



ESTRUCTURA  
DE DATOS

## Sistema De Vuelos Aereos



- ANÁLISIS DE CASO -
- DISEÑO Y CONCEPTUALIZACIÓN -
- ESTRUCTURAS DE DATOS -
- GESTOR SGBD EN C++ -

Universidad Autónoma de Santo  
Domingo, UASD  
Facultad De Ciencias  
Escuela de Informática  
Miguel Arturo Sanchez Cuello

Para Dawilka, quien siempre ha sido el pilar que me sostiene y la luz que me mostraba el camino de esperanza en los momentos difíciles... antes, durante y seguramente después del semestre. Aunque siempre le vence el sueño cuando le empiezo a hablar de temas relacionados a la computación, adoro que siempre ha estado ahí para escucharme y apoyarme.

Para mis padres por la educación que me han brindado. Para Jorge, mi hermano, quien con sus formas de pensar hacia lo práctico y productivo muchas veces me sacaba de lo obtuso. A mi hermano José Miguel por ser siempre un modelo ejemplar para seguir.

Un agradecimiento a mis compañeros de clases, en especial a Fernando, Esteban, Jesús Enmanuel, Oscar, Frady, Anabel, Wally, Indiana, y otros; quienes, de alguna forma u otra, al compartir ideas y/o recursos, fueron de gran ayuda para que fuera posible la elaboración de este proyecto.

Ultimo, pero no menos importante, un agradecimiento especial a los maestros Silverio Del Orbe, Víctor Núñez, Edward Ureña y Cesar Familia quienes, con su calidad moral y esmero, despiertan la sed de conocimiento y encienden la chispa del saber en sus alumnos.

Miguel Sanchez

# Índice de Contenido

Introducción .....	3
Conceptualización .....	4
Narrativa o descripción del caso .....	4
Modelo de Base de Datos .....	5
Modelo Conceptual .....	6
Modelo Básico Entidad-Relación .....	7
Modelo Físico .....	7
Diagrama Modelo Chen .....	9
Diagrama Pata de Gallo .....	10
Modelo Físico Detallado .....	11
Construcción del Catalogo del Sistema .....	15
Creando el Archivo SysTable .....	16
Creando el archivo SysColumn .....	17
Creando el Archivo SysPrimaryKey .....	20
Creando el Archivo SysForeignKey .....	21
Creando el Gestor de Datos en C++ .....	23
Conociendo la clase SysEntity .....	24
Conociendo la clase SysColumn .....	27
Conociendo la clase SysPrimaryKey .....	29
Conociendo la clase SysForeignKey .....	31
Funcionamiento del Gestor y Ejemplos .....	33
Manipulacion de Datos a traves de la Entidad .....	38
Sobre el autor.....	43

## Introducción:

Este material se ha conceptualizado como un regalo a las generaciones futuras, de modo que lo desarrollaremos en forma explicativa de manera que no solo pueda llegarse a un resultado de producción, sino que el lector pueda comprender paso a paso lo que está haciendo. Así pues, al finalizar el documento el lector será capaz de tener el conocimiento fundamental que le permita hacer un breve análisis de un caso y de llevar a cabo un proceso de desarrollo impreso (conceptualizando, diagramando y codificando) de dicho análisis haciendo uso de Estructuras de Datos.

Aunque enfocado en el marco general, estudiaremos brevemente el caso particular Vuelos Aéreos y para las demostraciones, desarrollaremos un pequeño Sistema Gestor de Base de Datos SGBD (O del inglés DBMS, Data Base Management System) en C++ que tendrá solo las funciones fundamentales como son:

- Utilizar un catálogo de sistema (SysCatalog) para la definición de nuestras estructuras de datos que se han de cargar en la base de datos en memoria.
- Administración y manipulación de la base de datos con las funciones básicas de Alta, Baja y Búsqueda
- Validación y verificación de datos
- Entre otras.

Sin más que agregar, por el momento, y con la esperanza de que este documento sea de gran utilidad...  
¡Manos a la obra!

## Conceptualización:



Narrativa o descripción del caso:

Suponga que usted está participando en el programa de intercambios culturales, Summer Work, y ha llegado el tan esperado día de partir. Desde Santo Domingo usted va al aeropuerto con todo lo que necesita... esto incluye: su pasaporte, su equipaje y su boleto aéreo. Usted ha de interactuar con el personal del aeropuerto quien verificara su tique, administrara sus maletas y le enviara a usted en dirección al avión correspondiente que le llevara a su destino.

La situación antes definida nos sitúa en un contexto a partir del cual podemos empezar a desarrollar nuestro sistema de base de datos. Como administradores del sistema es nuestra responsabilidad hacer cumplir procedimientos y estándares, por ende, nuestro papel cambia de un enfoque a la programación a un enfoque a los aspectos mas amplios de la administración de los recursos de datos y a la administración del complejo software de la base de datos.

Para hacer esto primero tenemos que pensar en el diseño y modelado de los datos. Recordemos que *los modelos son abstracciones simplificadas de eventos y condiciones del mundo real. Por ejemplo, tales abstracciones permiten explorar las características de entidades y las relaciones que se pueden crear*

*entre ellas.*<sup>1</sup> Es recomendable hacer un buen diseño para hacer un buen modelo, puesto que un buen diseño implica una buena aplicación final.

Un **modelo de bases de datos** es un conjunto de ideas lógicas utilizadas para representar la ESTRUCTURA DE DATOS y las relaciones entre ellos dentro de la base de datos. Estos modelos se pueden agrupar en dos categorías: modelos conceptuales y modelos lógicos.<sup>2</sup>

El modelo conceptual se enfoca en la naturaleza lógica de la representación de datos. Por consiguiente, este modelo esta comprometido con lo que esta representando en la base de datos, y no en cómo está representado. Los modelos conceptuales incluyen el modelo de Entidad-Relación.<sup>3</sup>

---

<sup>1</sup> Definición de modelo tomada del libro Sistemas de Bases de Datos: Diseño, Implementación y Administración 5ª Edición página 23, autores Peter Rob y Carlos Coronel.

