pueda ser M (macho) o H (hembra).
- `CHECK (peso_nacimiento > 0)`: Impide registrar pesos negativos.
- `DEFAULT 'Activo'` en `estado`: valor por defecto, ya que de momento que se carga en la base de datos ya se encuentra activo.
- `FOREIGN KEY (id_madre)`: Autorreferencia al animal madre.

3.1.2 Nacimiento

- `PRIMARY KEY (madre_id, nro_nacimiento)`: Identificador único de cada nacimiento por madre. Evita que una misma madre tenga dos partos con el mismo número.
- `CHECK (nro_nacimiento BETWEEN 1 AND 15)`: Controla que el número de nacimiento esté entre 1 y 15 (máximo 15 partos por madre).
- `CHECK (tipo_parto IN ('Normal', 'Cesárea', 'Asistido', 'Aborto', 'No presenta ternero'))`: Solo se permiten estos valores como tipo de parto.
- `DEFAULT GETDATE() en fecha`: Si no se carga fecha, por defecto se asigna la fecha actual.
- `FOREIGN KEY (madre_id)`: Cada nacimiento debe estar asociado a un animal registrado como madre en la tabla Animal.

3.1.3 Peso

- `PRIMARY KEY (fecha_medicion, id_animal)`: Identificador único de cada registro de peso por animal y fecha. Impide que un mismo animal tenga dos registros en la misma fecha.
- `CHECK (valor > 0)`: Garantiza que no se registren pesos nulos o negativos.
- `DEFAULT GETDATE() en fecha_medicion`: Si no se carga una fecha, por defecto se toma la fecha de ese mismo día.
- `FOREIGN KEY (id_animal)`: Cada registro de peso debe estar asociado a un animal existente en la tabla Animal.

3.1.4 Potrero

- `IDENTITY(1,1)` (`id_potrero`): genera automáticamente valores consecutivos, comenzando en 1 e incrementando de a 1, lo que simplifica la asignación de identificadores.
- `PRIMARY KEY (id_potrero)`: Identificador único para cada potrero.
- `CHECK (superficie > 0)`: Controla que la superficie cargada sea siempre un valor positivo.

- `NOT NULL (tipo)`: Obliga a registrar siempre un tipo de potrero.

3.1.5 Animal_Potrero

- `PRIMARY KEY (fecha_asignacion, id_animal, id_potrero)`: Identificador único que asegura que un mismo animal no pueda ser asignado dos veces al mismo potrero en la misma fecha.
- `FOREIGN KEY (id_animal)`: Relación con la tabla Animal, garantizando que el animal exista en el sistema antes de asignarlo a un potrero.
- `FOREIGN KEY (id_potrero)`: Relación con la tabla Potrero, asegurando que el potrero exista antes de asignarle un animal.
- `CHECK (fecha_asignacion <= GETDATE())`: Controla que la fecha de asignación no sea futura, solo actual o pasada.
- `DEFAULT GETDATE() en fecha_asignacion`: Si no se especifica, se carga automáticamente la fecha de ese día como asignación.

3.1.6 Evento

- `IDENTITY(1,1) (id_evento)`: genera automáticamente valores consecutivos, comenzando en 1 e incrementando de a 1, lo que simplifica la asignación de identificadores.
- `PRIMARY KEY (id_evento)`: clave primaria clustered, definida como un identificador único para cada evento.
- `NOT NULL en tipo_evento`: Obliga a que todo evento tenga un tipo definido.
- `DEFAULT 'General' en tipo_evento`: Si no se especifica, el sistema lo registra automáticamente como evento de tipo General.
- `NOT NULL en fecha`: Garantiza que siempre quede registrada la fecha del evento.
- `DEFAULT GETDATE() en fecha`: Si no se indica otra, se carga por defecto la fecha actual.
- `NOT NULL en observaciones`: Asegura que cada evento tenga al menos una descripción u observación registrada.

3.2 Esquema RRHH

3.2.1 Personal

- `IDENTITY(1,1) (id_personal)`: genera automáticamente valores consecutivos, comenzando en 1 e incrementando de a 1, lo que simplifica la asignación de identificadores.
- `PRIMARY KEY CLUSTERED (id_personal)`: Identificador único para cada persona registrada en el sistema. Organiza físicamente los registros de la tabla en disco, siguiendo este índice y ordenándolos por `id_personal`.
- `NOT NULL en nombre`: Asegura que siempre se registre el nombre del personal.
- `NOT NULL en apellido`: Obliga a registrar el apellido de la persona.

