Universidad Nacional Autónoma de México
Facultad de Ciencias

Reporte Torneo Pokemon

Equipo: Borregos

Fecha: 4 de diciembre de 2025

Profesor: Gerardo Avilés Rosas
Ayudante: Luis Enrique García Gómez
Ayudante: Jaime Octavio Delfín López
Ayudante: Ricardo Badillo Macías

Integrantes

Aaron López Mendoza	321019349
Rodrigo Zaldivar Alanis	424029605
Isaac Giovani Escobar González	321336400
Kevin Jonathan Garduño Escobar	321070629
Zapata Amezcua Santiago	321251796

Analisis de Requerimientos

Enumeración de Requerimientos Candidato

- **Requerimiento 01:** Registro de personal.
- **Requerimiento 02:** Registro de participantes y visitantes.
- **Requerimiento 03:** Control de torneos.
- **Requerimiento 04:** Inscripción al torneo.
- **Requerimiento 05:** Inventario y venta de alimentos.
- **Requerimiento 06:** Registro de ingresos.
- **Requerimiento 07:** Monitoreo del personal.
- **Requerimiento 08:** Control de ediciones.

Comprensión del Contexto del Sistema:

En el problema planteado, el sistema gestiona un entorno complejo donde interactúan diversos actores. Los encargados de registro dan de alta a los participantes y espectadores, además de inscribir a los participantes a los torneos.

Si un participante decide entrar al torneo de peleas, debe registrar hasta 6 pokémones para este fin. Durante los torneos, los monitores se encargan de registrar el progreso de los participantes. Por ejemplo:

- **Torneo de peleas:** Deciden el ganador de la pelea.
- **Torneo de shinies:** Agregan los shinies atrapados por los participantes.
- **Torneo de distancia recorrida:** Registran el metraje de los participantes en los distintos puntos.

Asimismo, los vendedores comercializan alimentos a participantes y espectadores (considerando que otros miembros del personal también pueden ser participantes) y les dan a conocer el precio de los alimentos con IVA. Existe también un encargado de administrar la base de datos que da de alta a los trabajadores con su información, salario y horarios.

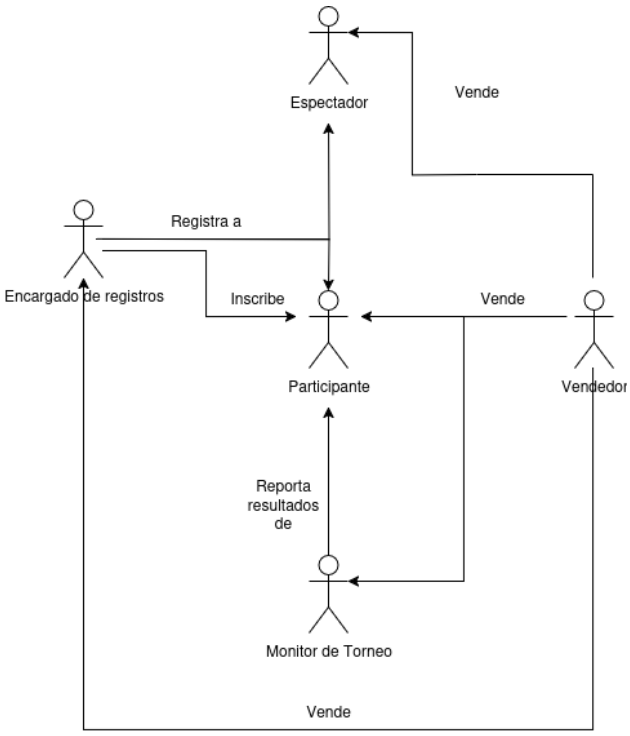

Figura 1: Diagrama que ilustra el sistema

Requerimientos Funcionales

Gestión de Participantes:

El sistema deberá cumplir con las siguientes funciones respecto a los participantes:

- Permitir registrar a los participantes con: nombre completo, fecha de nacimiento, sexo, números de teléfono y correos (cantidad variable), número de cuenta de la UNAM, facultad y carrera.
- Capturar la información de una o varias cuentas de *Pokémon Go* asociadas al participante. Los datos a capturar son: código de entrenador, nombre de usuario, nivel de entrenador y equipo.

Gestión de Torneos:

El sistema gestionará las inscripciones y la lógica de los distintos torneos:

- Inscribir participantes con la restricción de que aquellos en el *torneo de peleas* no podrán participar en ningún otro.
- Permitir a los encargados de registro realizar cambios en las inscripciones (ej. priorizar otros torneos sobre el de peleas) solamente durante el horario de registros.

Torneo de Peleas:

- Capturar información de los pokémones a usar (Mínimo 1, Máximo 6).
- Guardar información del pokémon: nombre/apodo, especie, tipo, puntos de combate, peso, sexo y si es shiny.
- Llevar registro de los enfrentamientos y del ganador.

Torneo de Recorrido (Distancia):

- Validar inicio del recorrido en la entrada de C.U., capturando hora de inicio y distancia actual de la app.
- Validar que se haya realizado al menos un chequeo en puntos designados (Rectoría y/o Universum).
- El registro final puede hacerse en cualquiera de los tres puntos, capturando la hora.

Torneo de Captura de Shinies:

- Registrar al shiny capturado (fecha y hora) y validar que la captura ocurrió durante el torneo.

Premiación:

- Determinar y mostrar al ganador de cada torneo con base en los criterios registrados por los monitores.

- Registrar y asignar recompensas (5000 pesos, trofeo y mercancía).

Gestión de Personal y Ventas:

- **Datos del Personal:** Guardar nombre completo, fecha de nacimiento, edad, sexo, teléfonos, correos, domicilio, rol y paga.
- **Inventario:** Capturar información de alimentos (tipo, nombre, perecedero/caducidad, precio con y sin IVA).
- **Ventas:**
  - Calcular precio total y generar orden de compra por venta.
  - Registrar qué vendedor realizó la venta.
  - Calcular el 25 % del total de ventas de cada vendedor para agregarlo a su pago.
- **Logística:**
  - Validar ubicación del puesto de cada vendedor.
  - Almacenar horario (9am-3pm o 3pm-9pm) y ubicación de limpiadores y cuidadores.
- **Incentivos y Validaciones:**
  - Incrementar pago del personal de registro en 50 pesos por cada registro realizado.
  - Validar que los registros de participantes sean entre las 7:00 y 9:00.

Requerimientos Misceláneos:

- **Espectadores:** Registrar nombre completo, fecha de nacimiento, género, hora de ingreso y salida.
- **Finanzas:** Registrar transacciones (bancarias/efectivo), medio de pago, llevar balance y obtener ganancias totales.
- **Ediciones:** Asociar toda la información a la edición correspondiente del torneo y su fecha.
- **Cálculos:** Calcular y mostrar la edad de las personas.
- **Vistas y Seguridad:**
  - Contar con vistas especializadas según el rol (registro, monitores de pelea, chequeos de distancia, registro de shinies, venta de alimentos).
  - Permitir creación de cuentas para personal con autenticación requerida para acceder a las vistas.

Requerimientos No Funcionales

Asociados a Requerimientos Funcionales:

- **Concurrencia:** El sistema deberá mantener y despachar múltiples conexiones y manejar adecuadamente operaciones simultáneas.
- **Seguridad:** Implementación de permisos estrictos. Ejemplo: Un encargado de registro no debe poder manipular sus ventas ni modificar ganadores.
- **Consistencia:** Garantizar consistencia de datos y reflejar cambios en tiempo real o con latencia mínima.

Restricciones del Proyecto (No asociados a Funcionalidad)

- **Fecha de Entrega:** El producto final debe entregarse antes del 3 de Diciembre del 2025 (Semestre 2026-1).
- **Entregas Parciales:** Entrega semanal de avances al líder administrativo de *Solrock Battle Association*.
- **Disponibilidad:** El sistema debe garantizar alta disponibilidad durante el evento.

Modelo Conceptual (E/R)

Consideraciones de diseño

Para la realización del Modelo Entidad Relación Extendido del Caso de Uso: Torneo Pokémon Go, se hizo un análisis completo con el objetivo de identificar las entidades, atributos y relaciones que formarán parte del sistema a modelar. Como primera parte, identificamos las entidades con sus respectivos atributos:

- **Persona**
  Atributos: idPersona, nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, sexo, teléfono, correo.

- **Participante (Hereda de Persona)**
  Atributos: nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, sexo, teléfono(s), correo(s) electrónico(s), número de cuenta, facultad, carrera.

- **Empleado (Hereda de Persona)**
  Atributos: idEmpleado, direccion (ciudad, calle, colonia, c.p., numero interior, numero exterior) nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, sexo, teléfono(s), correo(s) electrónico(s).

- **Cuidador (Hereda de Empleado)**
  Atributos: nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, sexo, teléfono(s), correo(s) electrónico(s), horario, localización, salario.

- **Limpiador (Hereda de Empleado)**
  Atributos: nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, sexo, teléfono(s), correo(s) electrónico(s), localización, salario, horario.

- **Encargado (Hereda de Empleado)**
  Atributos: nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, sexo, teléfono(s), correo(s) electrónico(s), pago, registro.

- **Vendedor (Hereda de Empleado)**
  Atributos: nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, sexo, teléfono(s), correo(s) electrónico(s), ventas, ubicación, pago.

- **Espectador**
  Atributos: idEspectador, nombre completo (nombre, apellido paterno, apellido materno), fecha de nacimiento, edad, hora de ingreso, hora de salida, género.
- **Pokémon**
  Atributos: IdPokemon, nombre, especie, tipo, peso, sexo, shiny, Combat Points.
- **Cuenta**
  Atributos: codigoEntrenador, nombreUsuario, nivel, equipo.
- **Venta**
  Atributos: idVenta, fecha/hora, total con IVA, total sin IVA, TipoPago.
- **MétodoPago**
  Atributos: idMetodo, tipo de pago.
- **Alimento**
  Atributos: idAlimento, nombre, precio sin IVA, precio con IVA, fecha de caducidad, perecedero.
- **Torneo**
  Atributos: idTorneo, periodo(inicio, final).
- **EdicionTorneo**
  Atributos: idEdicion, número de edición, fecha del evento, nota.
- **Pelea**
  Atributos: idPelea.