<sup>2</sup> Definición de modelo de bases de datos tomada del libro Sistemas de Bases de Datos: Diseño, Implementación y Administración 5ª Edición página 23, autores Peter Rob y Carlos Coronel.

<sup>3</sup> Definición de modelo conceptual tomada del libro Sistemas de Bases de Datos: Diseño, Implementación y Administración 5ª Edición página 23, autores Peter Rob y Carlos Coronel.

## Modelo Conceptual

Antes de realizar el modelo conceptual es muy importante que comprendamos la situación, por eso es recomendable describir la situación varias veces, puesto que mientras se piensa en el caso se desarrolla mejor el concepto de este.

Desde el contexto de nuestra situación descrita vamos a definir brevemente las entidades y sus atributos. Por ejemplo; Usted es un mamífero, una persona, es un cliente, es un pasajero, hay muchas formas de definir lo esto representa, sin embargo; durante este documento se tratará de utilizar la entidad que más se apegue al contexto... esto puede resultar subjetivo desde el punto de vista de la persona que realiza el análisis, con la practica y la experiencia usted afilará mejor su sentido de reconocimiento de entidades relevantes a su contexto.

Así podríamos decir que entre las Entidades mas notorias de nuestro contexto tendríamos, por ejemplo:

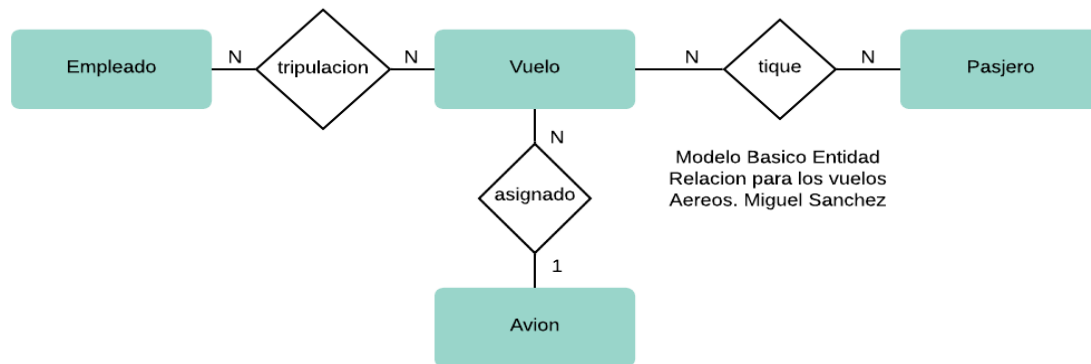
Vuelo Aéreo:

Entidad	Se corresponde a
Vuelo	La acción principal, o centro del contexto
Persona	Cliente, Pasajero
Empleado	Todo el personal del aeropuerto que interactúa con el cliente (pasajero)
Avión	Medio de transporte

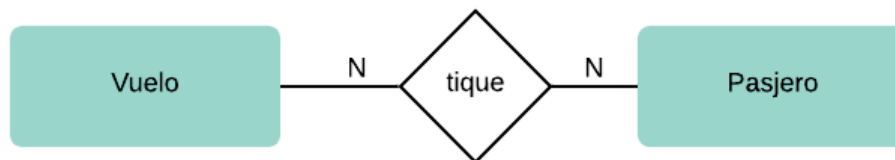
Nota: Sabemos que en un análisis más profundo se podrían incluir más Entidades y más relaciones, por ahora lo simplificaremos en la mayor medida posible para que sea mas digerible para el lector.

Relación	Entidades que relaciona
Pasaje o tique	Relaciona al pasajero con el vuelo
Asignación	Indica que vuelo es asignado a que avión
Tripulación	Relación de que personal es asignado a que vuelo

La representación básica de un Modelo Entidad Relación se muestra en la siguiente figura.<sup>4</sup>



Algo importante a mencionar es que en la practica se define una Entidad, pero también puede interpretarse como conjunto de entidades, por ejemplo, en la relación:



Podría confundir que un pasajero compra muchos tiques para muchos vuelos, aunque solo puede estar en un vuelo a la vez, así pues, Pasajero se lee como un conjunto de Entidades, aunque se representa como una entidad individual, desde el marco teórico siendo un conjunto de entidades se entiende que muchos pasajeros compren muchos tiques para muchos vuelos. Estamos trabajando desde el punto de vista de un pasajero... ya vera que a medida que avanzamos esta idea se vuelve más clara.

El diagrama antes presentado, llamado Modelo de Entidad-Relación (MER) representa el concepto de nuestro caso.

### EL MODELO FÍSICO:

*El **modelo físico** opera al mas bajo nivel de abstracción y describe la manera en la que se guardan los datos en medios magnéticos de almacenamiento como discos o cintas. El modelo físico requiere la definición tanto de dispositivos de almacenamiento físico tales como de métodos de acceso (físico) indispensables para llegar a los datos dentro de dichos dispositivos<sup>5</sup>.*

Puesto que el modelo de Entidad Relación no necesita que el diseñador se preocupe por las características físicas de almacenamiento de los datos, seguiremos definiendo nuestro diagrama de

<sup>4</sup> Puede indagar más sobre cómo crear Diagramas de Entidad Relación el libro: Sistemas de Bases de Datos: Diseño, Implementación y Administración 5ª Edición, Cap. 3 Modelado de Entidad-Relación, autores Peter Rob y Carlos Coronel.

<sup>5</sup> Definición de modelo físico del libro: Sistemas de Bases de Datos: Diseño, Implementación y Administración 5ª Edición, Cap. 3 Modelado de Entidad-Relación, Página 118 autores Peter Rob y Carlos Coronel.