- `NOT NULL en contacto_emerg`: Exige que se cargue un contacto de emergencia para cada persona.
- `NOT NULL en celular`: Impide que quede vacío el número de celular.
- `CHECK (LEN(celular) >= 8)`: Controla que el celular cargado tenga al menos 8 caracteres, evitando registros incompletos.

3.2.2 Peón

- `PRIMARY KEY CLUSTERED (id_personal)`: Identificador único del peón, heredado de la tabla Personal. Organiza físicamente los registros de la tabla en disco, siguiendo este índice y ordenándolos por `id_personal`.
- `FOREIGN KEY (id_personal)`: Relación con la tabla Personal, asegurando que cada peón esté previamente registrado como personal.
- `ON DELETE CASCADE`: Si se elimina un registro de Personal, automáticamente se eliminan también sus registros en Peón. Garantiza integridad en la especialización.
- `NULL en sector_asignado`: Permite que inicialmente no se defina el sector asignado.
- `NULL en turno`: Se admite que un peón no tenga turno definido.
- `CHECK (turno IN ('Mañana','Tarde','Noche') OR turno IS NULL)`: Restringe que el turno solo pueda ser uno de los tres valores válidos, o quedar vacío.

3.2.3 Capataz

- `PRIMARY KEY CLUSTERED (id_personal)`: Identificador único del capataz, heredado desde la tabla Personal. Organiza físicamente los registros de la tabla en disco, siguiendo este índice y ordenándolos por `id_personal`.
- `FOREIGN KEY (id_personal)`: Relación con la tabla Personal, asegurando que cada capataz sea también un miembro del personal.
- `ON DELETE CASCADE`: Si se elimina un registro de Personal, automáticamente se eliminan también sus registros en Capataz. Garantiza integridad en la especialización.
- `NULL en anios_experiencia`: Permite registrar capataces sin especificar la experiencia al momento de la carga.

3.2.4 Tractorista

- `PRIMARY KEY CLUSTERED (id_personal)`: Identificador único del tractorista, heredado de la tabla Personal. Organiza físicamente los registros de la tabla en disco, siguiendo este índice y ordenándolos por `id_personal`.
- `FOREIGN KEY (id_personal)`: Relación con la tabla Personal, asegurando que cada tractorista sea un miembro del personal registrado.
- `ON DELETE CASCADE`: Si se elimina un registro de Personal, automáticamente se eliminan también sus registros en Tractorista. Garantiza integridad en la especialización.

- `NOT NULL en licencia_nro`: Obliga a que cada tractorista tenga un número de licencia registrado.
- `UNIQUE (licencia_nro)`: Garantiza que no haya dos tractoristas con la misma licencia.
- `CHECK (LEN(licencia_nro) >= 8)`: Controla que el número de licencia tenga al menos 8 caracteres, evitando registros incompletos.
- `NOT NULL en tipo_maquinaria`: Obliga a especificar el tipo de maquinaria que opera cada tractorista.

3.2.5 Encargado

- `PRIMARY KEY CLUSTERED (id_personal)`: Identificador único del encargado, heredado de la tabla Personal. Organiza físicamente los registros de la tabla en disco, siguiendo este índice y ordenándolos por `id_personal`.
- `FOREIGN KEY (id_personal)`: Relación con la tabla Personal, asegurando que cada encargado sea un miembro del personal registrado.
- `ON DELETE CASCADE`: Si se elimina un registro de Personal, automáticamente se eliminan también sus registros en Encargado. Garantiza integridad en la especialización.
- `NOT NULL en cant_personal`: Obliga a registrar la cantidad de personal a su cargo.
- `CHECK (cant_personal >= 0)`: Controla que la cantidad de personal no sea negativa.
- `NULL en ppto_asignado`: Permite que inicialmente no se cargue presupuesto asignado.
- `CHECK (ppto_asignado IS NULL OR ppto_asignado >= 0)`: Garantiza que el presupuesto, si se ingresa, sea siempre un valor positivo.