Herencia / Especialización

Dado que se trata de un Modelo Entidad Relación Extendido, se hizo uso de herencia o especialización. En el diseño, se considero que una Persona puede especializarse en distintos roles, como Participante o Empleado. A su vez, los Empleados se dividen en subtipos como Cuidadores, Limpiadores, Vendedores y Encargados, cada uno con atributos específicos que dependen de su función dentro del evento; esto nos permitió reducir la redundancia de atributos y representar de manera más clara las diferencias entre los distintos tipos de personas. Decidimos que la entidad Espectador no hereda directamente de Persona, manteniéndola como una entidad separada, para asegurar que la superentidad Persona se mantuviera cohesiva y robusta. Sí Espectador se incluyera como una subentidad, los atributos de Persona tendrían que limitarse únicamente a aquellos que son comunes a Espectador, Participante, y Empleado. Este conjunto compartido es mucho más reducido, lo que obligaría a eliminar atributos relevantes como teléfono(s) y correo(s) electrónico(s) de la superentidad Persona , diluyendo el beneficio de la herencia para las subentidades Participante y Empleado, los cuales sí comparten un conjunto más amplio de características. También en la parte de los torneos se decidió usar la herencia puesto que aunque todos forman parte de la misma categoría general, cada tipo de torneo tiene particularidades que lo distinguen de los demás. Por esta razón, se definió una entidad genérica llamada Torneo como supertipo, y a partir de ella se especializaron tres subtipos: Torneo de Peleas, Torneo de Distancia Recorrida y Torneo de Captura de Shinys.

Relaciones (Cardinalidad y Participación)

Relación: Comprar

- Entidades: Persona, Venta
- Cardinalidad: 1:N (Uno a Muchos). Una persona puede comprar varios alimentos (se le pueden hacer muchas ventas), pero una venta solo es realizada a una persona.
- Participación:
  Persona: Parcial. No todas las personas están obligadas a comprar.
  Venta: Total. Las ventas necesitan de un comprador.

Relación: Guardar

- Entidades: Cuenta, Pokemon
- Cardinalidad: 1:N (Uno a Muchos). Una cuenta puede tener muchos Pokémon, pero un Pokémon pertenece a una sola cuenta.
- Participación:
  Cuenta: Parcial. Una cuenta puede no tener Pokémon.
  Pokemon: Total. Todo Pokémon debe pertenecer a una cuenta.

Relación: Vender

- Entidades: Vendedor, Venta
- Cardinalidad: 1:N (Uno a Muchos) Un vendedor puede hacer muchas ventas y una venta solo puede ser hecha por un vendedor.
- Participación:
  Vendedor: Parcial. Puede ser que un vendedor no venda nada en todo el torneo.
  Venta: Total, toda venta debió haber sido hecha por un vendedor.

Relación: Registrar

- Entidades: Alimento, Venta
- Cardinalidad: N:M (Muchos a Muchos) Muchos alimentos pueden ser registrados en muchas ventas y muchas ventas pueden tener muchos alimentos.
- Participación:
  Alimento: Parcial. Un alimento puede no ser registrado en una venta
  Venta: Total. Una venta no puede existir si no hay un registro de al menos un alimento/producto.

Relación: Tener

- Entidades: Participante, Cuenta
- Cardinalidad: 1:N (1 a Muchos) Un participante puede tener muchas cuentas, pero una cuenta debe de pertenecer solo a un participante.
- Participación:
  Participante: Total. Cada participante debe tener al menos una cuenta pokémon asociada desde el principio.
  Cuenta: Total. Una cuenta pokémon debe estar asociada si o si a un participante para existir.

Relación: Capturar

- Entidades: Pokemon, Participante, Captura Shiny
- Cardinalidad: 1:N:M (1 a Muchos a Muchos). Un participante puede capturar muchos pokemones en un torneo, un pokémon solo puede ser capturado por un participante una vez en el torneo, y en un torneo se pueden llevar a cabo diversas capturas realizadas por distintos participantes.
- Participación:
  Pokemon: Parcial. No todos los Pokémon son capturados por un participante.
  Participante: Parcial. Puede ser que un participante no captura un pokémon.
  Atributos: Fecha (Día, Mes), Hora

Relación: Recorrer

- Entidades: Participante, Recorrido
- Cardinalidad: N:1 (Muchos a Uno). En un torneo recorrido, muchos participantes hacen un recorrido y un participante hace solo un recorrido en el torneo.
- Participación:
  Participante: Parcial. No todos los participantes hacen recorridos
  Recorrido: Parcial. Pueden haber torneos en el que ningún participante haga recorridos.

Relación: Participar

- Entidades: Participante, Torneo
- Cardinalidad: N:M (Muchos a Muchos). Un participante puede participar en varios torneos y en un torneo participan muchos participantes.
- Participación:
  Participante: Parcial. No todos los participantes participan en torneos.
  Torneo: Parcial. Puede haber torneos sin participantes.

Relación: Ganar

- Entidades: Participante, Torneo
- Cardinalidad: N:M (Muchos a Muchos). Un participante puede ganar varios torneos y un torneo puede ser ganado por varios participantes (en diferentes categorías, por ejemplo).
- Participación:
  Participante: Parcial. No todos los Participantes ganan torneos.
  Torneo: Parcial. No todos los torneos tienen un ganador registrado (podrían estar en curso).

Relación: Vencer

- Entidades: Participante, Pelea
- Cardinalidad: 1:1 (1 a 1). Un participante puede ser el vencedor de una pelea, y una pelea solo puede tener a un participante como vencedor.
- Participación:
  Participante: Parcial. Un participante puede o no ser el vencedor en una pelea, no es obligatorio que gane.
  Pelea: Total. En una pelea siempre deberá de tener asociado un participante ganador al final de cada pelea, no existen los empates.

Relación: Suceder

- Entidades: Torneo, EdicionTorneo
- Cardinalidad: 1:N (Uno a Muchos). Un torneo puede tener múltiples ediciones, pero cada edición pertenece a un único tipo de torneo.
- Participación:
  Torneo: Parcial. Puede existir un tipo de torneo del cual aún no se ha realizado ninguna edición.
  EdicionTorneo: Total. Toda edición de torneo debe estar asociada a un tipo de torneo.

Relación: Ocurrir

- Entidades: Pelea, Enfrentamiento
- Cardinalidad: N:1 (Muchos a Uno). Pueden ocurrir muchas peleas en un torneo, pero cada pelea pertenece a un solo torneo.
- Participación:
  Pelea: Total. Toda pelea debe ocurrir dentro de un torneo.
  Enfrentamiento: Parcial. Un torneo puede no tener peleas registradas aún.

Relación: Pelear

- Entidades: Participante, Pelea
- Cardinalidad: 1:M
- Participación:
  Participante: Parcial.
  Pelea: Total.

Relación: Trabajar

- Entidades: Empleado, EdicionTorneo
- Cardinalidad: N:M (Muchos a muchos). Un empleado puede trabajar en varias ediciones de torneo y una edición de torneo tiene varios empleados.
- Participación:
  Empleado: Total.
  EdicionTorneo: Total (Asumimos que en el torneo siempre habrán empleados que trabajen).

Relación: Equipar

- Entidades: Participante, Pokemon, Enfrentamiento
- Cardinalidad: N:M (Muchos a muchos).
- Participación:
  Participante: Parcial.
  Enfrentamiento: Parcial.
  Pokemon: Parcial.

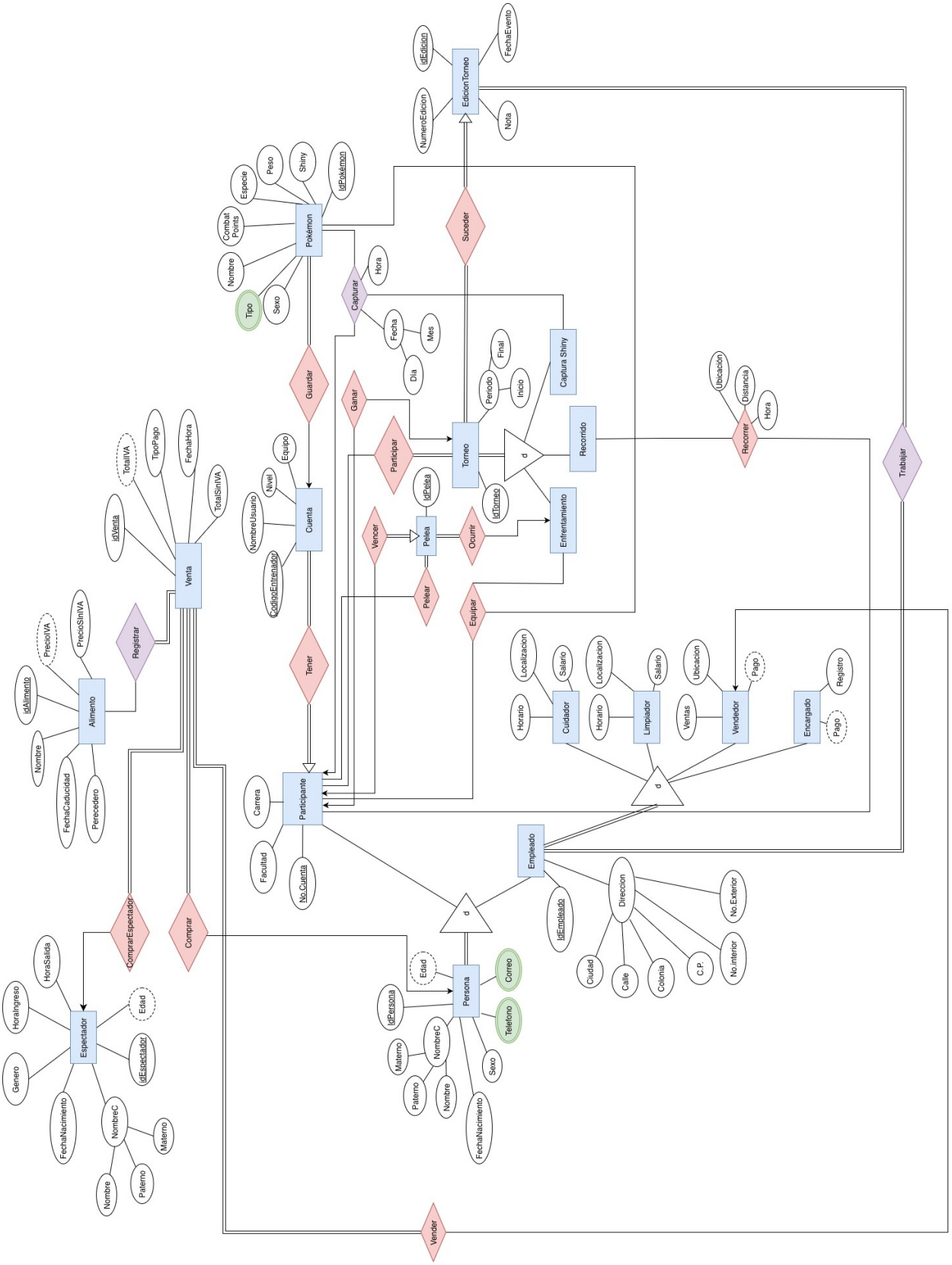

Figura 2: Diagrama del Modelo E/R (EntidadRelacion)

