Memorias Del Proyecto Paranoia

Sendoa Aldama Ramos

DAW1

Base de Datos

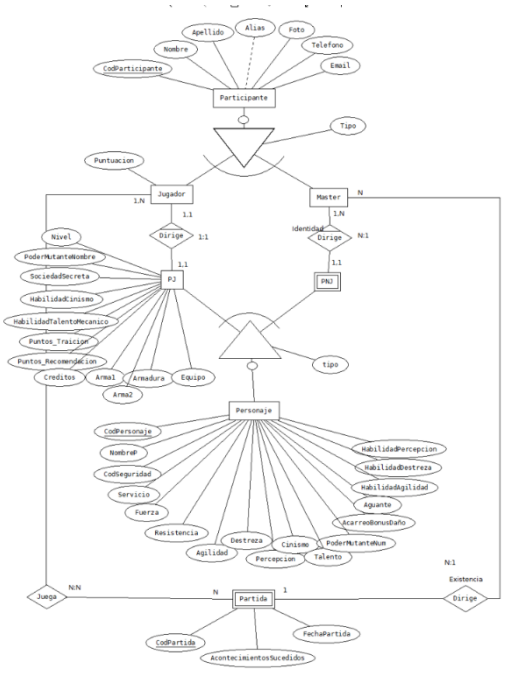
INDICE

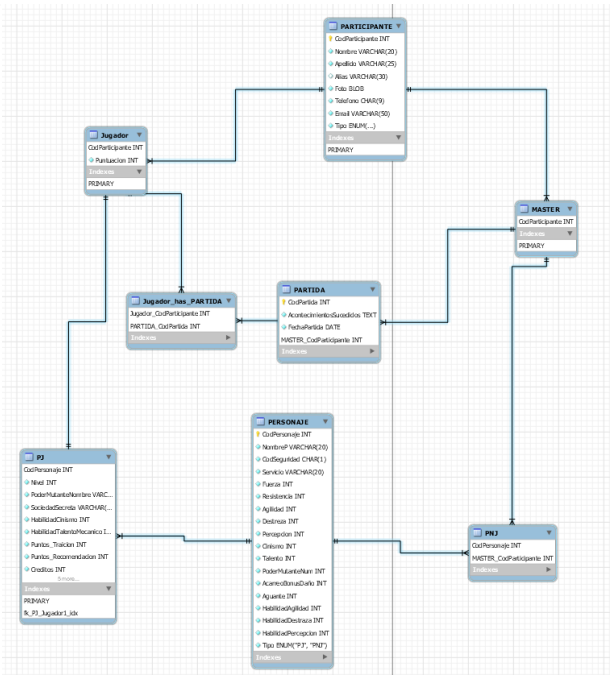
HITO1

En el Hito1 comenzamos creando la entidad de relación y la pata de gallo con los PDF e instrucciones que nos dieron.

En ellos teníamos información de como era la ficha de un personaje, sus habilidades y la formula con la que íbamos a conseguir el número que deben de tener en las características de los personajes, lo que tenía que contener un participante, el jugador, el Personaje del Jugador (PJ), el Personaje no Jugable (PNJ) y los datos que debíamos tener en una partida.

En principio cuando comenzamos este era nuestra entidad de relación y nuestra pata de gallo:





Nuestra idea principal era que a partida nos uníamos mediante Jugador y Master, ya que el Jugador solo iba a tener un PJ e iba a estar unido mediante su CodParticipante que luego lo íbamos a cambiar a CodJugador y el Master tenia barios PNJ debajo de su CodParticipante que luego cambiamos por CodMaster, el CodParticipante lo va a heredar de su tabla padre Participante.

Luego los CodPersonaje de PJ y PNJ lo heredan de Personaje y los CodParticipantes los reciben mediante una FK hacia Jugador y Master.

Partida a Master tiene una debilidad de existencia porque sin el master no hay una partida al igual que de PNJ a Master hay una de Identidad porque si no hay master los PNJ no son dirigidos ni se sabe quién los dirige.

Las cardinalidades que tenemos serían las siguientes:

- Jugador Dirige PJ son una cardinalidad de 1:1 porque un Jugador solo dirige a un PJ y un PJ solo puede ser dirigido por un Jugador.

- Master Dirige PNJ son una cardinalidad de N:1 porque un Master puede dirigir a más de un PNJ, pero solo puede a ver un Master dirigiendo la partida.

- Jugador Juega Partida son una cardinalidad de N:N porque hay muchos Jugadores Jugando muchas Partidas y un Jugador puede Jugar muchas Partidas

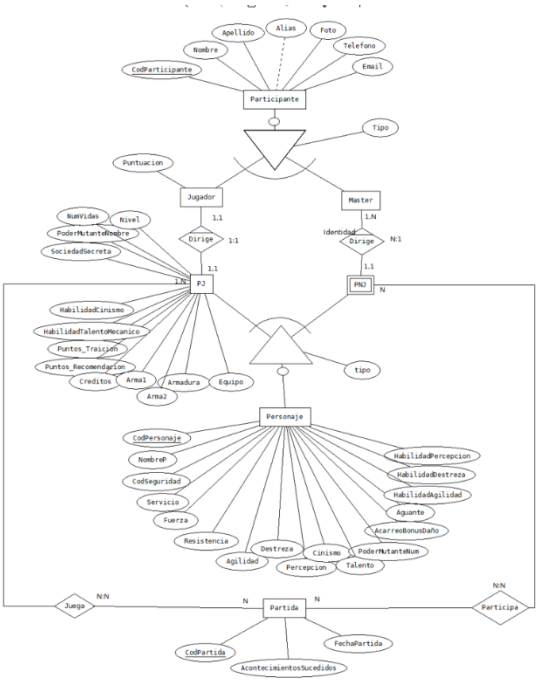
- Master Dirige Partida son una cardinalidad de N:1 porque un master puede Dirigir varias Partidas y solo puede haber un Master por Partida Dirigiendo