3.2.6 Veterinario

- `PRIMARY KEY CLUSTERED (id_personal)`: Identificador único del veterinario, heredado de la tabla Personal. Organiza físicamente los registros de la tabla en disco, siguiendo este índice y ordenándolos por `id_personal`.
- `FOREIGN KEY (id_personal)`: Relación con la tabla Personal, asegurando que cada veterinario sea un miembro del personal registrado.
- `ON DELETE CASCADE`: Si se elimina un registro de Personal, automáticamente se eliminan también sus registros en Veterinario. Garantiza integridad en la especialización.
- `NOT NULL en mat_profesional`: Obliga a registrar la matrícula profesional de cada veterinario.
- `UNIQUE (mat_profesional)`: Garantiza que no haya dos veterinarios con la misma matrícula profesional.
- `NOT NULL en especialidad`: Obliga a registrar la especialidad de cada veterinario.
- `Índice nonclustered en especialidad`: Facilita búsquedas rápidas por la especialidad del veterinario.

3.3 Esquema Gestión

3.3.1 Animal_Personal_Evento

- `PRIMARY KEY CLUSTERED (id_animal, id_personal, id_evento)`: Identificador único que asegura que un mismo animal, personal y evento no se repitan en un mismo registro. Organiza físicamente los registros de la tabla en disco siguiendo este índice, ordenados por `id_animal`, `id_personal` y `id_evento`.
- `FOREIGN KEY (id_animal)`: Relación con la tabla Animal, asegurando que el animal exista antes de registrar su participación en un evento.
- `FOREIGN KEY (id_personal)`: Relación con la tabla Personal, garantizando que la persona exista antes de asignarle un rol en un evento.
- `FOREIGN KEY (id_evento)`: Relación con la tabla Evento, asegurando que el evento exista antes de asignar animales y personal.
- `ON DELETE NO ACTION`: Evita la eliminación de animales, personal o eventos si están asociados en esta tabla.
- `NOT NULL en rol_evento`: Obliga a registrar el rol que cumple la persona o el animal en el evento (por ejemplo: “Asistente”, “Observador”, “Aplicación de vacuna”).

3.3.2 Potrero_Evento

- `PRIMARY KEY CLUSTERED (id_potrero, id_evento)`: Identificador único que asegura que un mismo potrero no se registre dos veces en el mismo evento. Organiza físicamente los registros de la tabla en disco siguiendo este índice, ordenados por `id_potrero` y `id_evento`.
- `FOREIGN KEY (id_potrero)`: Relación con la tabla Potrero, asegurando que el potrero exista antes de asociarlo a un evento.
- `FOREIGN KEY (id_evento)`: Relación con la tabla Evento, garantizando que el evento exista antes de asociar potreros.
- `ON DELETE NO ACTION`: Evita que se eliminen registros de potreros o eventos si están asociados en esta tabla.

4. Índices

4.1 Índice clustered

Para garantizar la unicidad de los registros, al crear cada una de las tablas de la base de datos **GRUPOVI**, se definieron claves primarias (`PRIMARY KEY`).

Cada una de estas claves automáticamente genera un índice **clustered**, permitiendo que los datos se almacenen físicamente en el disco siguiendo el orden de dicha clave. De esta forma, se optimiza el acceso y la búsqueda de registros en cada una de las tablas creadas.

Particularmente, las tablas con `primary key compuesta`, como es el caso de la tabla `Peso` (`fecha_medicion, id_animal`), el índice clustered ordena los registros físicamente por ambas columnas, lo que facilita consultas históricas y filtrados por animal y fecha, mejorando significativamente el rendimiento de este tipo de búsquedas.

4.2 Índice nonclustered

IX_Animal_categoria

En la tabla `produccion.Animal` se creó el índice `IX_Animal_categoria`, definido sobre la columna `categoria`. Este índice permite acelerar las consultas frecuentes que filtran o agrupan animales según su categoría (por ejemplo: traer todos los animales de la categoría vaquillona). El índice mantiene una estructura auxiliar que apunta a los registros de `Animal`.

```
CREATE NONCLUSTERED INDEX IX_Animal_categoria
ON produccion.Animal(categoria);
GO
```