Modelo Logico (Relacional)

Atributos (Dominio, Restricciones y Tipo de llave)

Usuarios:

Persona

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona:	bigint	NOT NULL, UNIQUE	PK
Nombre	varchar	NOT NULL	-
Paterno	varchar	NOT NULL	-
Materno:	varchar	NOT NULL	-
FechaNacimiento:	date	NOT NULL, CHECK (fechaNacimiento ≤ CURRENT_DATE)	-
Sexo	varchar(6)	NOT NULL	-

Participante

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
IdTorneo	bigint	NOT NULL	FK
No.Cuenta	varbit[9]	NOT NULL, UNIQUE	-
Facultad	varchar	NOT NULL	-
Carrera	varchar	NOT NULL	-
Ubicacion	numeric	-	-
Distancia	numeric	-	-
Hora:	timetz	-	-

Empleado

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
IdEmpleado	varchar	NOT NULL	-
Ciudad	varchar	NOT NULL	-
Calle	varchar	NOT NULL	-
Colonia	varchar	NOT NULL	-
C.P.	varbit[8]	NOT NULL	-
No.Interior	int	NOT NULL	-
No.Exterior	int	NOT NULL	-

Cuidador

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
Horario	timetz	NOT NULL	-
Localizacion	text	NOT NULL	-
Salario	money	NOT NULL	-

Limpiador

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
Horario	timetz	NOT NULL	-
Localizacion	text	NOT NULL	-
Salario	money	NOT NULL	-

Vendedor

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
Ventas	int	-	-
Ubicacion	text	NOT NULL	-

Encargado

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
Registro	varchar(8)	NOT NULL	-

Espectador

Atributo	Dominio	Restricciones	Tipo de llave
IdEspectador	bigint	NOT NULL, UNIQUE	PK
Nombre:	varchar	NOT NULL	-
Paterno:	varchar	NOT NULL	-
Materno:	varchar	NOT NULL	-
FechaNacimiento:	date	NOT NULL, CHECK (fechaNacimiento ≤ CURRENT_DATE)	-
Genero:	varchar(6)	NOT NULL	-
HoraIngreso:	timetz	NOT NULL	-
HoraSalida:	timetz	NOT NULL	-

Servicios

Teléfono

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
Telefono	int	NOT NULL	PK

Correo

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	PK
Correo	varchar	NOT NULL	PK

Alimento

Atributo	Dominio	Restricciones	Tipo de llave
IdAlimento	bigint	NOT NULL, UNIQUE	PK
Nombre	varchar	NOT NULL	-
FechaCaducidad	date	NOT NULL	-
Perecedero	bool	NOT NULL	-
PrecioSinIVA	money	NOT NULL	-

Venta

Atributo	Dominio	Restricciones	Tipo de llave
IdVenta	bigint	NOT NULL, UNIQUE	PK
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdEspectador	bigint	NOT NULL, UNIQUE	FK
TipoPago	varchar	NOT NULL	-
FechaHora	timestamptz	NOT NULL	-
TotalSinIva	money	NOT NULL	-

Registrar

Atributo	Dominio	Restricciones	Tipo de llave
IdAlimento	bigint	NOT NULL, UNIQUE	FK
IdVenta	bigint	NOT NULL, UNIQUE	FK

Trabajar

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdEdicion	bigint	NOT NULL	FK

Cuenta y Pokémon

Cuenta

Atributo	Dominio	Restricciones	Tipo de llave
CodigoEntrenador	char(5)	NOT NULL, UNIQUE	PK
IdPersona	bigint	NOT NULL, UNIQUE	FK
NombreUsuario	varchar	NOT NULL, UNIQUE	-
Nivel	varchar	NOT NULL	-
Equipo	varchar	NOT NULL	-

Pokemon

Atributo	Dominio	Restricciones	Tipo de llave
IdPokemon	int	NOT NULL, UNIQUE	PK
CodigoEntrenador	char(5)	NOT NULL, UNIQUE	FK
Nombre	varchar	NOT NULL	-
CombatPoints	int	-	-
Especie	varchar	NOT NULL	-
Peso	bit[3]	NOT NULL	-
Sexo	varchar(6)	NOT NULL	-
Shiny	bool	NOT NULL	-

Torneo y demás entidades

Torneo

Atributo	Dominio	Restricciones	Tipo de llave
IdTorneo	bigint	NOT NULL	PK
IdTorneo	bigint	NOT NULL	FK
Inicio	timetz	NOT NULL	-
Final	timetz	NOT NULL, CHECK (Inicio<Final)	-

Enfrentamiento

Atributo	Dominio	Restricciones	Tipo de llave
IdTorneo	bigint	NOT NULL	PK

Tipo

Atributo	Dominio	Restricciones	Tipo de llave
IdPokémon	int	NOT NULL, UNIQUE	PK
Tipo	varchar	NOT NULL	PK

Recorrido

Atributo	Dominio	Restricciones	Tipo de llave
IdTorneo	bigint	NOT NULL	PK

Pelea

Atributo	Dominio	Restricciones	Tipo de llave
IdPelea	int	NOT NULL, UNIQUE	PK
IdParticipante	bigint	NOT NULL, UNIQUE	FK
IdTorneo	bigint	NOT NULL	FK

CapturaShiny

Atributo	Dominio	Restricciones	Tipo de llave
IdTorneo	bigint	NOT NULL	PK

Edición Torneo

Atributo	Dominio	Restricciones	Tipo de llave
IdEdicion	bigint	NOT NULL, UNIQUE	PK
NumeroEdicion	int	NOT NULL, UNIQUE	-
FechaEvento	date	NOT NULL	-
Nota	varchar	NOT NULL	-

Participar

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdTorneo	bigint	NOT NULL	FK

Pelear

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdPelea	int	NOT NULL, UNIQUE	FK

Capturar

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdPokemon	int	NOT NULL, UNIQUE	FK
IdTorneo	bigint	NOT NULL	FK
Fecha y Hora	timestamptz	NOT NULL	-

Ganar

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdTorneo	bigint	NOT NULL	FK

Equipar

Atributo	Dominio	Restricciones	Tipo de llave
IdPersona	bigint	NOT NULL, UNIQUE	FK
IdTorneo	bigint	NOT NULL	FK
IdPokemon	integer	NOT NULL, UNIQUE	FK

Preferencias de diseño en el diagrama relacional:

- Pese a lo estipulado en las normas de traduccion de la especialización con completez y disyunción, que sugiere eliminar la relación padre y dejar solo a las tablas hijas, decidimos mantener en este caso a la relación Persona y la relación Torneo para evitar que se generaran más relaciones y tener un diseño un poco más simple pero funcional. La completez y disyunción se manejarán en la implementación de la base de datos con disparadores y restricciones.
- A todas la subentidades que heredan directa o indirectamente de Persona en el diagrama ER, tendrán como llave primaria la llave sustituta IdPersona que heredan de persona. Esto, para simplificar el diseño. Aunque no sean llaves primarias, en la implementación se asegurará la unicidad de llaves candidatas como No.Cuenta y IdEmpleado.
- Ayudándonos del punto anterior, se usara la llave primaria IdPersona para permitir que una tupla perteneciente a persona, sea referenciada en las relaciones Participante y Empleado (esto es un requerimiento). Y de esta manera evitar inconsistencias. El diagrama no expresa una restricción directa acerca de que empleados como Cuidadores, Vendedores o Limpiadores no sean Participantes. Esto se manejará en la implementación.
- Como el número de cuenta de la UNAM son todos números, se optó por un bigint para el dato.
- Dado que desconocemos cómo están formados los identificadores de empleado, decidimos que es un atributo de tipo varchar por si aparecen caracteres no numerales arábigos.
- Para representar las distancias de los participantes, estas estarán dadas en metros.
- En el diagrama entidad-relación, se decidio que Espectador dejara de heredar de Persona, ya que no comparte los atributos Telefono y Correo.
- Se evitaron el uso de acentos.
- Utilizamos líneas uno a uno para especificar que las llaves primarias de las relaciones hijas son las mismas que las de las relaciones padre.