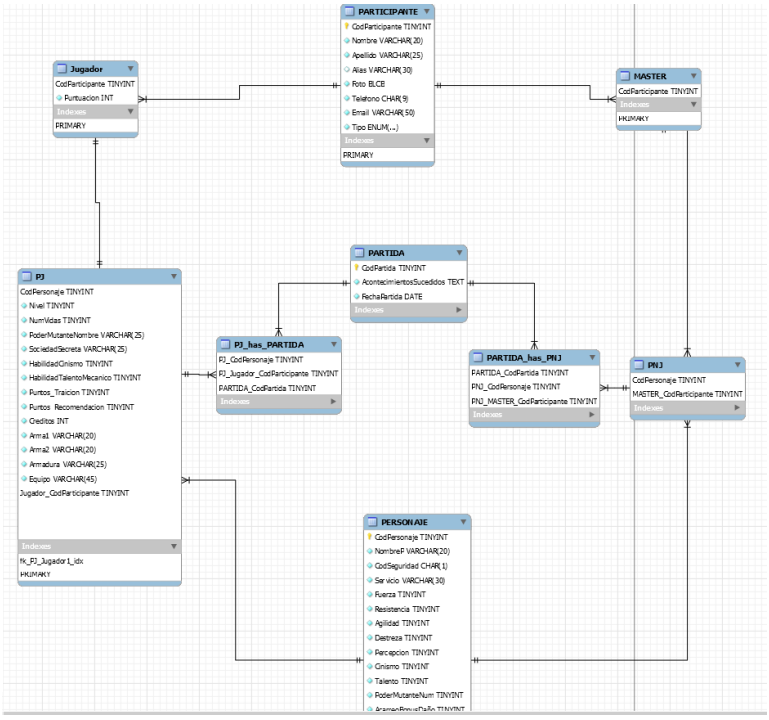
También los tipos de valores que puse en la pata de gallo, muchos eran valores que eran mejorables para la base de datos como en los Códigos de cada una de las tablas los cambié del valor INT a TINYINT que eran más correctos.

Luego sabremos más adelante sobre que, por ejemplo, Alias se convertirá en un valor tipo NOT NULL porque lo necesitaremos para hacer las rutinas para el HITO3 que aún queda para ello.

Todo esto es la parte del HITO1 que no fue APTO

Esto es lo que presentamos por segunda vez y fue APTO:





En este se puede observar que cambiamos la unión de Jugador y Master a partida por PJ y PNJ a partida ya que en sus tablas vendrán introducidos los códigos de los participantes que les controlan.

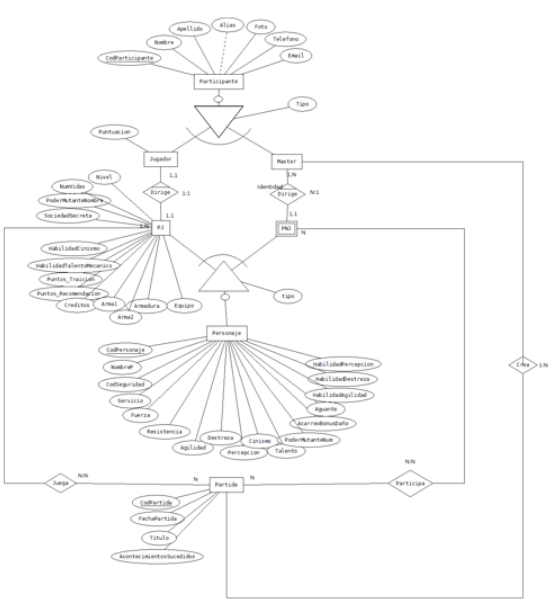
También en PJ hemos añadido una columna más llamada NumVidas que funcionara del 1 al 6 y pasara a 0 cuando se le acaben las vidas, hemos comenzado a cambiar los valores comentados anteriormente que estaban mal por unos más precisos y correctos.

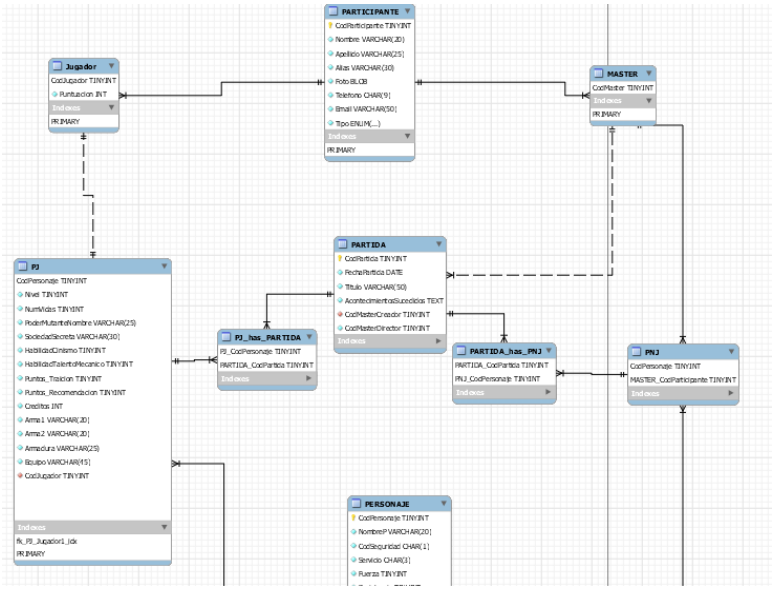
También en este caso tenemos dos tablas de unión que se llaman PJ_has_Partida y PNJ_has_Partida que luego les cambiaremos el nombre en la base de datos por PJ_Partida y PNJ_Partida que contendrán el CodJugador o CodMaster, el CodPersonaje que determinan que personaje están jugando y el CodPartida para saber que partida están jugando.

Luego añadiremos a NombreP que esta dentro de la tabla Personaje le añadiremos un VARCHAR(30) y por defecto se observará que tendrá un nombre ya que este puesto para saber si funcionan las funciones del HITO3, también tendrá que llevar un nombre en especial si es PJ, ya que estará formado por el nombre que le indiquemos, el código de seguridad, su lugar de residencia y el numero de vidas que tiene el personaje.

Después de todos estos cambios y alguno más del estilo de tener que aumentar valores ya que no entraban, este es el resultado final del HITO1 con todo lo modificado en los puntos posteriores y arreglado todo lo que fuera necesario para que funcionase todo correctamente.

Este contiene lo más actualizado de la base de datos con los últimos cambios que hemos metido en la base de datos en los HITOS 2 y 3 ya que teníamos que meterlos porque eran necesarios para tenerlo más completo y son datos que nos los han dado mientras estábamos terminando la base de datos ya que querían meter más cosas y corregir errores que no le gustaban.