IX_Animal_sexo_estado

En la tabla `produccion.Animal` se creó el índice `IX_Animal_sexo_estado`, definido sobre las columnas `sexo` y `estado`. Este índice acelera las consultas que filtran animales combinando ambas condiciones (por ejemplo: todos los machos que están en engorde). La estructura ordenada del índice permite acceder rápidamente a los registros que cumplen esas características.

```
CREATE NONCLUSTERED INDEX IX_Animal_sexo_estado
ON produccion.Animal(sexo, estado);
GO
```

IX_Nacimiento_madre

En la tabla `produccion.Nacimiento` se creó el índice `IX_Nacimiento_madre`, definido sobre la columna `madre_id`. Este índice permite consultas rápidas para obtener todos los partos asociados a una madre en particular (por ejemplo: listar todas las crías de la vaca con ID 120).

```
CREATE NONCLUSTERED INDEX IX_Nacimiento_madre
ON produccion.Nacimiento(madre_id);
GO
```

IX_Nacimiento_fecha_tipo

En la tabla `produccion.Nacimiento` se creó el índice `IX_Nacimiento_fecha_tipo`, definido sobre las columnas `fecha` y `tipo_parto`. Este índice acelera los reportes que analizan nacimientos por periodo de tiempo y tipo de parto (por ejemplo: cantidad de partos por cesárea en 2023).

```
CREATE NONCLUSTERED INDEX IX_Nacimiento_fecha_tipo
```

```
ON produccion.Nacimiento(fecha, tipo_parto);
GO
```

IX_Peso_animal_fecha

En la tabla `produccion.Peso` se creó el índice `IX_Peso_animal_fecha`, definido sobre las columnas `id_animal` y `fecha_medicion`. Este índice facilita recuperar rápidamente el historial de peso de un animal en orden temporal (por ejemplo: evolución de peso del animal 305).

```
CREATE NONCLUSTERED INDEX IX_Peso_animal_fecha
ON produccion.Peso(id_animal, fecha_medicion);
GO
```

IX_Potrero_tipo

En la tabla `produccion.Potrero` se creó el índice `IX_Potrero_tipo`, definido sobre la columna `tipo`. Este índice agiliza las búsquedas de potreros por su clasificación (por ejemplo: listar todos los potreros de pastura natural).

```
CREATE NONCLUSTERED INDEX IX_Potrero_tipo
ON produccion.Potrero(tipo);
GO
```

IX_Potrero_superficie

En la tabla `produccion.Potrero` se creó el índice `IX_Potrero_superficie`, definido sobre la columna `superficie`. Este índice permite filtrar o generar reportes de potreros según su tamaño (por ejemplo: buscar potreros con más de 50 hectáreas).

```
CREATE NONCLUSTERED INDEX IX_Potrero_superficie
ON produccion.Potrero(superficie);
GO
```

IX_AnimalPotrero_id_animal

En la tabla `produccion.Animal_Potrero` se creó el índice `IX_AnimalPotrero_id_animal`, definido sobre la columna `id_animal`. Este índice acelera la consulta en qué potrero estuvo o está asignado un animal específico.

```
CREATE NONCLUSTERED INDEX IX_AnimalPotrero_id_animal
ON produccion.Animal_Potrero(id_animal);
GO
```

IX_AnimalPotrero_id_potrero

En la tabla `produccion.Animal_Potrero` se creó el índice `IX_AnimalPotrero_id_potrero`, definido sobre la columna `id_potrero`. Este índice facilita listar todos los animales que se encuentran (o estuvieron) en un potrero determinado.

```
CREATE NONCLUSTERED INDEX IX_AnimalPotrero_id_potrero
ON produccion.Animal_Potrero(id_potrero);
GO
```

IX_Evento_tipo_fecha

En la tabla `produccion.Evento` se creó el índice `IX_Evento_tipo_fecha`, definido sobre `tipo_evento` y `fecha`. Este índice acelera las consultas que filtran actividades según el tipo y la fecha (por ejemplo: ver todos los servicios realizados en mayo de 2024).

```
CREATE NONCLUSTERED INDEX IX_Evento_tipo_fecha
ON produccion.Evento(tipo_evento, fecha);
GO
```