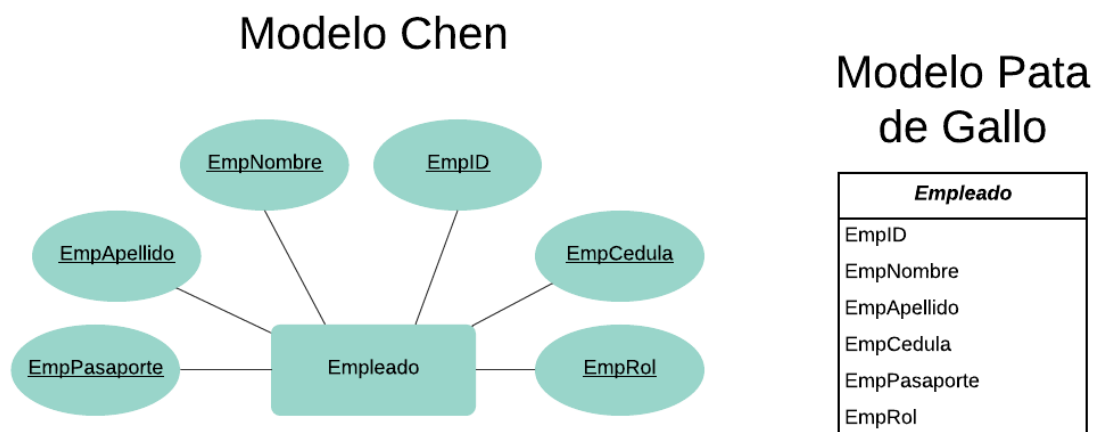


Entidad-Relación y nuestro modelo físico será representado por el tipo de dato y las reglas para que estos sean almacenados.

Es extremadamente importante que los diseñadores de bases de datos, los programadores y usuarios finales mantengan siempre una comunicación clara y precisa, sin ningún tipo de ambigüedad, puesto desde el punto individual todos podrían ver la información de manera diferente dependiendo de sus funciones y esto podría llevar a que el producto final no satisfaga las necesidades del usuario final ni los requerimientos de eficiencia de los datos.

Durante la presentación de este modelo físico vamos a presentar dos tipos de modelo de Entidad Relación, usaremos ambos para brevemente para ser mas descriptivos, el modelo Chen y el modelo Pata de Gallo<sup>6</sup>.

Antes de presentar el diagrama completo mostramos la representación de la Entidad Empleado tanto en el modelo Chen como en el modelo Pata de Gallo.

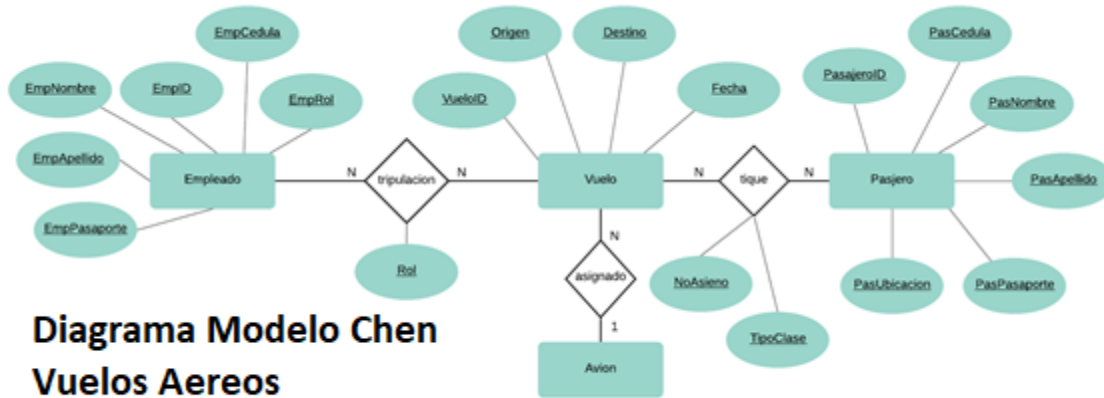


Presentacion de la Entidad Empleado en modelo Chen y modelo Pata de Gallo para los vuelos Aereos. Miguel Sanchez

Un diagrama mas terminado en el modelo Chen nos quedaría de la siguiente forma:

---

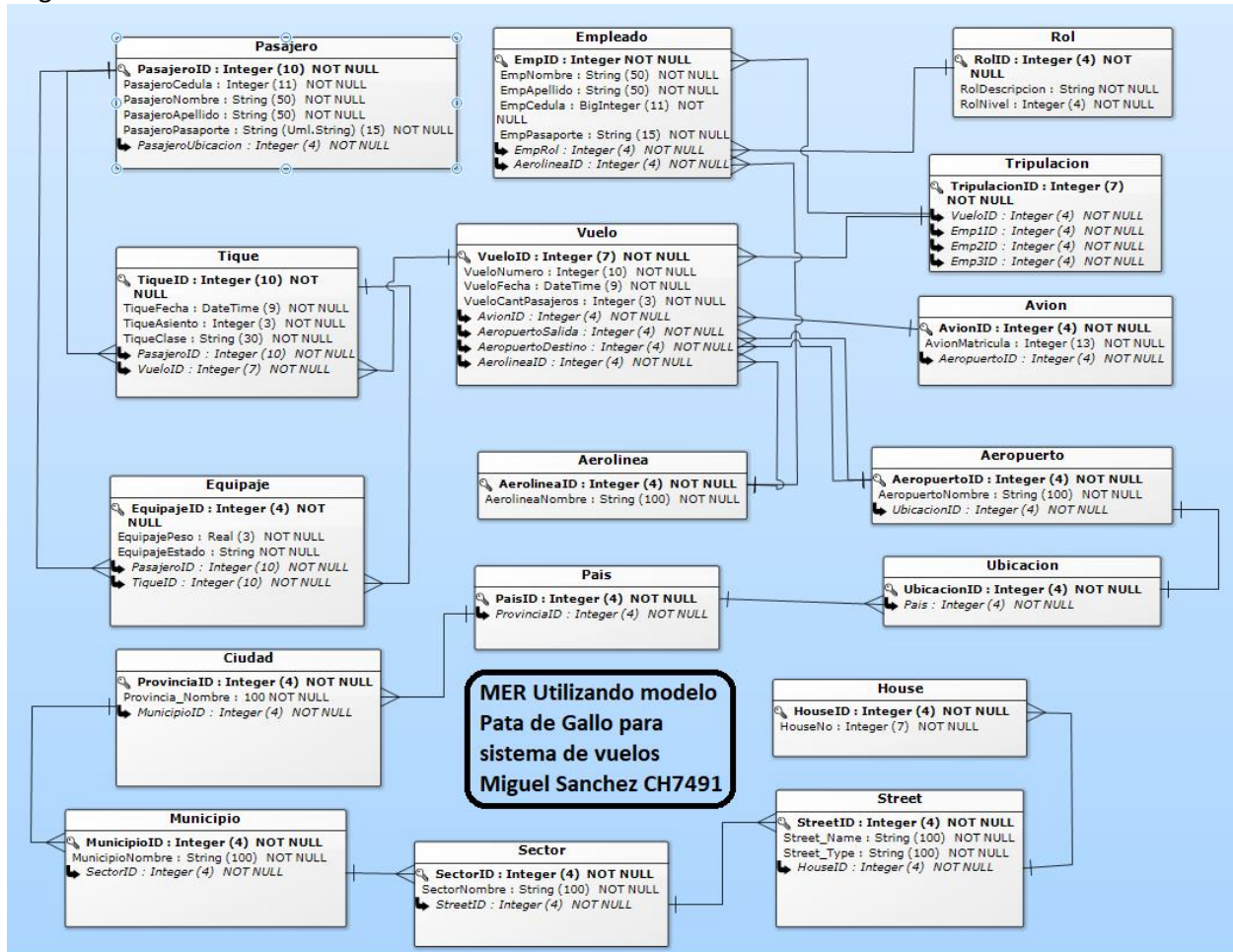
<sup>6</sup> El modelo Pata de Gallo está más orientado al a ejecución. Sistemas de Bases de Datos: Diseño, Implementación y Administración 5ª Edición, Cap. 3 Modelado de Entidad-Relación, autores Peter Rob y Carlos Coronel.



Ahora con una idea conceptual mejor definida vamos a proceder a usar el modelo Pata de Gallo para representar el modelo físico de forma que este indique las tablas (Entidades y Relaciones) con el respectivo tipo de dato a almacenar.

# Sistema de Vuelos Aéreos Con Gestor de Bases De Datos en C++

## Miguel Arturo Sanchez Cuello CH7491



Diseño realizado en el software de licencia gratuita no comercial Libre: Software Ideas Modeler.

En las siguientes paginas se detallan todas las ENTIDADES(Entity) y RELACIONES(Relationships) pertenecientes al diseño (los datos fueron exportados directamente del software de modelado). Si desea saltar a la sección del Diseño del Catálogo de Sistema salte a la página 15.

Ciudad (Entity)

-----

+ ProvincialID : Integer (4)  
  Provincia\_Nombre : 100  
# MunicipioID : Integer (4)

Municipio (Entity)

-----

+ MunicipioID : Integer (4)  
  MunicipioNombre : String (100)  
# SectorID : Integer (4)

Sector (Entity)

-----

+ SectorID : Integer (4)  
  SectorNombre : String (100)  
# StreetID : Integer (4)

Street (Entity)

-----

+ StreetID : Integer (4)  
  Street\_Name : String (100)  
  Street\_Type : String (100)  
# HouseID : Integer (4)

House (Entity)

-----

+ HouseID : Integer (4)  
  HouseNo : Integer (7)

Empleado (Entity)

-----

+ EmplID : Integer  
  EmpNombre : String (50)  
  EmpApellido : String (50)  
  EmpCedula : BigInteger (11)  
  EmpPasaporte : String (15)  
# EmpRol : Integer (4)  
# AerolinealID : Integer (4)

Rol (Entity)

-----  
+ RolID : Integer (4)  
RolDescripcion : String  
RolNivel : Integer (4)

[Rol → Empleado] (Relationship)

Tripulacion (Entity)

-----  
+ TripulacionID : Integer (7)  
# VueloID : Integer (4)  
# Emp1ID : Integer (4)  
# Emp2ID : Integer (4)  
# Emp3ID : Integer (4)

[Tripulacion → Empleado] (Relationship)

Vuelo (Entity)

-----  
+ VueloID : Integer (7)  
VueloNumero : Integer (10)  
VueloFecha : DateTime (9)  
VueloCantPasajeros : Integer (3)  
# AvionID : Integer (4)  
# AeropuertoSalida : Integer (4)  
# AeropuertoDestino : Integer (4)  
# AerolineaID : Integer (4)

[Tripulacion → Vuelo] (Relationship)

Avion (Entity)

-----  
+ AvionID : Integer (4)  
AvionMatricula : Integer (13)  
# AeropuertoID : Integer (4)

[Vuelo → Avion] (Relationship)

Aeropuerto (Entity)

-----  
+ AeropuertoID : Integer (4)  
AeropuertoNombre : String (100)  
# UbicacionID : Integer (4)

Aerolinea (Entity)

-----

+ AerolineaID : Integer (4)  
AerolineaNombre : String (100)

Ubicacion (Entity)

-----

+ UbicacionID : Integer (4)  
# Pais : Integer (4)

Equipaje (Entity)

-----

+ EquipajeID : Integer (4)  
EquipajePeso : Real (3)  
EquipajeEstado : String  
# PasajeroID : Integer (10)  
# TiqueID : Integer (10)

Tique (Entity)