Creamos una unión de Master a Partida que se llama crea, ya que hay un Master que crea partidas y solo puede crear 1 Master las partidas. Entonces sus cardinalidades seria de 1:N, N partidas crea el Master y 1 Master crea las partidas.

En estas últimas fotos se puede ver los últimos cambios realizados, uno de ellos como comentábamos antes era el Alias que lo hemos puesto NOT NULL ya que lo vamos a tener que usar más adelante, otro es quitar el PK de CodMaster y unir este mediante una FK a Partida en la columna CodMasterCreador, también vemos que en las tablas PJ_HAS_PARTIDA Y PNJ_HAS_PARTIDA tienen una columna menos, ya que ahora no va a recibir ni el CodJugador ni el CodMaster, ya que el CodJugador que está en PJ no es una PK.

En PJ vemos que ahora CodJugador deja de ser una PK y cambia a ser solo un FK, también añadimos el número de vidas para futuras rutinas y consultas.

En Partida añadimos unas columnas nuevas que son, el Titulo con un VARCHAR(50) para guardar el Titulo de la partida, CodMasterCreador para saber el Master que creó la partida y CodMasterDirector para saber qué Master la dirige.

HITO2

En esta fase empezamos a hacer la base de datos con lo que hemos realizado en el HITO 1, en esta fase comienzo a darme cuenta de que en algunas partes necesito más espacio en los VARCHAR o cambiar el VARCHAR por algún CHAR por algunos datos en concreto como en Servicio que es un numero aleatorio en el que hacemos una consulta a una tabla maestra para recibir un CHAR(3) con el Servicio que tiene el Personaje, también hacemos cambios en los nombres de muchos de los códigos para hacer reflejo para saber de qué tabla viene ese Código.

```
CREATE TABLE IF NOT EXISTS Personaje(
  CodPersonaje TINYINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  NombreP VARCHAR(30) UNIQUE NOT NULL DEFAULT "Jugador1",
  CodSeguridad CHAR(1) NOT NULL DEFAULT 'R',
  Servicio CHAR(3) NOT NULL,
  Fuerza TINYINT UNSIGNED NOT NULL,
  Resistencia TINYINT UNSIGNED NOT NULL,
  Agilidad TINYINT UNSIGNED NOT NULL,
  Destreza TINYINT UNSIGNED NOT NULL,
  Percepcion TINYINT UNSIGNED NOT NULL,
  Cinismo TINYINT UNSIGNED NOT NULL,
  Talento TINYINT UNSIGNED NOT NULL,
  PoderMutanteNum TINYINT UNSIGNED NOT NULL,
  AcarreoBonusDanho TINYINT UNSIGNED NOT NULL,
  Aguante TINYINT UNSIGNED NOT NULL,
  HabilidadAgilidad TINYINT UNSIGNED NOT NULL,
  HabilidadDestreza TINYINT UNSIGNED NOT NULL,
  HabilidadPercepcion TINYINT UNSIGNED NOT NULL,
  Tipo ENUM("PJ", "PNJ") NOT NULL
  );
```

```
CREATE TABLE IF NOT EXISTS Personaje(
  CodPersonaje TINYINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  NombreP VARCHAR(20) NOT NULL DEFAULT '-R- -1',
  CodSeguridad CHAR(1) NOT NULL DEFAULT 'R',
  Servicio VARCHAR(30) NOT NULL,
  Fuerza TINYINT NOT NULL,
  Resistencia TINYINT NOT NULL,
  Agilidad TINYINT NOT NULL,
  Destreza TINYINT NOT NULL,
  Percepcion TINYINT NOT NULL,
  Cinismo TINYINT NOT NULL,
  Talento TINYINT NOT NULL,
  PoderMutanteNum TINYINT NOT NULL,
  AcarreoBonusDanho TINYINT NOT NULL,
  Aguante TINYINT NOT NULL,
  HabilidadAgilidad TINYINT NOT NULL,
  HabilidadDestreza TINYINT NOT NULL,
  HabilidadPercepcion TINYINT NOT NULL,
  Tipo ENUM("PJ", "PNJ") NOT NULL
  );
```

Como hemos comentado antes, NombreP tiene como defecto ese nombre en la foto de la derecha que es la antigua para saber como debe ser el formato del nombre especifico que tenemos que crear mediante el nombre indicado + el código de seguridad + residencia que será una que nos introduzca + numero de vidas que tenga el personaje, en caso de que sea un PNJ pues tendrá un nombre normal como *"El Minero".*

También para evitar que el nombre del personaje se repita, le he puesto UNIQUE, para que no haya nadie mas con ese mismo nombre.

También se observa que el NombreP en la de la izquierda que es el del ultimo se ve que es un VARCHAR(30) ya que por pruebas no entraba y en el de la derecha que es el antiguo tiene un VARCHAR(20), también se ve que todos los TINYINT tienen UNISIGNED para que este más completo, no como el otro que solo lo tienen por ejemplo las PK.

En su principio dejé a Alias que está dentro de Participante como un valor NULL, pero al tener que usarlo más adelante para unos PROCEDURES en especial, lo puse como un valor NOT NULL y poder usarlo para saber el nombre del Participante o el código del participante y poderlo buscar en la base de datos ya que el Alias lo tengo como un UNIQUE.

```
CREATE TABLE IF NOT EXISTS Participante(
    CodParticipante TINYINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(20) NOT NULL,
    Apellido VARCHAR(25) NOT NULL,
    Alias VARCHAR(30) NOT NULL UNIQUE,
    Foto BLOB,
    Telefono CHAR(9) NOT NULL UNIQUE,
    Email VARCHAR(50) NOT NULL UNIQUE,
    Tipo ENUM("Jugador", "Master") NOT NULL
    );
```

```
CREATE TABLE IF NOT EXISTS Participante(
    CodParticipante TINYINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(20) NOT NULL,
    Apellido VARCHAR(25) NOT NULL,
    Alias VARCHAR(30) NULL UNIQUE,
    Foto BLOB,
    Telefono CHAR(9) NOT NULL UNIQUE,
    Email VARCHAR(50) NOT NULL UNIQUE,
    Tipo ENUM("Jugador", "Master") NOT NULL
    );
```

Se puede ver que tambien pone UNIQUE en la que es NULL pero al necesitarla para algunas consultas decidi que fuera obligatorio tener que rellenar ese campo y asi me facilitaba mas la vida para poder hacer consultas sin tener que volverse uno loco buscando otros puntos de union igual de precisos.