IX_Personal_NombreApellido

En la tabla `rrhh.Personal` se creó el índice `IX_Personal_NombreApellido`, definido sobre `nombre` y `apellido`. Este índice permite búsquedas rápidas de personas en función de sus datos personales (por ejemplo: encontrar a “Juan Pérez” en la base de empleados).

```
CREATE NONCLUSTERED INDEX IX_Personal_NombreApellido
ON rrhh.Personal(nombre, apellido);
GO
```

IX_Peon_Sector

En la tabla `rrhh.Peon` se creó el índice `IX_Peon_Sector`, definido sobre `sector_asignado`. Este índice acelera la búsqueda de peones en función del sector donde trabajan (por ejemplo: listar todos los peones asignados a la siembra).

```
CREATE NONCLUSTERED INDEX IX_Peon_Sector
ON rrhh.Peon(sector_asignado);
GO
```

IX_Capataz_Experiencia

En la tabla `rrhh.Capataz` se creó el índice `IX_Capataz_Experiencia`, definido sobre `anios_experiencia`. Este índice facilita obtener capataces según sus años de experiencia (por ejemplo: seleccionar los que tienen más de 10 años).

```
CREATE NONCLUSTERED INDEX IX_Capataz_Experiencia
ON rrhh.Capataz (anios_experiencia);
```

IX_Tractorista_Tipo_Maquinaria

En la tabla `rrhh.Tractorista` se creó el índice `IX_Tractorista_Tipo_Maquinaria`, definido sobre la columna `tipo_maquinaria`. Este índice agiliza la búsqueda de tractoristas especializados en determinado tipo de equipo, por ejemplo para identificar aquellos empleados que manejan una fumigadora.

```
CREATE NONCLUSTERED INDEX IX_Tractorista_Tipo_Maquinaria
ON rrhh.Tractorista (tipo_maquinaria);
GO
```

IX_Encargado_Ppto_Asignado

En la tabla `rrhh.Encargado` se creó el índice `IX_Encargado_Ppto_Asignado`, definido sobre `ppto_asignado`. Este índice permite generar reportes que clasifican encargados según el presupuesto que administran (ejemplo: identificar aquellos con presupuesto superior a un monto determinado). De esta manera se optimizan consultas de gestión económica relacionadas con los recursos humanos.

```
CREATE NONCLUSTERED INDEX IX_Encargado_Ppto_Asignado
ON rrhh.Encargado (ppto_asignado);
GO
```

IX_Veterinario_Especialidad

En la tabla `rrhh.Veterinario` se creó el índice `IX_Veterinario_Especialidad`, definido sobre `especialidad`. Este índice permite consultar de manera más rápida a los veterinarios según su área de trabajo (por ejemplo: especialistas en reproducción).

```
CREATE NONCLUSTERED INDEX IX_Veterinario_Especialidad
ON rrhh.Veterinario (especialidad);
GO
```

IX_APE_id_evento

En la tabla `gestion.Animal_Personal_Evento` se creó el índice `IX_APE_id_evento`, definido sobre `id_evento`. Este índice agiliza las consultas que buscan todas las personas y animales involucrados en un evento específico (por ejemplo: identificar participantes de una vacunación).

```
CREATE NONCLUSTERED INDEX IX_APE_id_evento
ON gestion.Animal_Personal_Evento(id_evento);
GO
```

IX_PE_id_evento

En la tabla `gestion.Potrero_Evento` se creó el índice `IX_PE_id_evento`, definido sobre `id_evento`. Este índice facilita encontrar todos los potreros asociados a un evento particular (por ejemplo: potreros que participaron en una rotación de pasturas).

```
CREATE NONCLUSTERED INDEX IX_PE_id_evento
ON gestion.Potrero_Evento(id_evento);
GO
```

IX_PE_Potrero_Evento

En la tabla `gestion.Potrero_Evento` se creó el índice `IX_PE_Potrero_Evento`, definido sobre las columnas `id_evento` e `id_potrero`. Este índice facilita encontrar todos los potreros asociados a un evento específico, así como también identificar todos los eventos en los que participó un determinado potrero. Optimiza consultas frecuentes como "ver qué potreros participaron en un evento" o "consultar todos los eventos de un potrero determinado" sin necesidad de escanear toda la tabla.

```
CREATE NONCLUSTERED INDEX IX_PE_Potrero_Evento
ON gestion.Potrero_Evento(id_evento, id_potrero)
GO
```

4.3 Índice de cobertura

IX_Peso_Consulta_Cover

El índice IX_Peso_Consulta_Cover permite buscar todos los registros de peso de un animal específico (`id_animal`) e incluye adicionalmente las columnas `fecha_medicion` y `valor`. De esta forma se optimizan las consultas como "mostrar el historial de pesos de un animal" o "obtener el peso actual de un animal" porque las columnas incluidas pueden obtenerse directamente desde el índice sin necesidad de recorrer toda la tabla.