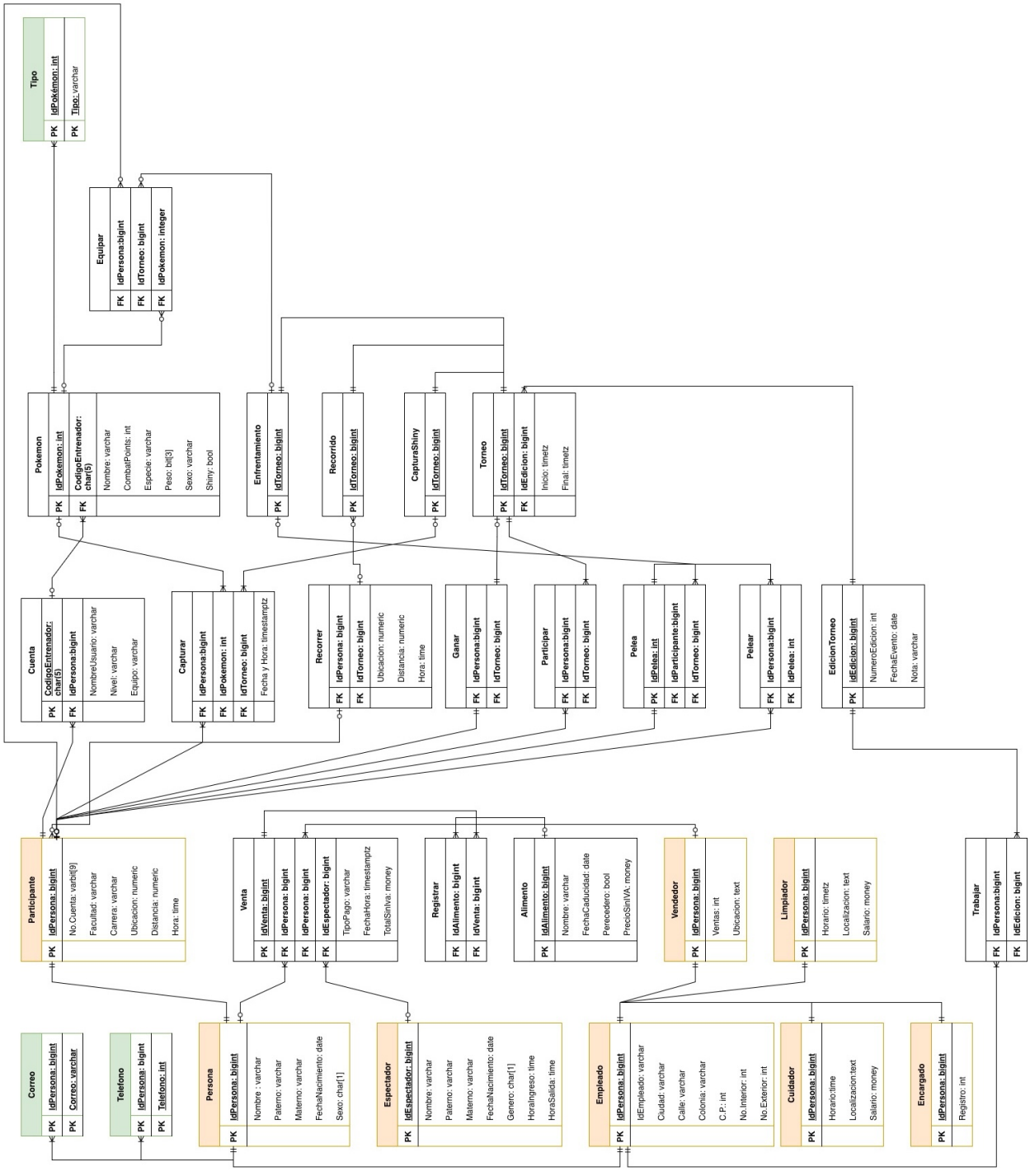

Figura 3: Diagrama del Modelo Relacional

SQL

DDL (Data Definition Language)

Se creo el modelo fisico para las 29 tablas que se desarrollaraon en el modelo relacional. A continuación, se especificarán las restricciones de dominio definidas para cada columna de cada tabla, tal y como se implementaron en nuestro archivo `DDL.sql`.

Persona

```sql
ALTER TABLE Persona ADD CONSTRAINT persona_d1 CHECK(nombre <> '');
ALTER TABLE Persona ADD CONSTRAINT persona_d2 CHECK(paterno <> '');
ALTER TABLE Persona ADD CONSTRAINT persona_d3 CHECK(materno <> '');
ALTER TABLE Persona ALTER COLUMN FechaNacimiento NOT NULL;
ALTER TABLE Persona ADD CONSTRAINT persona_d4 CHECK(sexo = 'M' OR sexo = '
    F');

-- Entidad
ALTER TABLE Persona ADD CONSTRAINT persona_pkey PRIMARY KEY (IdPersona);
```

Espectador

```sql
ALTER TABLE Espectador ADD CONSTRAINT espectador_d1 CHECK(nombre <> '');
ALTER TABLE Espectador ALTER COLUMN Nombre NOT NULL;
ALTER TABLE Espectador ADD CONSTRAINT espectador_d2 CHECK(paterno <> '');
ALTER TABLE Espectador ALTER COLUMN P NOT NULL; -- Nota: Posible error de
    OCR en origen (Paterno)
ALTER TABLE Espectador ADD CONSTRAINT espectador_d3 CHECK(materno <> '');
ALTER TABLE Espectador ALTER COLUMN FechaNaciemiento NOT NULL;
ALTER TABLE Espectador ADD CONSTRAINT espectador_d4 CHECK(genero = 'M' OR
    genero = 'F');
ALTER TABLE Espectador ADD CONSTRAINT espectador_d5 CHECK(Horalngreso
    BETWEEN '09:00:00' AND '19:00:00');
ALTER TABLE Espectador ADD CONSTRAINT espectador_d6 CHECK(HoraSalida
    BETWEEN Horalngreso AND '21:00:00');

-- Entidad
ALTER TABLE Espectador ADD CONSTRAINT espectador_pkey PRIMARY KEY (
    IdEspectador);
```

Empleado

```
ALTER TABLE Empleado ADD CONSTRAINT empleado_d1 CHECK(IdEmpleado <> '');
ALTER TABLE Empleado ALTER COLUMN IdEmpleado SET NOT NULL;
ALTER TABLE Empleado ADD CONSTRAINT empleado_id_unique UNIQUE (IdEmpleado);
ALTER TABLE Empleado ADD CONSTRAINT empleado_d2 CHECK(Ciudad <> '');
ALTER TABLE Empleado ALTER COLUMN Ciudad SET NOT NULL;
ALTER TABLE Empleado ADD CONSTRAINT empleado_d3 CHECK(Calle <> '');
ALTER TABLE Empleado ALTER COLUMN Calle SET NOT NULL;
ALTER TABLE Empleado ADD CONSTRAINT empleado_d4 CHECK(Colonia <> '');
ALTER TABLE Empleado ALTER COLUMN Colonia SET NOT NULL;
ALTER TABLE Empleado ADD CONSTRAINT empleado_d5 CHECK(CP BETWEEN 10000 AND 99999);

-- Entidad
ALTER TABLE Empleado ADD CONSTRAINT empleado_pkey PRIMARY KEY (IdPersona);

-- Referencial
ALTER TABLE Empleado ADD CONSTRAINT empleado_fkey FOREIGN KEY (IdPersona) REFERENCES Persona (IdPersona);
```

Cuidador

```
-- Dominio
ALTER TABLE Cuidador ALTER COLUMN Horario SET NOT NULL;
ALTER TABLE Cuidador ADD CONSTRAINT cuidador_d1 CHECK(Horario BETWEEN '09:00:00' AND '15:00:00');
ALTER TABLE Cuidador ADD CONSTRAINT cuidador_d2 CHECK (Localizacion <> '');
ALTER TABLE Cuidador ALTER COLUMN Localizacion SET NOT NULL;
ALTER TABLE Cuidador ALTER COLUMN Salario SET NOT NULL;

-- Entidad
ALTER TABLE Cuidador ADD CONSTRAINT cuidador_pkey PRIMARY KEY (IdPersona);

-- Referencial
ALTER TABLE Cuidador ADD CONSTRAINT cuidador_fkey FOREIGN KEY (IdPersona) REFERENCES Empleado (IdPersona);
```

Tipo

```
ALTER TABLE Tipo ALTER COLUMN IdTipo ADD GENERATED BY DEFAULT AS IDENTITY;
ALTER TABLE Tipo ALTER COLUMN Tipo SET NOT NULL;
ALTER TABLE Tipo ADD CONSTRAINT tipo_tipo_chk CHECK (char_length(Tipo) > 0);

-- Integridad de entidad (PK)
ALTER TABLE Tipo ADD CONSTRAINT tipo_pk PRIMARY KEY (IdTipo);
```

Telefono

```
-- Dominio
ALTER TABLE Telefono ALTER COLUMN IdPersona NOT NULL;
ALTER TABLE Telefono ALTER COLUMN Telefono NOT NULL;

-- Referencial
ALTER TABLE Telefono ADD CONSTRAINT telefono_fkey FOREIGN KEY (IdPersona)
    REFERENCES Persona (IdPersona);
```

Correo

```
-- Dominio
ALTER TABLE Correo ALTER COLUMN IdPersona NOT NULL;
ALTER TABLE Correo ADD CONSTRAINT correo_d1 CHECK (correo LIKE '%_@_%._%'
    );
ALTER TABLE Correo ALTER COLUMN Correo NOT NULL;

-- Referencial
ALTER TABLE Correo ADD CONSTRAINT correo_fkey FOREIGN KEY (IdPersona)
    REFERENCES Persona (IdPersona);
```

Participante

```
-- Dominio
ALTER TABLE Participante ADD CONSTRAINT participante_d1 CHECK (NoCuenta >
    0);
ALTER TABLE Participante ADD CONSTRAINT participante_d2 CHECK (Facultad <>
     '');
ALTER TABLE Participante ADD CONSTRAINT participante_d3 CHECK (Carrera <>
    '');
ALTER TABLE Participante ADD CONSTRAINT participante_d4 CHECK (Distancia
    >= 0);
ALTER TABLE Participante ADD CONSTRAINT participante_d5 CHECK (Hora IS
    NULL OR Hora::TEXT <> '');

-- Entidad
ALTER TABLE Participante ADD CONSTRAINT participante_pkey PRIMARY KEY (
    IdPersona);

-- Referencias
ALTER TABLE Participante ADD CONSTRAINT participante_fkey_torneo FOREIGN
    KEY (IdTorneo) REFERENCES Torneo (IdTorneo);
```

Limpiador

```
-- Dominio
ALTER TABLE Limpiador ALTER COLUMN Horario SET NOT NULL;
```

```
ALTER TABLE Limpiador ADD CONSTRAINT limpiador_d1 CHECK(Horario IN ('09:00:00'::TIME, '15:00:00'::TIME));
ALTER TABLE Limpiador ADD CONSTRAINT limpiador_d2 CHECK(Localizacion <> '');
ALTER TABLE Limpiador ALTER COLUMN Localizacion SET NOT NULL;
ALTER TABLE Limpiador ALTER COLUMN Salario SET NOT NULL;

-- Entidad
ALTER TABLE Limpiador ADD CONSTRAINT limpiador_pkey PRIMARY KEY (IdPersona);

-- Referencial
ALTER TABLE Limpiador ADD CONSTRAINT limpiador_fkey FOREIGN KEY (IdPersona) REFERENCES Empleado (IdPersona);
```

Vendedor

```
-- Dominio
ALTER TABLE Vendedor ADD CONSTRAINT vendedor_d1 CHECK(Ubicacion <> '');
ALTER TABLE Vendedor ALTER COLUMN Ubicacion SET NOT NULL;
ALTER TABLE Vendedor ADD CONSTRAINT vendedor_d2 CHECK(Ventas >= 0);
ALTER TABLE Vendedor ALTER COLUMN Ventas SET NOT NULL;

-- Entidad
ALTER TABLE Vendedor ADD CONSTRAINT vendedor_pkey PRIMARY KEY (IdPersona);

-- Referencial
ALTER TABLE Vendedor ADD CONSTRAINT vendedor_fkey FOREIGN KEY (IdPersona) REFERENCES Empleado (IdPersona) ON DELETE CASCADE;
```