-----

+ TiqueID : Integer (10)  
TiqueFecha : DateTime (9)  
TiqueAsiento : Integer (3)  
TiqueClase : String (30)  
# PasajeroID : Integer (10)  
# VueloID : Integer (7)

Pasajero (Entity)

-----

+ PasajeroID : Integer (10)  
PasajeroCedula : Integer (11)  
PasajeroNombre : String (50)  
PasajeroApellido : String (50)  
PasajeroPasaporte : String (Uml.String) (15)  
# PasajeroUbicacion : Integer (4)

[Pasajero → Tique] (Relationship)

[Pasajero → Equipaje] (Relationship)

[Tique → Equipaje] (Relationship)

[Tique → Vuelo] (Relationship)

[Empleado → Aerolinea] (Relationship)

[Vuelo → Aeropuerto] (Relationship)

Sistema de Vuelos Aéreos Con Gestor de Bases De Datos en C++

Miguel Arturo Sanchez Cuello CH7491

[Vuelo → Aeropuerto] (Relationship)

[Vuelo → Aerolinea] (Relationship)

Pais (Entity)

-----

+ PaisID : Integer (4)

# ProvincialID : Integer (4)

[Ubicacion → Pais] (Relationship)

[Pais → Ciudad] (Relationship)

[Aeropuerto → Ubicacion] (Relationship)

[Ciudad → Municipio] (Relationship)

[Municipio → Sector] (Relationship)

[Sector → Street] (Relationship)

[Street → House] (Relationship)

## Construcción del Catalogo del Sistema

El **catalogo de sistema** es un diccionario de datos de sistema muy detallado que describe todos los objetos dentro de la base, incluso los datos sobre los nombres de tabla, el creador y la fecha de creación de la tabla, el numero de columnas en cada una, el tipo de datos correspondiente a cada columna, nombres de archivo de índice, creadores del índice, usuarios autorizados, privilegios de acceso, etc. Al igual que el diccionario de datos, el catalogo de sistema contiene metadatos. Como el catalogo de sistema contiene toda la información relacionada con el diccionario de datos requerida, los términos catalogo de sistema y diccionario de datos a menudo se utilizan indistintamente. De hecho, el actual software de bases de datos relacional generalmente proporciona solo un catalogo de sistema, del que se puede obtener información sobre el diccionario de datos del diseñador. El catalogo de sistema realmente es una base de datos creada por el sistema cuyas tablas guardan las características y contenido de la que han creado el usuario o el diseñador. Por consiguiente, las tablas de catalogo de sistema pueden consultarse como cualquier otra tabla creada por estos.<sup>7</sup>

Puede encontrar mas información sobre catalogo de sistema en el libro que se ha mencionado en la nota al pie de página. Ahora vamos a elaborar un catalogo simple que se apegue estrictamente a las tablas que ya presentamos con anterioridad en el modelo entidad relación.

La complejidad de un catalogo de sistemas completo esta fuera del alcance de este documento, nos limitaremos a crear 4 archivos fundamentales para el funcionamiento de un Catálogo de Sistema simple:

Archivo	Función en el catálogo
SysTable.csv	Almacena el índice de las tablas de la base de datos con sus respectivos nombres de tabla.
SysColumn.csv	Almacena el índice de las columnas de la base de datos con sus respectivos nombres de columna.
SysPrimaryKey.csv	Almacena el índice de las llaves primarias de la base de datos.
SysForeignKey.csv	Almacena el índice de las llaves foráneas de la base de datos.

Nota: Decidimos llamar los archivos tipo CSV por definición de Archivos Separados por Coma, de las ingles (Comma Separated Values) puesto que son archivos fáciles de manejar para nuestros propósitos. Usted puede elegir el nombre que desee para sus archivos y la forma en como los datos se guardaran si desea. Nosotros por cuestiones de simplicidad usaremos el formato CSV.

Aunque parezca increíble solo con estos cuatro archivos es posible cargar una base de datos, predefinida y hacer operaciones básicas de manipulación de datos, Alta, Baja y Búsqueda (son las que demostraremos más adelante).

En la próxima pagina vamos a explicar una forma sencilla de como realizaremos nuestro catálogo de sistema.