Para poder hacer la prueba de introducir datos tuvimos que poner a Foto que era un BLOB a NULL para que nos dejara porque no íbamos a meter ninguna foto para no liarnos en la prueba de la base de datos.

```
CREATE TABLE IF NOT EXISTS Participante(
    CodParticipante TINYINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(20) NOT NULL,
    Apellido VARCHAR(25) NOT NULL,
    Alias VARCHAR(30) NOT NULL UNIQUE,
    Foto BLOB,
    Telefono CHAR(9) NOT NULL UNIQUE,
    Email VARCHAR(50) NOT NULL UNIQUE,
    Tipo ENUM("Jugador", "Master") NOT NULL
    );
```

También añadimos aquí esas tablas como PJ_Partida y PNJ_Partida en el que nos daremos cuenta de que en vez de ser así:

```
--          PJ_Partida

-- Crear la tabla Jugador_Partida si no existe

CREATE TABLE IF NOT EXISTS PJ_Partida(
    CodPersonaje TINYINT UNSIGNED NOT NULL,
    CodJugador TINYINT UNSIGNED NOT NULL,
    CodPartida TINYINT UNSIGNED NOT NULL,
    PRIMARY KEY (CodPersonaje, CodJugador, CodPartida)
);

--      PNJ_Partida

-- Crear la tabla PNJ_Partida si no existe

CREATE TABLE IF NOT EXISTS PNJ_Partida(
    CodPersonaje TINYINT UNSIGNED NOT NULL,
    CodMaster TINYINT UNSIGNED NOT NULL,
    CodPartida TINYINT UNSIGNED NOT NULL,
    PRIMARY KEY (CodPersonaje, CodMaster, CodPartida)
);
```

Es así por cambios que hemos tenido que hacer por información extra que hemos recibido tras iban pasando los días en los que estábamos haciendo la base de datos:

```
--          PJ_Partida

-- Crear la tabla Jugador_Partida si no existe

CREATE TABLE IF NOT EXISTS PJ_Partida(
    CodPersonaje TINYINT UNSIGNED NOT NULL,
    CodPartida TINYINT UNSIGNED NOT NULL,
    PRIMARY KEY (CodPersonaje, CodPartida)
);

--      PNJ_Partida

-- Crear la tabla PNJ_Partida si no existe

CREATE TABLE IF NOT EXISTS PNJ_Partida(
    CodPersonaje TINYINT UNSIGNED NOT NULL,
    CodPartida TINYINT UNSIGNED NOT NULL,
    PRIMARY KEY (CodPersonaje, CodPartida)
```

También en Jugador sabemos que aparte del CodJugador tendremos los puntos que tiene que será un valor por defecto de 0:

```
CREATE TABLE IF NOT EXISTS Jugador(
  CodJugador TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  Puntuacion INT NOT NULL DEFAULT 0,
  PRIMARY KEY (CodJugador)
);
```

Así queda la tabla Master:

```
CREATE TABLE IF NOT EXISTS Master(
  CodMaster TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (CodMaster)
  );
```

Tenemos que tender todas las FK de la Base de datos después de cada tabla creada, tendremos arriba de las tablas maestras necesarias para las Rutinas que tendremos que realizar en el HITO3 y las dejaremos añadidas dentro de la base de datos para tener todo

completo. Esto seria una imagen de todas las FK creadas antes y las que nos han terminado quedando tras haber realizado cambios en las tablas y en la lógica de la base de datos y de la pata de gallo:

ANTES

Son 10 ya que hay dos a doble en PJ_Partida y PNJ_Partida

```
-- FK
-- Poner las FK de jugador a paritipante (CodParticipante)

ALTER TABLE Jugador ADD FOREIGN KEY (CodJugador) REFERENCES Participante(CodParticipante) on update cascade on delete restrict;

-- Poner las FK de Master a Paritipante (CodParticipante)

ALTER TABLE Master ADD FOREIGN KEY (CodMaster) REFERENCES Participante(CodParticipante) on update cascade on delete restrict;

-- Poner la FK de PJ a CodPersonaje de Personaje y CodJugador a Jugador

ALTER TABLE PJ ADD FOREIGN KEY (CodPersonaje) REFERENCES Personaje(CodPersonaje) on update cascade on delete restrict;
ALTER TABLE PJ ADD FOREIGN KEY (CodJugador) REFERENCES Jugador(CodJugador) on update cascade on delete restrict;

-- Poner la FK de PNJ a CodPersonaje de Personaje y CodMaster a Master

ALTER TABLE PNJ ADD FOREIGN KEY (CodPersonaje) REFERENCES Personaje(CodPersonaje) on update cascade on delete restrict;
ALTER TABLE PNJ ADD FOREIGN KEY (CodMaster) REFERENCES Master(CodMaster) on update cascade on delete restrict;

-- FK de CodPersonaje a Personaje, CodJugador a Jugador y CodPartida a Partida

ALTER TABLE PJ_Partida ADD FOREIGN KEY (CodPersonaje,CodJugador) REFERENCES PJ(CodPersonaje,CodJugador) on update cascade on delete restrict;
ALTER TABLE PJ_Partida ADD FOREIGN KEY (CodPartida) REFERENCES Partida(CodPartida) on update cascade on delete restrict;

-- FK de CodPersonaje a Personaje, CodMaster a Master y CodPartida a Partida

ALTER TABLE PNJ_Partida ADD FOREIGN KEY (CodPersonaje,CodMaster) REFERENCES PNJ(CodPersonaje,CodMaster) on update cascade on delete restrict;
ALTER TABLE PNJ_Partida ADD FOREIGN KEY (CodPartida) REFERENCES Partida(CodPartida) on update cascade on delete restrict;
```