Encargado

```
-- Dominio
ALTER TABLE Encargado ADD CONSTRAINT encargado_d1 CHECK (Registro >= 0);
ALTER TABLE Encargado ALTER COLUMN Registro SET NOT NULL;

-- Entidad
ALTER TABLE Encargado ADD CONSTRAINT encargado_pkey PRIMARY KEY (IdPersona);

-- Referencial
ALTER TABLE Encargado ADD CONSTRAINT encargado_fkey FOREIGN KEY (IdPersona) REFERENCES Empleado (IdPersona) ON DELETE CASCADE;
```

Edicion Torneo

```
-- Dominio
ALTER TABLE EdicionTorneo ALTER COLUMN NumeroEdicion SET NOT NULL;
ALTER TABLE EdicionTorneo ALTER COLUMN FechaEvento SET NOT NULL;
```

```
ALTER TABLE EdicionTorneo ADD CONSTRAINT edicionTorneo_d1 CHECK(Nota <> '');
ALTER TABLE EdicionTorneo ADD CONSTRAINT edicionTorneo_d2 CHECK(NumeroEdicion > 0);
```

Venta

Nota: Se modifica el nombre de IdPersona a IdPersonaV para no tener duplicado en el nombre de la llave. Registra ventas a participantes o espectadores.

```
-- Dominio
ALTER TABLE Venta ADD CONSTRAINT venta_d1 CHECK (TipoPago <> '');
ALTER TABLE Venta ADD CONSTRAINT venta_d2 CHECK (TipoPago IN ('Efectivo', 'Tarjeta', 'Transferencia', 'Otro'));
ALTER TABLE Venta ADD CONSTRAINT venta_d3 CHECK (TotalSinIva IS NULL OR TotalSinIva > 0);
ALTER TABLE Venta ADD CONSTRAINT venta_d4 CHECK (FechaHora <= NOW());

-- Asegurar que la venta sea a un participante O a un espectador, no a ambos ni a ninguno.
CHECK ((IdPersona IS NOT NULL AND IdEspectador IS NULL) OR (IdPersona IS NULL AND IdEspectador IS NOT NULL));

-- Entidad
ALTER TABLE Venta ADD CONSTRAINT venta_pkey PRIMARY KEY (IdVenta);

-- Referencias
ALTER TABLE Venta ADD CONSTRAINT venta_fkey_idpersona FOREIGN KEY (IdPersona) REFERENCES Participante (IdPersona);
ALTER TABLE Venta ADD CONSTRAINT venta_fkey_idpersonav FOREIGN KEY (IdPersonaV) REFERENCES Participante (IdPersonaV);
ALTER TABLE Venta ADD CONSTRAINT venta_fkey_espectador FOREIGN KEY (IdEspectador) REFERENCES Espectador(IdEspectador);
```

Alimento

```
-- Dominio
ALTER TABLE Alimento ADD CONSTRAINT alimento_d1 CHECK (Nombre <> '');
ALTER TABLE Alimento ADD CONSTRAINT alimento_d6 CHECK ((Perecedero = TRUE AND FechaCaducidad > CURRENT_DATE) OR (Perecedero = FALSE));
ALTER TABLE Alimento ADD CONSTRAINT alimento_d3 CHECK (PrecioSinIVA IS NULL OR PrecioSinIVA > 0);
ALTER TABLE Alimento ADD CONSTRAINT alimento_d4 CHECK (PrecioConIVA IS NULL OR PrecioConIVA > 0);
ALTER TABLE Alimento ADD CONSTRAINT alimento_d5 CHECK (PrecioConIVA IS NULL OR PrecioSinIVA IS NULL OR PrecioConIVA >= PrecioSinIVA);

-- Entidad
ALTER TABLE Alimento ADD CONSTRAINT alimento_pkey PRIMARY KEY (IdAlimento);
```

Cuenta

```
-- Dominio
ALTER TABLE Cuenta ADD CONSTRAINT cuenta_d1 CHECK (CHAR_LENGTH(
    CodigoEntrenador) = 5);
ALTER TABLE Cuenta ADD CONSTRAINT cuenta_d2 CHECK (IdPersona > 0);
ALTER TABLE Cuenta ADD CONSTRAINT cuenta_d3 CHECK (NombreUsuario <> '');
ALTER TABLE Cuenta ADD CONSTRAINT cuenta_d4 CHECK (Nivel <> '');
ALTER TABLE Cuenta ADD CONSTRAINT cuenta_d5 CHECK (Equipo <> '');

-- Entidad
ALTER TABLE Cuenta ADD CONSTRAINT cuenta_pkey PRIMARY KEY (
    CodigoEntrenador);

-- Referencial
ALTER TABLE Cuenta ADD CONSTRAINT cuenta_fkey FOREIGN KEY (IdPersona)
    REFERENCES Participante (IdPersona);
```

Pokemon

```
-- Dominio
ALTER TABLE Pokemon ALTER COLUMN IdPokemon ADD GENERATED BY DEFAULT AS
    IDENTITY;
ALTER TABLE Pokemon ALTER COLUMN CodigoEntrenador SET NOT NULL;
ALTER TABLE Pokemon ALTER COLUMN Nombre SET NOT NULL;
ALTER TABLE Pokemon ALTER COLUMN Especie SET NOT NULL;
ALTER TABLE Pokemon ALTER COLUMN Shiny SET DEFAULT false;
ALTER TABLE Pokemon ALTER COLUMN Nivel SET DEFAULT 1;
ALTER TABLE Pokemon ALTER COLUMN CombatPoints SET DEFAULT 0;

-- Integridad de entidad (PK)
ALTER TABLE Pokemon ADD CONSTRAINT pokemon_pk PRIMARY KEY (IdPokemon);

-- Integridad referencial (FK)
ALTER TABLE Pokemon ADD CONSTRAINT pokemon_codigoentrenador_fk FOREIGN KEY
     (CodigoEntrenador) REFERENCES Cuenta (CodigoEntrenador) ON DELETE
    CASCADE;
```

Pelea

```
-- Entidad
ALTER TABLE Pelea ADD CONSTRAINT pelea_pkey PRIMARY KEY(IdPelea);

-- Referencial
ALTER TABLE Pelea ADD CONSTRAINT pelea_fkey_participante FOREIGN KEY(
    IdParticipante) REFERENCES Participante(IdPersona);
ALTER TABLE Pelea ADD CONSTRAINT pelea_fkey_torneo FOREIGN KEY(IdTorneo)
    REFERENCES Enfrentamiento (IdTorneo);
```

Capturar

```sql
-- Dominio
ALTER TABLE Capturar ADD CONSTRAINT capturar_d1 CHECK (IdPokemon > 0);
ALTER TABLE Capturar ADD CONSTRAINT capturar_d2 CHECK (IdPersona > 0);
ALTER TABLE Capturar ADD CONSTRAINT capturar_d3 CHECK (FechaYHora <= NOW()
    );

-- Referencias
ALTER TABLE Capturar ADD CONSTRAINT capturar_fkey_persona FOREIGN KEY (
    IdPersona) REFERENCES Participante (IdPersona);
ALTER TABLE Capturar ADD CONSTRAINT capturar_fkey_pokemon FOREIGN KEY (
    IdPokemon) REFERENCES Pokemon (IdPokemon);
ALTER TABLE Capturar ADD CONSTRAINT capturar_fkey_torneo FOREIGN KEY (
    IdTorneo) REFERENCES Torneo (IdTorneo);
```

Ganar

```sql
-- Dominio / defaults / checks
ALTER TABLE Ganar ALTER COLUMN IdPersona SET NOT NULL;
ALTER TABLE Ganar ALTER COLUMN IdTorneo SET NOT NULL;

-- Integridad de entidad (PK compuesta)
ALTER TABLE Ganar ADD CONSTRAINT ganar_pk PRIMARY KEY (IdPersona, IdTorneo
    );

-- Integridad referencial (FK)
ALTER TABLE Ganar ADD CONSTRAINT ganar_idpersona_fk FOREIGN KEY (IdPersona
    ) REFERENCES public.persona (IdPersona) ON UPDATE CASCADE ON DELETE
    RESTRICT;
ALTER TABLE Ganar ADD CONSTRAINT ganar_idtorneo_fk FOREIGN KEY (IdTorneo)
    REFERENCES public.torneo (IdTorneo) ON UPDATE CASCADE ON DELETE CASCADE
    ;
```

Participar

```sql
-- Referencial
ALTER TABLE Participar ADD CONSTRAINT participar_fkey_persona FOREIGN KEY
    (IdPersona) REFERENCES Participante (IdPersona);
ALTER TABLE Participar ADD CONSTRAINT participar_fkey_torneo FOREIGN KEY (
    IdTorneo) REFERENCES Torneo (IdTorneo);
```

Registrar

```sql
-- Dominio
ALTER TABLE Registrar ADD CONSTRAINT registrar_d1 CHECK (IdAlimento > 0);
ALTER TABLE Registrar ADD CONSTRAINT registrar_d2 CHECK (IdPersona > 0);

```

```
-- Referencias
ALTER TABLE Registrar ADD CONSTRAINT registrar_fkey_alimento FOREIGN KEY (
    IdAlimento) REFERENCES Alimento (IdAlimento);
ALTER TABLE Registrar ADD CONSTRAINT registrar_fkey_persona FOREIGN KEY (
    IdPersona) REFERENCES Participante (IdPersona);
```

Trabajar

```
-- Referencial
ALTER TABLE Trabajar ADD CONSTRAINT trabajar_fkey_persona FOREIGN KEY (
    IdPersona) REFERENCES Empleado (IdPersona);
ALTER TABLE Trabajar ADD CONSTRAINT trabajar_fkey_edicion FOREIGN KEY(
    IdEdicion) REFERENCES EdicionTorneo (IdEdicion);
```

Pelear

```
-- Referencial
ALTER TABLE Pelear ADD CONSTRAINT pelear_fkey_persona FOREIGN KEY (
    IdPersona) REFERENCES Participante (IdPersona);
ALTER TABLE Pelear ADD CONSTRAINT pelear_fkey_pelea FOREIGN KEY(IdPelea)
    REFERENCES Pelea(IdPelea);
```

Enfrentamiento

```
-- Dominio / defaults / checks
ALTER TABLE Enfrentamiento ALTER COLUMN IdTorneo ADD GENERATED BY DEFAULT
    AS IDENTITY;

-- Integridad de entidad (PK)
ALTER TABLE Enfrentamiento ADD CONSTRAINT enfrentamiento_pk PRIMARY KEY (
    IdTorneo);
```