De esta manera, las consultas frecuentes que requieren estas tres columnas se ejecutan de forma más eficiente, mejorando el rendimiento de los reportes y análisis sobre los datos de peso.

```
CREATE NONCLUSTERED INDEX IX_Peso_Consulta_Cover
ON produccion.Peso(id_animal)
INCLUDE (fecha_medicion, valor);
GO
```

IX_APE_Personal_Evento_Cover

En este caso, el índice IX_APE_personal_evento_Cover permite buscar todos los registros de participación de un personal en un evento específico (`id_personal` y `id_evento`) y obtener directamente las columnas `id_animal` y `rol_evento`, indicando qué animal estuvo involucrado y

cuál fue el rol del personal en ese evento. De esta forma, optimiza consultas frecuentes como "mostrar todos los animales y roles de un trabajador en un evento" o "consultar el historial de participación de un empleado en distintos eventos" sin necesidad de recorrer toda la tabla.

```
CREATE NONCLUSTERED INDEX IX_APE_personal_evento_Cover
ON gestion.Animal_Personal_Evento(id_personal, id_evento)
INCLUDE (id_animal, rol_evento);
GO
```

5. Modificación de Tablas

5.1 Modificación en el Tipo de Dato de una Columna Existente.

En este apartado se realizan modificaciones a tablas existentes con el fin de mejorar la integridad de los datos. En primer lugar, se agregan restricciones `(CHECK)` a la columna `celular` de la tabla `rrhh.Personal` para validar su formato, verificando que la cantidad de dígitos ingresados se encuentre entre 8 y 18 caracteres. Además, se agrega una restricción de unicidad `(UNIQUE)` para evitar duplicados en los números de celular registrados. Esta medida es necesaria porque se requiere contar con dos números de teléfonos diferentes, uno del personal para poder comunicarse directamente y otro de contacto de emergencia, de manera que siempre exista un número válido en caso de situaciones imprevistas.

```
ALTER TABLE rrhh.Personal
ADD CONSTRAINT CHK_CelularFormatoCompleto CHECK (celular NOT LIKE '%[^0-9]%' AND
LEN(celular) BETWEEN 8 AND 18);

ALTER TABLE rrhh.Personal
ADD CONSTRAINT UQ_Personal_Celular UNIQUE (celular);
```

Otro cambio realizado consiste en agregar una restricción a la columna `anios_experiencia`, que no debe admitir valores negativos, garantizando así la coherencia lógica de la información.

```
ALTER TABLE rrhh.capataz
ADD CONSTRAINT CHK_Experiencia CHECK (anios_experiencia >= 0);
```

5.2 Comandos ALTER y DROP

En este apartado se muestra cómo agregar una nueva columna a una tabla y, en caso de ser necesario, eliminarla de forma definitiva. Por ejemplo, se agrega la columna `correo` a la tabla `rrhh.Personal`, permitiendo almacenar direcciones de correo electrónico de hasta 30 caracteres y aceptando valores nulos:

```
ALTER TABLE rrhh.Personal
ADD correo VARCHAR(30) NULL;
```

Posteriormente, si se desea eliminar esta columna de la tabla de manera permanente, se utiliza el comando `DROP COLUMN`:

```
ALTER TABLE rrhh.Personal
DROP COLUMN correo;
```

6. Eliminación de Objetos

6.1 Comando DROP

Antes de eliminar un objeto, se creó explícitamente el índice `IX_Peso_fecha_valor` sobre las columnas `fecha_medicion` y `valor` de la tabla `produccion.Peso` con la siguiente sentencia:

```
CREATE NONCLUSTERED INDEX IX_Peso_fecha_valor
ON produccion.Peso(fecha_medicion, valor);
GO
```

Este índice se creó con la intención de luego demostrar cómo eliminar un objeto de la base de datos. La eliminación se realiza con el comando `DROP INDEX`:

```
DROP INDEX IX_Peso_fecha_valor ON produccion.Peso;
```