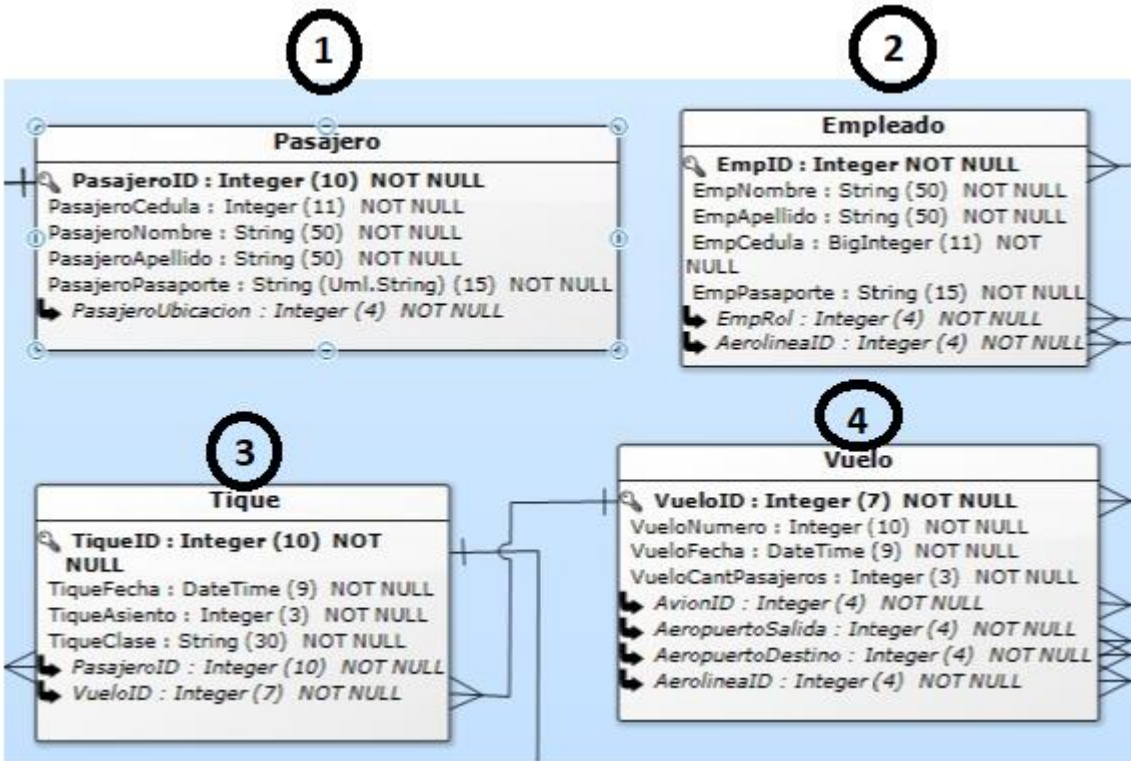
---

<sup>7</sup> Definición de catálogo de sistema del libro: Sistemas de Bases de Datos: Diseño, Implementación y Administración 5ª Edición, Cap. 2 Modelo de Base De Datos Relacional, Pág 76, autores Peter Rob y Carlos Coronel.



## Creando el archivo SysTable.csv

Nuestro primer paso es designar un ID único para cada tabla, a modo de ejemplo tomaremos las primeras cuatro tablas de la parte superior izquierda del diagrama:



Usted puede elegir el ID que prefiera para sus tablas, se sugiere números enteros cortos para que el sistema sea más eficiente al cargar los IDs. Para el ejemplo mostrado tendríamos las tablas:

TablaID	NombreTabla
1	Pasajero
2	Empleado
3	Tique
4	Vuelo

Ya que estamos realizando un proceso manual, debemos asegurarnos de que el ID de la tabla sea único, es decir que no se repita en ninguna otra tabla. Los Sistemas De diseño y gestión de bases de datos tienen definido el catalogo de sistemas de las bases de datos de forma automática, usted puede consultar dichos catálogos para aprender más sobre el tema.

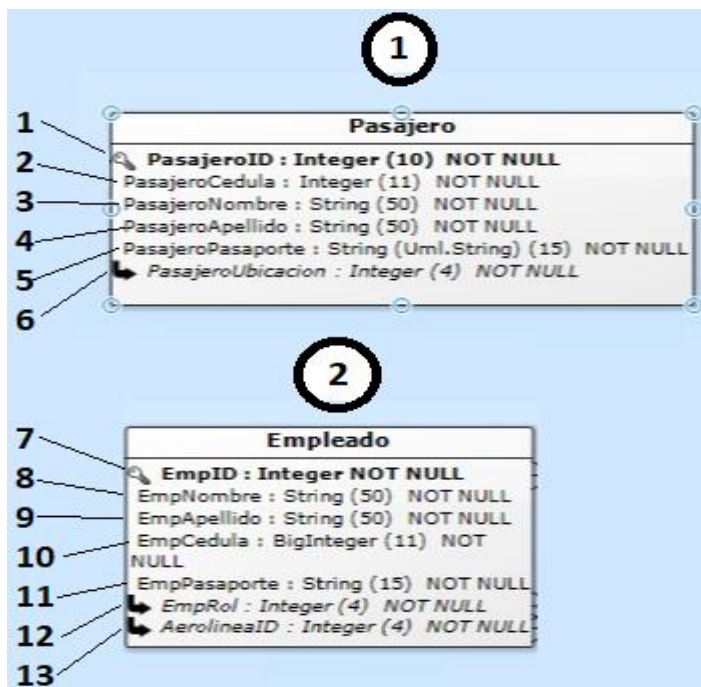
Nuestro diagrama completo consta de 17 tablas, una vez realizado el proceso, obtenemos el resultado siguiente:

TablaID,NombreTabla

- 1,Pasajero
- 2,Empleado
- 3,Rol
- 4,Tique
- 5,Vuelo
- 6,Tripulacion
- 7,Avion
- 8,Equipaje
- 9,Aerolinea
- 10,Aeropuerto
- 11,Ubicacion
- 12,Pais
- 13,Ciudad
- 14,Municipio
- 15,Sector
- 16,Street
- 17,House

## Creando el archivo SysColumn.csv

De manera similar se procede a asignar un ID único para las columnas de todas las tablas del sistema, por ejemplo:



Creando un archivo SysColumn.csv básico, sin detalles ni tipos, tendríamos la siguiente estructura:

Tabla\_ID, ColumnaID y NombreColumna; siguiendo el ejemplo anterior tendríamos una tabla de la siguiente manera:

TablaID	Column_ID	NombreColumna	Nota
1	1	PasajeroID	Durante el ejemplo omitimos las columnas adicionales que representan el tipo de datos, la longitud y si el cambio es obligatorio o no (Null o Not Null). Pero si desarrollamos el archivo principal lo más completo posible.
1	2	PasajeroCedula	
1	3	PasajeroNombre	
1	4	PasajeroApellido	
1	5	PasajeroPasaporte	
1	6	PasajeroUbicacion	
2	7	EmpID	
2	8	EmpNombre	
2	9	EmpApellido	
2	10	EmpCedula	
2	11	EmpPasaporte	
2	12	EmpRol	
2	13	AerolineaID	

Así pues, nuestro archivo SysColumn.csv resultante nos queda de la siguiente forma:

Tabla\_ID,ColumnaID,NombreColumna,Tipo,Longitud,Nulo

1,1,PasajeroID,Integer,10,FALSE  
 1,2,PasajeroCedula,Integer,11,FALSE  
 1,3,PasajeroNombre,String,50,FALSE  
 1,4,PasajeroApellido,String,50,FALSE  
 1,5,PasajeroPasaporte,String,15,FALSE  
 1,6,PasajeroUbicacion,Integer,4,FALSE  
 2,7,EmpID,Integer,4,FALSE  
 2,8,EmpNombre,String,50,FALSE  
 2,9,EmpApellido,String,50,FALSE  
 2,10,EmpCedula,Integer,11,FALSE  
 2,11,EmpPasaporte,String,15,FALSE  
 2,12,EmpRol,Integer,4,FALSE  
 2,13,AerolineaID,Integer,4,FALSE  
 3,14,RolID,Integer,4,FALSE  
 3,15,RolDescripcion,String,100,FALSE  
 3,16,RolNivel,Integer,4,FALSE  
 4,17,TiqueID,Integer,10,FALSE  
 4,18,TiqueFecha,String,9,FALSE  
 4,19,TiqueAsiento,Integer,3,FALSE  
 4,20,TiqueClase,String,30,FALSE  
 4,21,PasajeroID,Integer,10,FALSE  
 4,22,VueloID,Integer,7,FALSE  
 5,23,VueloID,Integer,7,FALSE  
 5,24,VueloNumero,Integer,10,FALSE  
 5,25,VueloFecha,String,9,FALSE  
 5,26,VueloCantPasajeros,Integer,3,FALSE

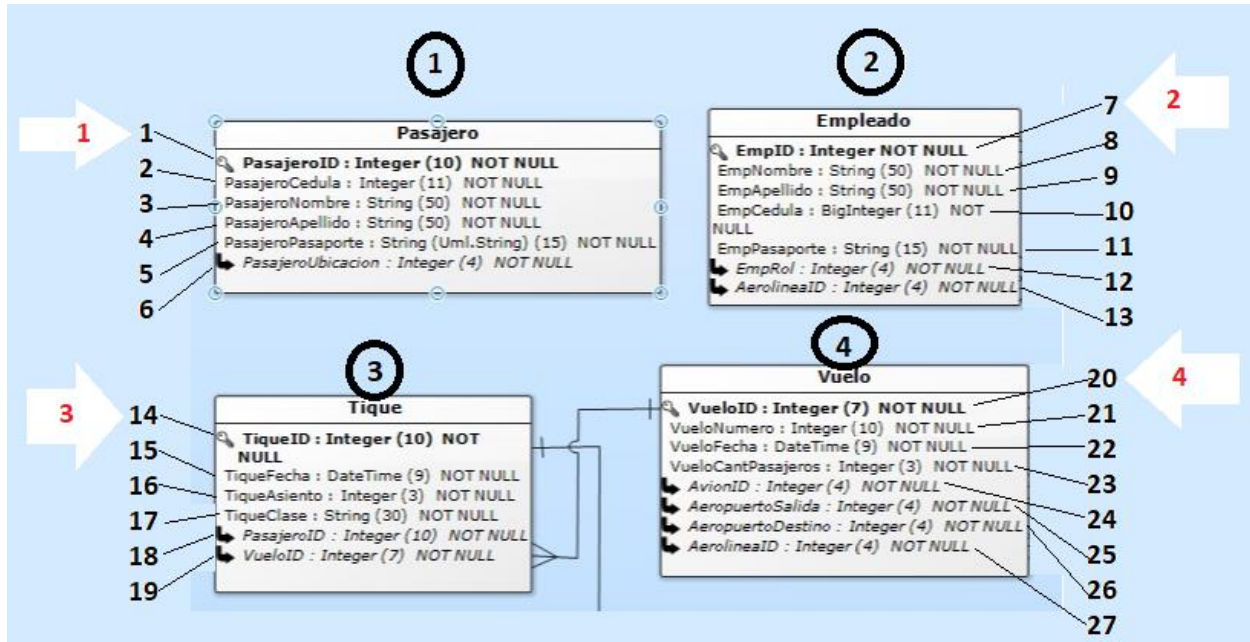
## Sistema de Vuelos Aéreos Con Gestor de Bases De Datos en C++

Miguel Arturo Sanchez Cuello CH7491

5,27,AvionID,Integer,4,FALSE  
5,28,AeropuertoSalida,Integer,4,FALSE  
5,29,AeropuertoDestino,Integer,4,FALSE  
5,30,AerolineaID,Integer,4,FALSE  
6,31,TripulacionID,Integer,7,FALSE  
6,32,VueloID,Integer,4,FALSE  
6,33,Emp1ID,Integer,4,FALSE  
6,34,Emp2ID,Integer,4,FALSE  
6,35,Emp3ID,Integer,4,FALSE  
7,36,AvionID,Integer,4,FALSE  
7,37,AvionMatricula,String,13,FALSE  
7,38,AeropuertoID,Integer,4,FALSE  
8,39,EquipajeID,Integer,4,FALSE  
8,40,EquipajePeso,Float,3,FALSE  
8,41,EquipajeEstado,String,100,FALSE  
8,42,PasajeroID,Integer,10,FALSE  
8,43,TiqueID,Integer,10,FALSE  
9,44,AerolineaID,Integer,4,FALSE  
9,45,AerolineaNombre,String,100,FALSE  
10,46,AeropuertoID,Integer,4,FALSE  
10,47,AeropuertoNombre,String,100,FALSE  
10,48,UbicacionID,Integer,4,FALSE  
11,49,UbicacionID,Integer,4,FALSE  
11,50,UbicacionNombre,String,100,FALSE  
11,51,PaisID,Integer,4,FALSE  
12,52,PaisID,Integer,4,FALSE  
12,53,PaisNombre,String,100,FALSE  
12,54,ProvincialID,Integer,4,FALSE  
13,55,ProvincialID,Integer,4,FALSE  
13,56,ProvinciaNombre,String,100,FALSE  
13,57,MunicipioID,Integer,4,FALSE  
14,58,MunicipioID,Integer,4,FALSE  
14,59,MunicipioNombre,String,100,FALSE  
14,60,SectorID,Integer,4,FALSE  
15,61,SectorID,Integer,4,FALSE  
15,62,SectorNombre,String,100,FALSE  
15,63,CalleID,Integer,4,FALSE  
16,64,CalleID,Integer,4,FALSE  
16,65,CalleNombre,String,100,FALSE  
16,66,CalleTipo,String,100,FALSE  
16,67,CasaID,Integer,4,FALSE  
17,68,CasaID,Integer,4,FALSE  
17,69,CasaNumero,String,7,FALSE