Recorrido

```
-- Dominio
ALTER TABLE Recorrido ALTER COLUMN IdTorneo ADD GENERATED BY DEFAULT AS
    IDENTITY;

-- Integridad de entidad (PK)
ALTER TABLE Recorrido ADD CONSTRAINT recorrido_pk PRIMARY KEY (IdTorneo);
```

CapturaShiny

```sql
-- Dominio
ALTER TABLE CapturaShiny ALTER COLUMN IdTorneo ADD GENERATED BY DEFAULT AS
    IDENTITY;

-- Integridad de entidad (PK)
ALTER TABLE CapturaShiny ADD CONSTRAINT capturashiny_pk PRIMARY KEY (
   IdTorneo);
```

Torneo

```sql
-- Dominio
ALTER TABLE Torneo ALTER COLUMN IdTorneo ADD GENERATED BY DEFAULT AS
   IDENTITY;
ALTER TABLE Torneo ALTER COLUMN IdEdicion SET NOT NULL;
ALTER TABLE Torneo ALTER COLUMN Inicio SET NOT NULL;
ALTER TABLE Torneo ALTER COLUMN Final SET NOT NULL;

-- Integridad de entidad (PK)
ALTER TABLE Torneo ADD CONSTRAINT torneo_pk PRIMARY KEY (IdTorneo);

-- Integridad referencial (FK)
ALTER TABLE Torneo ADD CONSTRAINT torneo_idedicion_fk FOREIGN KEY (
   IdEdicion) REFERENCES EdicionTorneo (IdEdicion) ON UPDATE CASCADE ON
   DELETE RESTRICT;
```

Equipar

```sql

-- Dominio
ALTER TABLE Equipar ALTER COLUMN IdPersona SET NOT NULL;
ALTER TABLE Equipar ADD CONSTRAINT equipar_d1 CHECK (IdPersona > 0);

ALTER TABLE Equipar ALTER COLUMN IdTorneo SET NOT NULL;
ALTER TABLE Equipar ADD CONSTRAINT equipar_d2 CHECK (IdTorneo > 0);

ALTER TABLE Equipar ALTER COLUMN IdPokemon SET NOT NULL;
ALTER TABLE Equipar ADD CONSTRAINT equipar_d3 CHECK (IdPokemon > 0);

-- Integridad de entidad (PK compuesta)
ALTER TABLE Equipar ADD CONSTRAINT equipar_pkey PRIMARY KEY (IdPersona,
   IdTorneo, IdPokemon);

-- Integridad referencial (FK)
ALTER TABLE Equipar ADD CONSTRAINT equipar_fkey_participante
FOREIGN KEY (IdPersona) REFERENCES Participante (IdPersona)
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE Equipar ADD CONSTRAINT equipar_fkey_torneo
FOREIGN KEY (IdTorneo) REFERENCES Enfrentamiento (IdTorneo)
ON DELETE CASCADE ON UPDATE CASCADE;
```

```
23
24 ALTER TABLE Equipar ADD CONSTRAINT equipar_fkey_pokemon
25 FOREIGN KEY (IdPokemon) REFERENCES Pokemon (IdPokemon)
26 ON DELETE CASCADE ON UPDATE CASCADE;
```

DML (Data Manipulation Language)

- Table name: **Alimento**
  Rows: 1000
  Format: SQL
  Descripcion: Almacena información de los alimentos con caducidad, perecibilidad y precio sin IVA.

- Table name: **Correo**
  Rows: 5250
  Format: SQL
  Descripcion: Almacena las direcciones de correo electrónico asociadas a las personas registradas en el sistema.

- Table name: **Cuenta**
  Rows: 1400
  Format: SQL
  Descripcion: Almacena la información de las cuentas de los entrenadores que participan en el torneo, incluyendo su nombre de usuario, nivel de experiencia y equipo al que pertenecen.

- Table name: **Cuidador**
  Rows: 1000
  Format: SQL
  Descripcion: Almacena los datos de un empleado que trabaja como cuidador en un torneo, incluyendo horario y localización.

- Table name: **Empleado**
  Rows: 4000
  Format: SQL
  Descripcion: Almacena la información de aquellas personas que son empleados de un torneo, incluyendo su domicilio.

- Table name: **Encargado**
  Rows: 1000
  Format: SQL
  Descripcion: Almacena la información de los empleados con el rol de encargado, incluyendo su número de registro.

- Table name: **Espectador**
  Rows: 1000
  Format: SQL
  Descripcion: Almacena los datos personales y los horarios de ingreso y salida de los espectadores del evento.

- Table name: **Limpiador**
  Rows: 1000
  Format: SQL

Descripcion: Almacena la información específica de los empleados que desempeñan el rol de limpieza, incluyendo horario y zona asignada.

- Table name: **Participante**
  Rows: 1250
  Format: SQL
  Descripcion: Almacena información de los participantes del torneo, incluyendo su facultad, carrera y métricas de ubicación.

- Table name: **Persona**
  Rows: 5000
  Format: SQL
  Descripcion: Almacena la información básica (nombre, fecha de nacimiento, sexo) de todas las personas que participan y/o trabajan en el Torneo.

- Table name: **Pokemon**
  Rows: 5000
  Format: SQL
  Descripcion: Almacena información de los Pokemon capturados por cada entrenador, incluyendo sus estadísticas (CP, peso) y si es shiny.

- Table name: **Registrar**
  Rows: 1000
  Format: SQL
  Descripcion: Tabla intermedia que registra el detalle de los alimentos incluidos en cada venta realizada.

- Table name: **Telefono**
  Rows: 6000
  Format: SQL
  Descripcion: Almacena los números telefónicos de contacto asociados a cada persona.

- Table name: **Tipo**
  Rows: 5700
  Format: SQL
  Descripcion: Catálogo que almacena los diferentes tipos elementales de los Pokémon.

- Table name: **Vendedor**
  Rows: 1000
  Format: SQL
  Descripcion: Almacena la información de los empleados con el rol de vendedor, incluyendo su ubicación y cantidad de ventas.

- Table name: **Venta**
  Rows: 1000
  Format: SQL
  Descripcion: Registra la información general de las transacciones de venta realizadas (fecha, tipo de pago, totales).

- Table name: **EdiciónTorneo**
  Rows: 3
  Format: SQL
  Descripcion: Almacena la información de las diferentes ediciones del torneo, incluyendo número de edición, fecha del evento y notas.
- Table name: **Trabajar**
  Rows: 10,000
  Format: SQL
  Descripcion: Relación que indica qué empleados trabajan en qué edición del torneo.
- Table name: **Participar**
  Rows: 3000
  Format: SQL
  Descripcion: Relación que indica en qué torneos específicos se ha inscrito cada participante.
- Table name: **Torneo**
  Rows: 9
  Format: SQL
  Descripcion: Define las instancias generales de los torneos (Pelea, Recorrido, Shiny) por edición, con sus horarios de inicio y fin.
- Table name: **CapturaShiny**
  Rows: 3
  Format: SQL
  Descripcion: Especialización de torneo; almacena los identificadores de los torneos de tipo Captura de Shiny.
- Table name: **Recorrido**
  Rows: 3
  Format: SQL
  Descripcion: Especialización de torneo; almacena los identificadores de los torneos de tipo Recorrido (Distancia).
- Table name: **Enfrentamiento**
  Rows: 3
  Format: SQL
  Descripcion: Especialización de torneo; almacena los identificadores de los torneos de tipo Enfrentamiento (Peleas).
- Table name: **Pelear**
  Rows: 1000
  Format: SQL
  Descripcion: Tabla intermedia que relaciona a los participantes con las peleas específicas en las que compiten.
- Table name: **Pelea**
  Rows: 1000

Format: SQL
Descripcion: Entidad que representa cada combate individual llevado a cabo dentro del torneo de enfrentamientos.

- Table name: **Ganar**
  Rows: 1000
  Format: SQL
  Descripcion: Registra a los participantes que han resultado vencedores en los distintos torneos.

- Table name: **Recorrer**
  Rows: 1000
  Format: SQL
  Descripcion: Almacena la información del desempeño de los participantes en el torneo de distancia recorrida.

- Table name: **Capturar**
  Rows: 1000
  Format: SQL
  Descripcion: Registra los eventos de captura de Pokémon realizados por los participantes durante el torneo (ej. captura de Shinys).

- Table name: **Equipar**
  Rows: 1000
  Format: SQL
  Descripcion: Relación que detalla qué equipo de Pokémon ha seleccionado cada participante para utilizar en el torneo de peleas.

Consultas

A continuación se presentan las consultas realizadas para extraer información específica de la base de datos.

Consulta 1

Lista cada venta individual, calculando precio base sin IVA, IVA, precio final con IVA, ganancia neta SBA (75 %) y comisión vendedor (25 %).

```sql
SELECT v.IdVenta, v.TipoPago, v.FechaHora,
       v.TotalSinIva AS "Precio sin Iva",
       (calcular_iva_venta(v.TotalSinIva) - v.TotalSinIva) AS "IVA",
       calcular_iva_venta(v.TotalSinIva) AS "Precio con IVA",
       (v.TotalSinIva * 0.75) AS "Ganancia neta SBA",
       (v.TotalSinIva * 0.25) AS "Comision para vendedor"
FROM Venta v;
```

Consulta 2

Listar Nombre Vendedor, Nombre Alimento, FechaCaducidad, Precio con Iva del alimento y Cantidad de vendidos de ese alimento. Ordenado de mayor a menor.

```sql
SELECT per.Nombre, per.Paterno, per.Materno, ali.Nombre AS Alimento,
       ali.FechaCaducidad,
       calcular_iva_venta(ali.PrecioSinIVA) AS "Precio con IVA",
       COUNT(v.IdPersonaV) AS CantidadVendidos
FROM Alimento ali JOIN Registrar r ON ali.IdAlimento = r.IdAlimento
            JOIN Venta v ON r.IdVenta = v.IdVenta
            JOIN Vendedor ven ON v.IdPersonaV = ven.IdPersona
            JOIN Empleado emp ON ven.IdPersona = emp.IdPersona
            JOIN Persona per ON emp.IdPersona = per.IdPersona
GROUP BY ali.IdAlimento, ali.Nombre, ali.FechaCaducidad, ali.PrecioSinIVA,
       per.IdPersona, per.Nombre, per.Paterno, per.Materno
ORDER BY CantidadVendidos DESC;
```