6.2 Comando TRUNCATE

El comando `TRUNCATE` permite eliminar todos los registros de la tabla `produccion.Peso`, manteniendo la estructura de la tabla y sus índices asociados.

```
TRUNCATE TABLE produccion.Peso;
```

7. Estructuras Temporales

7.1 Tabla Temporal Local

A continuación, se crea una tabla temporal local `#Pesajes_Hoy`, que solo se visualiza en la sesión actual. Dicha tabla almacena los pesajes de animales realizados ese mismo día, permitiendo optimizar las consultas, ya que solo se trabaja con los datos del día.

La estructura de la tabla temporal `#Pesajes_Hoy` es la siguiente:

- `id_animal`: entero (`INT`), identifica al animal.
- `peso`: decimal con 5 dígitos totales y 2 decimales (`DECIMAL(5,2)`), representa el peso del animal.

```
CREATE TABLE #Pesajes_Hoy (
    id_animal INT,
    peso DECIMAL(5,2)
);
```

Los datos se cargan con un `INSERT INTO... SELECT` desde la tabla `produccion.Peso`, filtrando la fecha de medición con la fecha actual (`GETDATE()`).

```
INSERT INTO #Pesajes_Hoy
SELECT id_animal, valor
FROM produccion.Peso
WHERE fecha_medicion = CAST(GETDATE() AS DATE);
```

Consulta para control:

```
SELECT * FROM #Pesajes_Hoy;
```

7.2 Tabla Temporal Global

Se crea una tabla temporal global que se elimina automáticamente cuando la última sesión que la utiliza se cierra. En ella se almacenan los peones disponibles en el turno de la mañana, lo que resulta útil en un entorno colaborativo, evitando consultas repetitivas.

La estructura de la tabla temporal global `##Personal_Disponible_Peon` es la siguiente:

- `id_personal`: entero (`INT`), identifica al empleado.
- `nombre`: cadena de hasta 50 caracteres (`VARCHAR(50)`), obligatorio.
- `apellido`: cadena de hasta 50 caracteres (`VARCHAR(50)`), obligatorio.
- `turno`: cadena de hasta 20 caracteres (`VARCHAR(20)`), indica el turno de trabajo.

```
CREATE TABLE ##Personal_Disponible_Peon (
    id_personal INT,
    nombre VARCHAR(50),
    apellido VARCHAR(50),
    turno VARCHAR(20)
);
```

Se cargan los datos mediante un `INSERT INTO ... SELECT` que vincula las tablas `rrhh.Personal` y `rrhh.Peon`.

```
INSERT INTO ##Personal_Disponible_Peon
SELECT p.id_personal, p.nombre, p.apellido, pe.turno
FROM rrhh.Personal p
JOIN rrhh.Peon pe ON p.id_personal = pe.id_personal
WHERE pe.turno = 'Mañana';
```

Consulta para control:

```
SELECT * FROM ##Personal_Disponible_Peon;
```

7.3 Variable de tabla

En este ejemplo se utiliza una variable de tabla `@CapatacesSenior` para almacenar de forma transitoria la información de los capataces con más de 10 años de experiencia. Esta variable permite trabajar con los datos de manera temporal dentro de la sesión actual, sin afectar la estructura de la tabla original.

La estructura de la variable de tabla `@CapatacesSenior` es la siguiente:

- `id_capataz`: entero (`INT`), identifica al capataz.
- `nombre`: cadena de hasta 50 caracteres (`VARCHAR(50)`), obligatorio.
- `apellido`: cadena de hasta 50 caracteres (`VARCHAR(50)`), obligatorio.
- `anios_experiencia`: entero (`INT`), representa la cantidad de años de experiencia.

```
DECLARE @CapatacesSenior TABLE (
    id_capataz INT,
    nombre VARCHAR(50),
    apellido VARCHAR(50),
    anios_experiencia INT
);
```

Al igual que la tabla temporal local y global, los datos se cargan con un `INSERT INTO ... SELECT` que obtiene los datos desde `rrhh.Capataz` y `rrhh.Personal`.