DESPUES

Son 11 separadas, hemos creado una nueva para Partida, para saber cual es el CodMasterCreador para saber que master es el que crea la partida he indicarlo en la tabla de Partida. También desaparecen las doble PK que estaban antes ya que se quita una de las columnas en ambas tablas ya que dejan de ser útiles.

```
-- Creacion de las FOREING KEY

-- Poner las FK de jugador a paritipante (CodParticipante)

ALTER TABLE Jugador ADD FOREIGN KEY (CodJugador) REFERENCES Participante(CodParticipante) on update cascade on delete restrict;

-- Poner las FK de Master a Paritipante (CodParticipante)

ALTER TABLE Master ADD FOREIGN KEY (CodMaster) REFERENCES Participante(CodParticipante) on update cascade on delete restrict;

-- Poner la FK de PJ a CodPersonaje de Personaje y CodJugador a Jugador

ALTER TABLE PJ ADD FOREIGN KEY (CodPersonaje) REFERENCES Personaje(CodPersonaje) on update cascade on delete restrict;
ALTER TABLE PJ ADD FOREIGN KEY (CodJugador) REFERENCES Jugador(CodJugador) on update cascade on delete restrict;

-- Poner la FK de PNJ a CodPersonaje de Personaje y CodMaster a Master

ALTER TABLE PNJ ADD FOREIGN KEY (CodPersonaje) REFERENCES Personaje(CodPersonaje) on update cascade on delete restrict;
ALTER TABLE PNJ ADD FOREIGN KEY (CodMaster) REFERENCES Master(CodMaster) on update cascade on delete restrict;

-- FK de CodPersonaje a Personaje y CodPartida a Partida

ALTER TABLE PJ_Partida ADD FOREIGN KEY (CodPersonaje) REFERENCES PJ(CodPersonaje) on update cascade on delete restrict;
ALTER TABLE PJ_Partida ADD FOREIGN KEY (CodPartida) REFERENCES Partida(CodPartida) on update cascade on delete restrict;

-- FK de CodPersonaje a Personaje y CodPartida a Partida

ALTER TABLE PNJ_Partida ADD FOREIGN KEY (CodPersonaje) REFERENCES PNJ(CodPersonaje) on update cascade on delete restrict;
ALTER TABLE PNJ_Partida ADD FOREIGN KEY (CodPartida) REFERENCES Partida(CodPartida) on update cascade on delete restrict;

-- FK de Partida master a master

ALTER TABLE Partida add FOREIGN KEY (CodMasterCreador) REFERENCES master (CodMaster) on update cascade on delete restrict;
```

Después de tener estas 11 FK empezamos a cambiar lo delete por las siguientes lógicas.

Y tras seguir la lógica de los borrados quedaría así:

```
-- Poner las FK de jugador a paritipante (CodParticipante)

ALTER TABLE Jugador ADD FOREIGN KEY (CodJugador) REFERENCES Participante(CodParticipante) on update cascade on delete cascade;

-- Poner las FK de Master a Paritipante (CodParticipante)

ALTER TABLE Master ADD FOREIGN KEY (CodMaster) REFERENCES Participante(CodParticipante) on update cascade on delete cascade;

-- Poner la FK de PJ a CodPersonaje de Personaje y CodJugador a Jugador

ALTER TABLE PJ ADD FOREIGN KEY (CodPersonaje) REFERENCES Personaje(CodPersonaje) on update cascade on delete cascade;
ALTER TABLE PJ ADD FOREIGN KEY (CodJugador) REFERENCES Jugador(CodJugador) on update cascade on delete restrict;

-- Poner la FK de PNJ a CodPersonaje de Personaje y CodMaster a Master

ALTER TABLE PNJ ADD FOREIGN KEY (CodPersonaje) REFERENCES Personaje(CodPersonaje) on update cascade on delete cascade;
ALTER TABLE PNJ ADD FOREIGN KEY (CodMaster) REFERENCES Master(CodMaster) on update cascade on delete restrict;

-- FK de CodPersonaje a Personaje y CodPartida a Partida

ALTER TABLE PJ_Partida ADD FOREIGN KEY (CodPersonaje) REFERENCES PJ(CodPersonaje) on update cascade on delete cascade;
ALTER TABLE PJ_Partida ADD FOREIGN KEY (CodPartida) REFERENCES Partida(CodPartida) on update cascade on delete restrict;

-- FK de CodPersonaje a Personaje y CodPartida a Partida

ALTER TABLE PNJ_Partida ADD FOREIGN KEY (CodPersonaje) REFERENCES PNJ(CodPersonaje) on update cascade on delete cascade;
ALTER TABLE PNJ_Partida ADD FOREIGN KEY (CodPartida) REFERENCES Partida(CodPartida) on update cascade on delete restrict;

-- FK de Partida master a master

ALTER TABLE Partida add FOREIGN KEY (CodMasterCreador) REFERENCES master (CodMaster) on update cascade on delete cascade;
```

El sentido en el que en algunos delete haya unos de cascade o en restriction es la importancia de que por ejemplo:

- Si borramos un Personaje, también se nos debería borrar en PNJ y en PJ, ya que están unidos y ambos son personajes pero controlados por otros, al igual que si borras un participante borras al jugador y al máster.

- Si estamos en el caso de que se borra el PNJ o el PJ, no se debería borrar ni el Master ni el Jugador, ya que el podría jugar con otro personaje, el PJ o el PNJ por así decirlo dependen del jugador o el máster para existir.

A partida le añadimos unas columnas más que son:

- Titulo
- CodMasterCreador: Hay un master que Crea la Partida y puede Dirigirla también.
- CodMasterDirector: Hay un master que Dirige la Partida y puede ser el Creador también.

También hemos ordenado un poco la tabla para que el orden del contenido fuera más fácil de memorizar para realizar las Consultas del HITO3.

El único cambio gordo es ese que hemos comentado, ya que el resto ya estaban.

Esta es la imagen de ahora:

```
CREATE TABLE IF NOT EXISTS Partida(
  CodPartida TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  FechaPartida DATE NOT NULL,
  Titulo VARCHAR(50) NOT NULL,
  AcontecimientosSucedidos TEXT,
  CodMasterCreador TINYINT UNSIGNED NULL,
  CodMasterDirector TINYINT UNSIGNED NULL,
  PRIMARY KEY (CodPartida)
);
```

CodMasterCreador y CodMasterDirector son NULL para que nos acepte la entrada de datos de sus códigos en las consultas sin que haga falta meter default.