Consulta 3

Listar alimentos perecederos ordenados por ventas, con una columna que indique si su estado es Crítico si la fecha de caducidad es el día del evento.

```sql
SELECT ali.Nombre AS Alimento, ali.FechaCaducidad,
       COUNT(reg.IdAlimento) AS Vendidos,
       CASE
           WHEN ali.FechaCaducidad IN (SELECT FechaEvento FROM
   EdicionTorneo) THEN 'Critico'
           ELSE 'Bien'
       END AS Estado
FROM Alimento ali LEFT JOIN Registrar reg ON ali.IdAlimento = reg.
   IdAlimento
WHERE ali.Perecedero = TRUE
```

```
GROUP BY ali.IdAlimento, ali.Nombre, ali.FechaCaducidad
ORDER BY Vendidos ASC;
```

Consulta 4

Listar el nombre completo, edad, sexo y fecha de nacimiento de las personas que hayan hecho al menos 2 compras con un monto total mayor a $150 pesos cada una.

```
SELECT per.Nombre, per.Paterno, per.Materno,
       (calcular_edad_persona(per.FechaNacimiento)) AS Edad, per.Sexo,
       per.FechaNacimiento
FROM Persona per JOIN Venta v ON per.IdPersona = v.IdPersona
WHERE calcular_iva_venta(v.TotalSinIva) > 150::money
GROUP BY per.IdPersona, per.Nombre, per.Paterno, per.Materno,
       per.FechaNacimiento, per.Sexo
HAVING COUNT(v.IdVenta) >= 2;
```

Consulta 5

Listar a todos los participantes de peleas. Con columnas que indiquen Total Peleas, Victorias, Derrotas y % Efectividad, ordenado por efectividad de mayor a menor.

```
SELECT per.Nombre, per.Paterno, per.Materno, par.Facultad,
       COUNT(DISTINCT pelr.IdPelea) AS TotalPeleas,
       COUNT(DISTINCT pe.IdPelea) AS Victorias,
       (COUNT(DISTINCT pelr.IdPelea) - COUNT(DISTINCT pe.IdPelea)) AS
   Derrotas,
       ROUND((COUNT(DISTINCT pe.IdPelea) * 100.0) / NULLIF(COUNT(DISTINCT
   pelr.IdPelea), 0), 2) AS "Porcentaje Efectividad"
FROM Persona per JOIN Participante par ON per.IdPersona = par.IdPersona
            JOIN Pelear pelr ON par.IdPersona = pelr.IdPersona
            LEFT JOIN Pelea pe ON par.IdPersona = pe.IdParticipante
GROUP BY per.IdPersona, per.Nombre, per.Paterno, per.Materno, par.Facultad
ORDER BY "Porcentaje Efectividad" DESC;
```

Consulta 6

Listar las especies de Pokémon. Con columnas que indiquen: Cantidad de cada especie registrada, Promedio de CP, Peso Promedio, ordenado por Especie.

```
SELECT po.Especie,
       COUNT(po.IdPokemon) AS CantidadRegistrada,
       ROUND(AVG(po.CombatPoints), 2) AS "Promedio de CombatPoints",
       ROUND(AVG(po.Peso), 2) AS "Peso Promedio"
FROM Pokemon po
GROUP BY po.Especie
ORDER BY po.Especie;
```

Consulta 7

Listar a todos los espectadores que ingresaron al evento, calculando cuántas horas permanecieron dentro. Solamente aquellos que se quedaron más de 4 horas.

```
SELECT esp.Nombre, esp.Paterno, esp.Materno,
       (calcular_edad_espectador(esp.FechaNacimiento)) AS Edad, esp.Genero
   ,
       (esp.HoraSalida - esp.HoraIngreso) AS "Horas Permanecidas"
FROM Espectador esp
WHERE (EXTRACT (HOUR FROM (esp.HoraSalida - esp.HoraIngreso))) > 4;
```

Consulta 8

Mostrar cuántos miembros de Sabiduría, Instinto y Valor hay por cada Facultad y carrera.

```
SELECT par.Facultad, par.Carrera, c.Equipo,
       COUNT(par.IdPersona) AS "Cantidad de Miembros"
FROM Participante par JOIN Cuenta c ON par.IdPersona = c.IdPersona
GROUP BY par.Facultad, par.Carrera, c.Equipo
ORDER BY par.Facultad, par.Carrera, c.Equipo;
```

Consulta 9

Calcular la edad exacta en años, meses y días de todos los participantes y espectadores registrados, clasificándolos por "Generación Z" o "Millennial" según su año de nacimiento, ordenado del más joven al más viejo.

```
SELECT per.Nombre, per.Paterno, per.Materno, per.FechaNacimiento,
       (calcular_edad_persona(per.FechaNacimiento)) AS "Edad en Anos",
       (EXTRACT(MONTH FROM AGE(per.FechaNacimiento))) AS "Meses",
       (EXTRACT(DAY FROM AGE(per.FechaNacimiento))) AS "Dias",
       CASE
           WHEN EXTRACT(YEAR FROM per.FechaNacimiento) BETWEEN 1981 AND
   1996 THEN 'Millennial'
           WHEN EXTRACT(YEAR FROM per.FechaNacimiento) BETWEEN 1997 AND
   2010 THEN 'Generacion Z'
           ELSE 'Otra Generacion'
       END AS Generacion,
       'Participante' AS Tipo
FROM Persona per JOIN Participante par ON per.IdPersona = par.IdPersona
UNION
SELECT esp.Nombre, esp.Paterno, esp.Materno, esp.FechaNacimiento,
       (calcular_edad_espectador(esp.FechaNacimiento)) AS "Edad en Anos",
       (EXTRACT(MONTH FROM AGE(esp.FechaNacimiento))) AS "Meses",
       (EXTRACT(DAY FROM AGE(esp.FechaNacimiento))) AS "Dias",
       CASE
           WHEN EXTRACT(YEAR FROM esp.FechaNacimiento) BETWEEN 1981 AND
   1996 THEN 'Millennial'
           WHEN EXTRACT(YEAR FROM esp.FechaNacimiento) BETWEEN 1997 AND
   2010 THEN 'Generacion Z'
           ELSE 'Otra Generacion'
```

```
        END AS Generacion,
        'Espectador' AS Tipo
FROM Espectador esp
ORDER BY "Edad en Anos" ASC, "Meses" ASC, "Dias" ASC;
```

Consulta 10

Calcular el nivel promedio de los participantes agrupado por Facultad y Carrera.

```
SELECT par.Facultad, par.Carrera,
       ROUND(AVG(c.Nivel), 2) AS "Nivel Promedio"
FROM Participante par JOIN Cuenta c ON par.IdPersona = c.IdPersona
GROUP BY par.Facultad, par.Carrera
ORDER BY par.Facultad, par.Carrera;
```

Consulta 11

Mostrar a todos los participantes del torneo de distancia, con columnas calculadas que muestren "Llegó a Rectoría: SÍ/NO", "Llegó a Universum: SÍ/NO" y "Llegó a entrada de CU: SÍ/NO".

```
SELECT per.Nombre, per.Paterno, per.Materno, par.NoCuenta, par.Facultad,
    par.Carrera,
       MAX(CASE
           WHEN LOWER(rec.Ubicacion) = 'rectoria' THEN 'Si'
           ELSE 'NO'
       END) AS "Llego a Rectoria",
       MAX(CASE
           WHEN LOWER(rec.Ubicacion) = 'universum' THEN 'Si'
           ELSE 'NO'
       END) AS "Llego a Universum",
       MAX(CASE
           WHEN LOWER(rec.Ubicacion) = 'cu' THEN 'Si'
           ELSE 'NO'
       END) AS "Llego a entrada de CU"
FROM Persona per JOIN Participante par ON per.IdPersona = par.IdPersona
            LEFT JOIN Recorrer rec ON par.IdPersona = rec.IdPersona
WHERE per.IdPersona IN (SELECT IdPersona FROM Participar) AND (rec.
    IdTorneo IN (SELECT IdTorneo FROM Recorrido) OR rec.IdTorneo IS NULL)
GROUP BY per.IdPersona, per.Nombre, per.Paterno, per.Materno, par.NoCuenta
    , par.Facultad, par.Carrera;
```

Consulta 12

Listar el nombre completo, edad, fecha de nacimiento, sexo y ciudad de los trabajadores que su codigo postal empiece con 4 pero que no sean Encargados.

```
SELECT per.Nombre, per.Paterno, per.Materno,
       (calcular_edad_persona(per.FechaNacimiento)) AS Edad,
       per.FechaNacimiento, per.Sexo, emp.Ciudad, emp.CP
FROM Persona per JOIN Empleado emp ON per.IdPersona = emp.IdPersona
WHERE CAST(emp.CP AS TEXT) LIKE '4%' AND emp.IdPersona NOT IN (SELECT
    IdPersona FROM Encargado);
```

Consulta 13

Calcular cuántos Pokémons registró cada participante para el torneo de peleas por cada una de las ediciones.

```
SELECT per.IdPersona AS Participante,
    per.Nombre || ' ' || per.Paterno || ' ' || per.Materno AS Participante
    ,
    et.NumeroEdicion AS Edicion,
    et.FechaEvento,
    COUNT(DISTINCT pok.IdPokemon) AS Total_Pokemons_Registrados
FROM Participante part
            JOIN Persona per ON part.IdPersona = per.IdPersona
            JOIN Cuenta c ON c.IdPersona = part.IdPersona
            JOIN Pokemon pok ON pok.CodigoEntrenador = c.CodigoEntrenador
            JOIN Pelear pel ON pel.IdPersona = part.IdPersona
            JOIN Pelea pea ON pea.IdPelea = pel.IdPelea
            JOIN Torneo t ON t.IdTorneo = pea.IdTorneo
            JOIN EdicionTorneo et ON et.IdEdicion = t.IdEdicion
GROUP BY per.IdPersona, per.Nombre, per.Paterno, per.Materno, et.IdEdicion
    , et.NumeroEdicion, et.FechaEvento
ORDER BY et.NumeroEdicion, Total_Pokemons_Registrados DESC;
```

Consulta 14

Listar los Pokémones shinys, que fueron capturados durante el evento, únicamente si fueron capturados entre las 14:00hrs y las 18:00hrs.

```
SELECT * FROM Pokemon INNER JOIN Capturar ON Pokemon.IdPokemon = Capturar.
    IdPokemon
WHERE Pokemon.Shiny = TRUE AND CAST(Capturar.FechaYHora as TIME) BETWEEN '
    14:00:00' AND '18:00:00';
```

Consulta 15

Obtener la lista de participantes que estén inscritos en el Torneo de Captura de Shiny y a su vez que no estén inscritos en el torneo de distancia recorrida.

```
SELECT DISTINCT pa.IdPersona, per.Nombre, per.Paterno, per.Materno
FROM Participar pa
            INNER JOIN Persona per ON pa.IdPersona = per.IdPersona
            INNER JOIN CapturaShiny cs ON pa.IdTorneo = cs.IdTorneo
WHERE pa.IdPersona NOT IN ( SELECT pa2.IdPersona FROM Participar pa2
            INNER JOIN Recorrido r ON pa2.IdTorneo = r.IdTorneo);
```

Disparadores (Triggers)

A continuación se detallan los disparadores (triggers) implementados para mantener la integridad de las reglas de negocio en la base de datos.