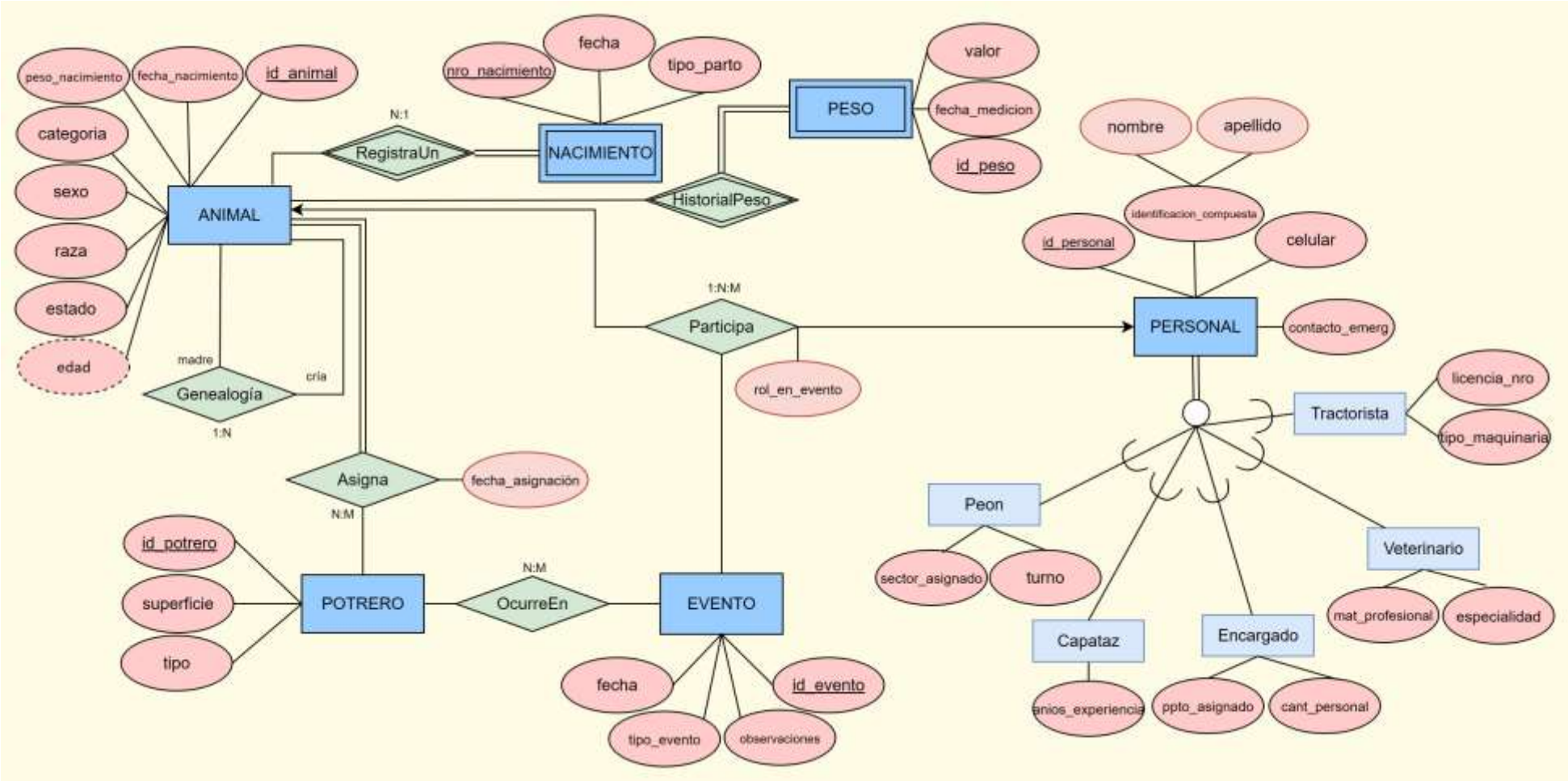
```
INSERT INTO @CapatacesSenior
SELECT c.id_personal, p.nombre, p.apellido, c.anios_experiencia
FROM rrhh.Capataz c
JOIN rrhh.Personal p ON c.id_personal = p.id_personal
WHERE c.anios_experiencia >= 10;
```

Consulta para control:

```
SELECT * FROM @CapatacesSenior;
```

ANEXOS

Diagrama DER



Modelo entidad–relación