Y esta es la de antes:

```
CREATE TABLE IF NOT EXISTS Partida(
  CodPartida TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
  AcontecimientosSucedidos TEXT NOT NULL,
  FechaPartida DATE NOT NULL,
  PRIMARY KEY (CodPartida)
);
```

En la tabla PJ hemos quitado de PK a CodJugador ya que no nos es útil y solo lo necesitamos para recibir los datos de el mediante la FK ya que ahora por los cambios realizados en la tabla PJ_Partida, ya no aparece ahí el CodJugador ya que ahora nos lo podemos ahorrar de PK en la tabla PJ, se puede observar que antes en la pata de gallo era en vez de CodJugador era CodParticipante, esto facilita más en entender de que tabla viene el código, también hemos añadido más sitio en Sociedad Secreta ya que era muy larga y no nos entraba, también hemos añadido UNSIGNED a lo que necesitaba para ser más precisos, hemos añadido un default en el que tenemos un tipo de poción ya que era NOT NULL pero en el momento de introducir información al PJ no le hemos puesto ningún equipo y así lo tenemos por lo menos más completo y por ultimo hemos añadido el NumVidas ya que lo necesitaremos para más adelante ya que aunque este en el nombre del personaje también tenemos que tenerlo en cuenta en la tabla de PJ.

Esta sería la imagen de antes y ahora:

```
CREATE TABLE IF NOT EXISTS PJ(
  CodPersonaje TINYINT UNSIGNED NOT NULL,
  CodJugador TINYINT UNSIGNED NOT NULL,
  NombreP VARCHAR(20) NOT NULL,
  NumVidas TINYINT NOT NULL DEFAULT "1",
  Nivel TINYINT NOT NULL DEFAULT "1",
  PoderMutanteNombre VARCHAR(25) NOT NULL,
  SociedadSecreta VARCHAR(25) NOT NULL,
  HabilidadCinismo TINYINT NOT NULL,
  HabilidadTalentoMecanico TINYINT NOT NULL,
  Puntos_Traicion TINYINT NOT NULL,
  Puntos_Recomendacion TINYINT NOT NULL,
  Creditos INT NOT NULL DEFAULT 100,
  Arma1 VARCHAR(20) NOT NULL DEFAULT 'Pistola laser(20)',
  Arma2 VARCHAR(20) NULL,
  Armadura VARCHAR(25) NOT NULL DEFAULT 'Reflec roja (L4)',
  Equipo VARCHAR(45) NOT NULL,
  PRIMARY KEY (CodPersonaje, CodJugador)
);
```

```
CREATE TABLE IF NOT EXISTS PJ(
  CodPersonaje TINYINT UNSIGNED NOT NULL PRIMARY KEY,
  CodJugador TINYINT UNSIGNED NOT NULL,
  NumVidas TINYINT UNSIGNED NOT NULL DEFAULT "1",
  Nivel TINYINT UNSIGNED NOT NULL DEFAULT "1",
  PoderMutanteNombre VARCHAR(25) NOT NULL,
  SociedadSecreta VARCHAR(30) NOT NULL,
  HabilidadCinismo TINYINT UNSIGNED NOT NULL,
  HabilidadTalentoMecanico TINYINT UNSIGNED NOT NULL,
  Puntos_Traicion TINYINT UNSIGNED NOT NULL,
  Puntos_Recomendacion TINYINT UNSIGNED NOT NULL,
  Creditos INT UNSIGNED NOT NULL DEFAULT 100,
  Arma1 VARCHAR(20) NOT NULL DEFAULT 'Pistola laser(20)',
  Arma2 VARCHAR(20) NULL,
  Armadura VARCHAR(25) NOT NULL DEFAULT 'Reflec roja (L4)',
  Equipo VARCHAR(45) NOT NULL DEFAULT "Jugo de manzana"
);
```

La tabla PNJ no ha recibido ningún cambio desde la creación de ella, es la única tabla que no ha recibido ningún tipo de cambio ya que, por así decirlo, las que han recibido cambio de algún tipo han sido las tablas que hacen que herede datos.

Esta es una imagen de la tabla PNJ como nos ha terminado quedando sin haber realizado ningún cambio en ella:

```
--      PNJ

-- Crear tabla PNJ si no existe

CREATE TABLE IF NOT EXISTS PNJ(
  CodPersonaje TINYINT UNSIGNED NOT null,
  CodMaster TINYINT UNSIGNED NOT NULL,
  PRIMARY KEY (CodPersonaje, CodMaster)
);
```

Esto son todos los datos que hemos actualizado y arreglados para poder llegar al HITO3 con todo perfecto para que nos funcionen todas las rutinas y no tener que volver atrás y cambiar cosas aunque estemos esperando el total visto bueno, todo esto es lo que pienso que está

completamente correcto y sin errores comprobado por tiempo de esfuerzo para que no surjan errores mientras terminamos esto, el archivo ira adjunto a este documento y contendrá también las rutinas programadas, y continuemos con el HITO3 que ya es la última parte de este proyecto y en el viene la parte más importante para saber si todo los valores y NULL están bien puestos para que esta base de datos pueda ser totalmente funcional en un posible futuro.

Para comprobar que todas las tablas están bien creadas con todas sus características y columnas, introducimos mediante los INSERT datos para comprobar de que funcionan todas y cada una de ellas y luego mediante SELECT miraremos su contenido para comprobar que se han metido correctamente.

HITO3

En este HITO lo que teníamos que hacer eran unas rutinas, estas son las rutinas que teníamos que entregar para este HITO:

- Crear rutina para dar de alta Participantes. (PROCEDURE)
  En este PROCEDURE vamos a usar unos cuantos parámetros para rellenar los datos de la tabla Participantes y podamos también diferenciar si el participante es un Jugador o es un Master, esta es una imagen de los parámetros que tendremos que meter en el CALL:

```
CREATE PROCEDURE alta(
IN jNombre VARCHAR(20),
IN jApellido VARCHAR(25),
IN jAlias VARCHAR(30),
IN jTelefono CHAR(9),
IN jEmail VARCHAR(50),
IN jTipo ENUM("Jugador","Master")
)
```

  Se puede ver que pedimos un Nombre, un Apellido, el Alias, el numero de Teléfono, el Email y el Tipo para diferenciar si es un Jugador o un Master.