Validación de cantidad de Pokémon

Este disparador valida que un participante no cuente con más de 6 pokémon registrados en un mismo torneo.

```
-- Disparadores para la Base de Datos del Torneo Pokemon
-- Valida que un participante no cuente con mas de 6 pokemon en un mismo torneo
-- Cada participante puede tener hasta un maximo de 6 pokemon en un torneo especifico pero no mas.

CREATE OR REPLACE FUNCTION check_num_pokemones()
RETURNS TRIGGER AS
$$
DECLARE
    pokemones INTEGER; -- Variable donde guardaremos el conteo de pokemon del participante en el torneo
BEGIN
    SELECT COUNT(*) INTO pokemones
    FROM Equipar
    WHERE idPersona = NEW.idPersona
      AND idTorneo = NEW.idTorneo;

    IF TG_OP = 'INSERT' OR
       (TG_OP = 'UPDATE' AND (NEW.idPersona != OLD.idPersona OR NEW.idTorneo != OLD.idTorneo)) THEN
        pokemones := pokemones + 1;
    END IF;

    IF pokemones > 6 THEN
        RAISE EXCEPTION 'Solamente se admiten hasta 6 pokemon por participante en un torneo.';
    END IF;

    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

-- Trigger
CREATE TRIGGER trg_check_num_pokemones
BEFORE INSERT OR UPDATE ON Equipar
FOR EACH ROW
EXECUTE FUNCTION check_num_pokemones();
```

Restricción de Torneos de Pelea

Este disparador valida que un participante inscrito en un torneo de tipo "Pelea" (Enfrentamiento) no pueda inscribirse en otros torneos simultáneamente, garantizando la exclusividad requerida.

```
-- Valida que un participante inscrito en un torneo de pelea no pueda estar en otro torneo.
-- Solamente puede estar en mas de un torneo un participante si no pertenece a un torneo de pelea.

CREATE OR REPLACE FUNCTION check_participante_en_torneos()
RETURNS TRIGGER AS
$$
DECLARE
    es_pelea_nuevo BOOLEAN;
    ya_en_pelea BOOLEAN;
BEGIN
    -- Primero verificamos si el torneo nuevo es de pelea
    SELECT EXISTS (
        SELECT 1 FROM Enfrentamiento e WHERE e.idtorneo = NEW.idtorneo
    ) INTO es_pelea_nuevo;

    -- Comprobamos si el participante ya esta inscrito en algun torneo de pelea, no necesariamente el mismo
    SELECT EXISTS (
        SELECT 1 FROM Participar pa
        JOIN Enfrentamiento e ON e.idtorneo = pa.idtorneo
        WHERE pa.idpersona = NEW.idpersona
          AND (TG_OP = 'INSERT' OR pa.idtorneo <> NEW.idtorneo)
    ) INTO ya_en_pelea;

    -- Si ya esta en un torneo de pelea, no puede inscribirse en ningun otro
    IF ya_en_pelea THEN
        RAISE EXCEPTION 'Un participante en torneo de pelea no puede estar en otros torneos.';
    END IF;

    -- Si el torneo nuevo es de pelea y el participante ya esta inscrito en otro torneo entonces no puede inscribirse
    IF es_pelea_nuevo THEN
        IF EXISTS (
            SELECT 1 FROM Participar pa
            WHERE pa.idpersona = NEW.idpersona
              AND (TG_OP = 'INSERT' OR pa.idtorneo <> NEW.idtorneo)
        ) THEN
            RAISE EXCEPTION 'Un participante en torneo de pelea no puede estar en otros torneos.';
        END IF;
    END IF;

    RETURN NEW;
END;
```

```
$$
LANGUAGE plpgsql;

-- Trigger sobre Participar
CREATE TRIGGER trg_check_participante_en_torneos
BEFORE INSERT OR UPDATE ON Participar
FOR EACH ROW
EXECUTE FUNCTION check_participante_en_torneos();
```

Procesos Almacenados

A continuación se detallan los procedimientos almacenados desarrollados para facilitar la gestión y administración de los datos del torneo.

Mostrar Información de Participantes

Este procedimiento recorre la tabla de participantes y realiza una unión con la tabla de personas para mostrar en la consola la información detallada de cada inscrito.

```
-- Procedimiento para mostrar a todos los participantes que estan registrados y almacenados en nuestra base de datos.
CREATE OR REPLACE PROCEDURE mostrar_informacion_participantes() AS
$$
DECLARE
    fila RECORD;
    i INT := 1;
BEGIN
    FOR fila IN
        SELECT
            Participante.IdPersona,
            Persona.Nombre || ' ' || Persona.Paterno || ' ' || Persona.Materno AS NombreCompleto,
            Participante.NoCuenta,
            Participante.Facultad,
            Participante.Carrera,
            calcular_edad_persona(Persona.FechaNacimiento) AS Edad,
            Persona.Sexo
        FROM Participante
        JOIN Persona ON Participante.IdPersona = Persona.IdPersona
    LOOP
        RAISE NOTICE 'Participante %: IdPersona = % | Nombre = % | NoCuenta = % | Facultad = % | Carrera = % | Edad = % | Sexo = %',
            i, fila.IdPersona, fila.NombreCompleto, fila.NoCuenta, fila.Facultad, fila.Carrera, fila.Edad, fila.Sexo;
        i := i + 1;
    END LOOP;
END;
$$
LANGUAGE plpgsql;
```

Conversión de Encargado a Participante

Este procedimiento permite que un Encargado se registre también como participante, validando previamente su existencia.

```
-- Procedimiento para convertir un Encargado en Participante para que pueda participar en diferentes torneos.
CREATE OR REPLACE PROCEDURE encargado_a_participante(
    p_idpersona BIGINT,
    p_idtorneo BIGINT,
    p_nocuenta BIT(9),
```

```
    p_facultad VARCHAR,
    p_carrera VARCHAR,
    p_ubicacion NUMERIC,
    p_distancia NUMERIC,
    p_hora TIME
) AS
$$
DECLARE
    existe INT;
BEGIN
    -- Primero nos aseguramos que el idPersona del Encargado exista
    SELECT COUNT(*) INTO existe FROM Encargado WHERE IdPersona =
   p_idpersona;
    IF existe = 0 THEN
        RAISE EXCEPTION 'El IdPersona % no corresponde a ningun Encargado
   registrado, no fue posible la conversion a Participante.', p_idpersona;
    END IF;

    -- Una vez que vemos que existe dicho encargado, lo insertamos ahora
   como un participante
    INSERT INTO Participante (
        IdPersona, IdTorneo, NoCuenta, Facultad, Carrera, Ubicacion,
   Distancia, Hora
    ) VALUES (
        p_idpersona, p_idtorneo, p_nocuenta, p_facultad, p_carrera,
   p_ubicacion, p_distancia, p_hora
    );

    RAISE NOTICE 'Encargado % convertido en participante y sera
   participante del torneo %', p_idpersona, p_idtorneo;
END;
$$
LANGUAGE plpgsql;
```

Cambio de Pokémon en Torneo

Gestiona la sustitución de un Pokemon por otro en la tabla Equipar, validando la propiedad del Pokémon y evitando duplicados en el equipo.

```
-- Procedimiento para permitir el cambio de pokemon asociado a un
   participante en un torneo especifico.
CREATE OR REPLACE PROCEDURE cambiar_pokemon_participante(
    p_idpersona BIGINT,
    p_idtorneo BIGINT,
    p_idpokemon_old BIGINT,
    p_idpokemon_new BIGINT
) AS
$$
DECLARE
    existe_old INT;
    existe_new INT;
BEGIN
```

```
    -- Verificamos que exista el pokemon antiguo asociado al participante en el torneo para poder garantizar el cambio
    SELECT COUNT(*) INTO existe_old
    FROM Equipar
    WHERE idPersona = p_idpersona
      AND idTorneo = p_idtorneo
      AND idPokemon = p_idpokemon_old;

    IF existe_old = 0 THEN
        RAISE EXCEPTION 'El Pokemon antiguo no existe para este participante en el torneo.';
    END IF;

    -- Nos aseguramos de que el nuevo pokemon pertenezca al participante mediante su cuenta
    SELECT COUNT(*) INTO existe_new
    FROM Pokemon
    WHERE IdPokemon = p_idpokemon_new
      AND CodigoEntrenador = (
          SELECT CodigoEntrenador FROM Cuenta WHERE IdPersona = p_idpersona
      );

    IF existe_new = 0 THEN
        RAISE EXCEPTION 'El nuevo Pokemon no forma parte de alguna de las cuentas del participante.';
    END IF;

    -- Por ultimo, checamos que el nuevo pokemon a asignar no este ya registrado en el torneo
    IF EXISTS (
        SELECT 1 FROM Equipar
        WHERE idPersona = p_idpersona
          AND idTorneo = p_idtorneo
          AND idPokemon = p_idpokemon_new
    ) THEN
        RAISE EXCEPTION 'El Pokemon que se desea asignar ya esta registrado para este participante en el torneo.';
    END IF;

    -- Una vez validados las condiciones anteriores, procedemos ahora si a hacer el cambio correspondiente
    DELETE FROM Equipar
    WHERE idPersona = p_idpersona
      AND idTorneo = p_idtorneo
      AND idPokemon = p_idpokemon_old;

    INSERT INTO Equipar (idPersona, idTorneo, idPokemon)
    VALUES (p_idpersona, p_idtorneo, p_idpokemon_new);

    RAISE NOTICE 'El proceso de cambio de pokemon para el participante % en el torneo % se realizo con exito.', p_idpersona, p_idtorneo;
END;
$$
```

```
58 LANGUAGE plpgsql;
```