  Y esta es una imagen del CALL con un Master y un Jugador:

```
CALL alta("Sendoa","Aldama","tete","678801485","SendoaAldama@gmail.com","Master");
CALL alta("Roberto","Nachos","fere","676193756","Roberto@gmail.com","Jugador");
```

  En este no hemos necesitado variables.

- Crear PJ y PNJ. (PROCEDURE)
  En este PROCEDURE solicitaremos unos parámetros para poder unir el Personaje con el Jugador o con el Master, para diferenciar si será PJ o PNJ, se lo solicitaremos en la llamada y para saber a que Jugador o a que Master le pertenece, le solicitaremos el Alias ya que es único.
  Esto es una imagen de los parámetros que tenemos:

```
CREATE PROCEDURE JugadorPartida(
IN jNombrep VARCHAR(30),
IN jAlias VARCHAR(30),
IN jSector CHAR(3),
IN jTipo ENUM('PJ','PNJ')
```

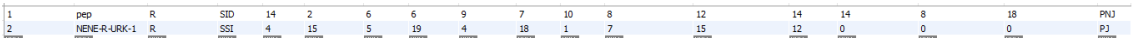
  Tenemos el NombreP que es el nombre del personaje, el Alias para saber a que Jugador le pertenece, Sector que es para saber la residencia del personaje y con ella crearemos el nombre y el tipo para saber si es PJ o PNJ, si es PJ, el nombre será la

unión de NombreP-CodigoSeguridad-Sector-NumVidas y si el numero de vidas cambia, el nombre también ya que hace referencia al clon.

Esto es la imagen de los CALL:

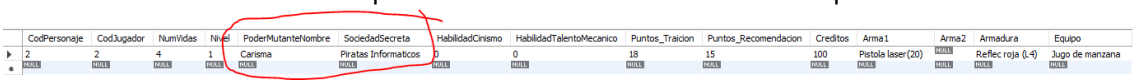
```
CALL JugadorPartida("pep","tete","URK","PNJ");
CALL JugadorPartida("NENE","fere","URK","PJ");
```

Así se ve el nombre del PJ y del PNJ:



1	pep	R	SID	14	2	6	6	9	7	10	8	12	14	14	8	18	PNJ
2	NENE-R-URK-1	R	SSI	4	15	5	19	4	18	1	7	15	12	0	0	0	PJ

Esos valores numéricos son números ramdon dados por el comando (FLOOR(1+RAND()*19) y en la tabla PJ usamos esos números para sacar el valor que contendrán mediante una búsqueda de datos en las tablas maestras que creamos.



CodPersonaje	CodJugador	NumVidas	Nivel	PoderMutanteNombre	SociedadSecreta	HabilidadCinismo	HabilidadTalentoMecanico	Puntos_Traicion	Puntos_Recomendacion	Creditos	Arma1	Arma2	Armadura	Equipo
2	2	4	1	Carisma	Piratas Informaticos	0	0	18	15	100	Pistola laser(20)	NULL	Reflec roja (L4)	Jugo de manzana
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

En este hemos necesitado estas variables:

```
-- Variables
DECLARE Letra1 TINYINT;
DECLARE Letra2 TINYINT;
DECLARE Letra3 TINYINT;
DECLARE Aleatorio TINYINT;
DECLARE JMaster TINYINT;
DECLARE Nombre VARCHAR(30);

SET Nombre = concat_ws("-",jNombrep,"R",jSector,1);
SELECT FLOOR(1+RAND()*19) INTO Letra1;
SELECT FLOOR(1+RAND()*19) INTO Letra2;
SELECT FLOOR(1+RAND()*19) INTO Letra3;
SELECT FLOOR(1+RAND()*19) INTO Aleatorio;
SET JMaster = (SELECT CodParticipante FROM Participante WHERE Alias = jAlias);
```

- Introducir una partida en la aplicación, esta partida no contendrá texto ya que se incluirá cuando finalice la partida, pero deberá tener un titulo y el master que la crea. (PROCEDURE)
  En este meteremos mediante comandos el Titulo de la partida, la fecha de creación de ella que yo la he puesto mediante un NOW para crearla en el momento del CALL, el alias del master que dirige la partida y el alias del master que crea la partida, uso esos alias para buscar ya que el alias es único, busco el master al que le pertenecen esos alias, esto es una imagen de los parámetros:

```
CREATE PROCEDURE CrearPartida(
 IN jTitulo VARCHAR(50),
 IN jFecha DATE,
 IN jAliasMasterDirige VARCHAR(30),
 IN jAliasMasterCrea VARCHAR(30)
)
```

Y esta es la imagen del CALL:

```
CALL CrearPartida("Pesos pesados",now(),"tete","tete");
```

El Alias es el mismo ya que solo tengo un master y así lo uso como creador y director ya que puede ser lo mismo el mimo master.
Antes el siguiente PROCEDURE, introduzco datos en las tablas intermedias para que me funcionen las siguientes rutinas.

```
INSERT INTO PJ_Partida VALUES(2,1);
INSERT INTO PNJ_Partida VALUES(1,1);
```

- Listar los PJs y Jugadores de una partida dada. (PROCEDURE)

  Este consiste en que en los parámetros solicitaremos el nombre del personaje y el numero de la partida.
  La imagen de los parámetros usados es:

```
IN jNombreP VARCHAR(50),
IN iNumPartida TINYINT
```

  Y la imagen del CALL es:

```
CALL ListarJugadores("NENE-R-URK-1",1);
```

  En este no nos hizo falta usar variables.

- Cuando muere un “Clon” actualizar los datos del PJ para que estén afectados y que, si llega a 0, dar por eliminado al jugador. (Cuando vaya cambiando el número de vidas, también cambiara el número que aparece en el nombre del personaje) (PROCEDURE)

  Utilizamos para saber que personaje, introducimos su nombre dentro de un CALL y el numero de vidas que ha usado, digamos que si ha muerto durante la partida 2 veces, pues ponemos en el CALL su nombre y el número de vidas usadas.

  El mío digamos que funciona según termina la partida, cogemos e indicamos el número de veces que ha puerto en esa partida, si por ejemplo a muerto 3 veces, el jugador pasa de estar en su primera vida a la cuarta. Este ejemplo se ve bien.

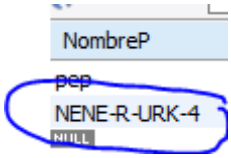
Esta es la imagen de los parámetros que hemos usado:

```
CREATE PROCEDURE DatosClon(
IN jNombre VARCHAR(30),
IN jNumeroClon TINYINT
)
```

Esta es la imagen del CALL que hemos usado para probar si funciona:

```
CALL DatosClon("NENE-R-URK-1",3);
```

Y esta es la imagen de que funciona, en la primera se puede observar como cambia el nombre y en la segunda el numero de vidas.



- Crear una rutina que según reciba como parámetro el número de clon y un carácter y proporcione un listado donde:
  Si el carácter vale "S" se listará el número de PJs que tienen gastada esa vida y desde esa hasta la última.
  Si el carácter vale "N" se listará el número de PJs que tienen gastada esa cantidad de vidas.

Ejemplo: (4,"S") listaría cuántos clones están en la vida 4, cuántos en la 5 y cuántos en la 6; y (4,"N") listaría cuántos clones están en la vida 4. (PROCEDURE)

Esta es la imagen de los parámetros que vamos a usar, parte de ellos ya nos lo dice:

```
CREATE PROCEDURE VidasPJ(
jNumClon TINYINT,
jEleccion ENUM("S","N")
)
```

Uno de ellos es el Numero de clones para filtrar y el otro es la elección entre S y N.
En esta rutina no nos a hecho falta usar ninguna variable para poderlo hacer, con los parámetros solicitados y los IF para ir seleccionando el que queremos que haga nos ha servido.
Esta es una foto de cómo es el CALL:

```
CALL VidasPJ(4,"S");
```

- Crear una rutina que según reciba como parámetro un alias de participante, o nada, proporcione un listado de los personajes asociados a ese participante donde se den sus datos y características principales y en qué partidas ha tomado parte, sea PJ o PNJ.

Nota: Ya he realizado el diseño para que, en lugar de nada, tengan que escribir ninguno como valor del parámetro, de manera que ese dato no puede ser un valor permitido para alias. (PROCEDURE)

Ya que la búsqueda la realizaremos desde Alias, pues solo necesitaremos ese parámetro, en este caso, si en vez de poner un alias de un jugador y pone literalmente *"ninguno"* se imprimirán la información de todos.
Esta es la imagen de los parámetros:

```
CREATE PROCEDURE InformacionAlias(
jAlias VARCHAR(30)
)
```

En este caso si hemos necesitado una variable que lo hemos llamado jCodP.
Esto es una imagen de esta variable y del HANDLER que hemos necesitado esta partida:

```
-- VARIABLES

DECLARE jCodP TINYINT;

-- Handlers

DECLARE EXIT HANDLER FOR NOT FOUND
    SELECT "El alias de este jugador no esta en esta base de datos" AS ERROR;

-- Codigo
```

Esta es la imagen del CALL:

```
CALL InformacionAlias ("ninguno");
```

Pongo ninguno como ejemplo para probar si funciona.

- Para que los jugadores puedan consultar los datos sobre personajes, se va a crear una vista que relacione Participantes, con sus tablas hijas, con Personajes, con sus tablas hijas, y Partida. (VIEW)

Esta es la creación y la llamada del VIEW, aparte que se pueden ver las uniones necesarias para hacer que me suelte toda la información que me tiene que dar sin que falte nada.

```
CREATE VIEW InfomacionGeneralJugadores AS

SELECT pa.Nombre, pa.Apellido, pa.Alias, pa.Telefono, pa.Email, pa.Foto, j.Puntuacion,
per.NombreP, per.CodSeguridad, per.Servicio, per.Fuerza, per.Resistencia, per.Agilidad, per.Destreza, per.Percepcion, per.Cinismo, per.Talento, per.PoderMutanteNum, per.AcarreoBonusDanho, per.Aguante, per.HabilidadAgilidad, per.HabilidadDestreza, per.HabilidadPercepcion, per.ti
pj.Puntos_Recomendacion, pj.Puntos_Traicion, pj.SociedadSecreta, pj.Nivel, pj.PoderMutanteNombre, pj.Creditos, pj.Arma1, pj.Arma2, pj.Armadura, pj.HabilidadCinismo, pj.HabilidadTalentoMecanico, pj.NumVidas, p.Titulo, p.FechaPartida
FROM Participante pa , Jugador j, Personaje per ,pj , Partida p, PJ_Partida ptp
WHERE pa.CodParticipante = j.CodJugador AND j.CodJugador = pj.CodJugador AND pj.CodPersonaje = per.CodPersonaje AND p.CodPartida = ptp.CodPartida AND ptp.CodPersonaje = pj.CodPersonaje;

SELECT * FROM InfomacionGeneralJugadores;
```

Algunas de estas rutinas tienen creadas variables por lo que he comentado y que han sido necesarias para por ejemplos los aleatorios o para en algunos casos en los

updates ya que no podíamos hacer una consulta interna y eso nos cogía lo que quería, entonces para hacerlo más preciso, teníamos que usar variables como comodines en esos casos.

Esta sería una imagen de los CALL:

```
--  ALTA
CALL alta("Sendoa","Aldama","tete","678801485","SendoaAldama@gmail.com","Master");
CALL alta("Roberto","Nachos","fere","676193756","Roberto@gmail.com","Jugador");

-- Crear PJ y PNJ
CALL JugadorPartida("pep","tete","URK","PNJ");
CALL JugadorPartida("NENE","fere","URK","PJ");

-- Crear la partida
CALL CrearPartida("Pesos pesados",now(),"tete","tete");

-- Valores para las tablas intermedias para que funcionen algunas consultas
INSERT INTO PJ_Partida VALUES(2,1);
INSERT INTO PNJ_Partida VALUES(1,1);

-- CALL para listar los jugadores
CALL ListarJugadores("NENE-R-URK-1",1);

-- Cambiar el numero de vidas del clon
CALL DatosClon("NENE-R-URK-1",3);

-- Ordenar VidasPJ
CALL VidasPJ(4,"S");

-- Informacion del alias
CALL InformacionAlias ("ninguno");

-- Llamada a la VIEW
SELECT * FROM InfomacionGeneralJugadores;
```

Esos insert son los comentados anteriormente, que son usados para que funcionen algunos de los procedures.