## Creando el archivo SysPrimaryKey.csv

Para crear el Archivo de llaves primarias del sistema se crea una tabla cuyas columnas serán PrimaryKeyID, TablaID y ColumnaID (si se prefiere se agrega otra que tenga el nombre del campo que es llave primaria, nosotros lo haremos en el archivo final). Como ejemplo presentamos lo siguiente:



Y la tabla resultante del ejemplo mostrado seria:

PrimaryKeyID	TablaID	ColumnaID	PKNombre
1	1	1	Esta columna no es obligatoria, pero puede ser útil para guiarnos al momento de programar
2	2	7	
3	3	14	
4	4	20	

Nota: como ya se tiene el archivo SysColumn organizado se entiende que la primera ocurrencia del Campo TablaID en el SysColumn es una llave foránea, trate siempre de mantener un orden para que se facilite el trabajo. En esta ocasión se recomienda hacer el proceso manual para que se familiarice con los conceptos y los procesos; pero si ha de tener un archivo de muchos registros cree su propio algoritmo que tome los datos por usted. Si es un sistema comercial se recomienda usar programas gestores dedicados a estos fines que lo hacen de forma automática.

Así pues, nuestro archivo SysColumn.csv resultante nos queda de la siguiente forma:

PrimaryKeyID,TablaID,ColumnaID,PKNombre

1,1,1,PasajeroID

2,2,7,EmpID

3,3,14,RolID

4,4,17,TiqueID

## Creando el archivo SysForeignKey.csv

Para crear el archivo de llaves foráneas primero se ha de colocar el ID para la clave foránea, luego se ha de tomar la TablaHija(esta es en la tabla en que la llave primaria aparece como foránea), luego se ha de tomar la ColumnaHija(Este es el numero de la columna que tiene asignada dicha clave foránea), luego se ha de tomar la TablaPadre(Esta es la tabla en que se define como clave primaria) y por ultimo tomaremos la ColumnaPadre(Este es el numero de la columna de la clave primaria en la tabla que es clave primaria).

Presentamos el siguiente ejemplo para esclarecer, en caso de dudas, mejor:

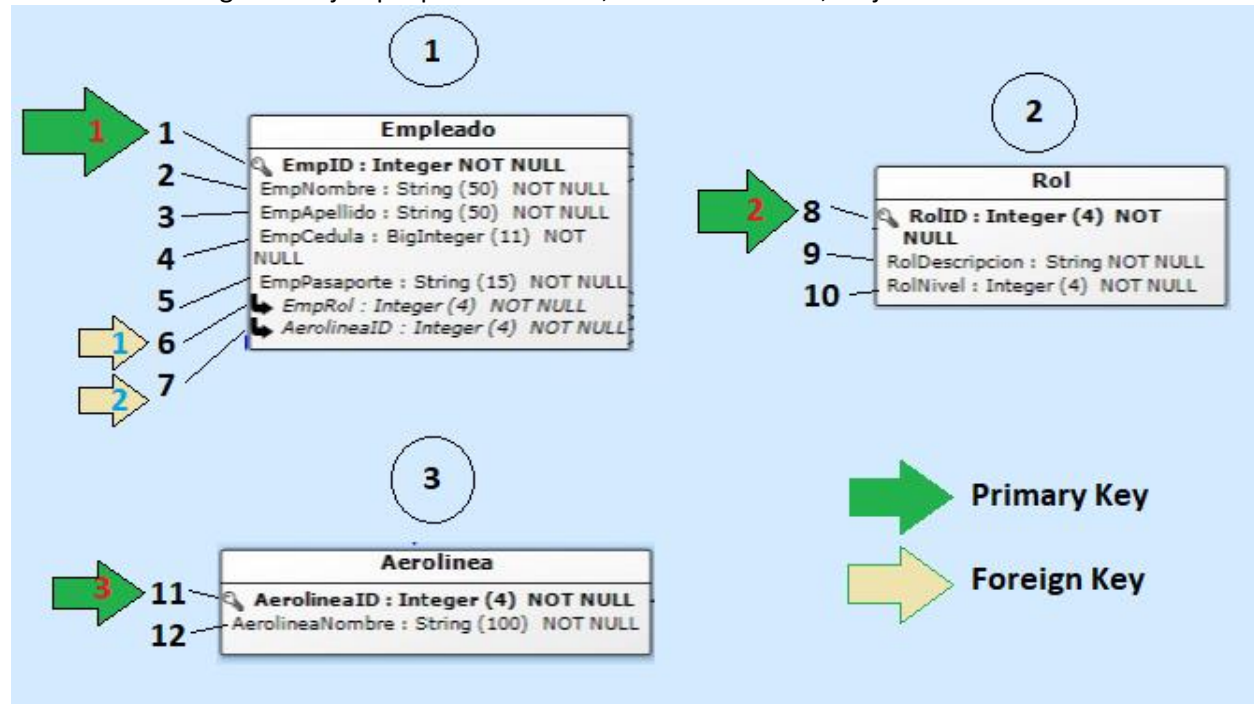


Imagen que describe 3 tablas con sus respectivas columnas, primary keys y foreignkey que usaremos para el ejemplo de la tabla siguiente.

ForeignKeyID	TablaHija	ColumnaHija	TablaPadre	ColumnaPadre	FKNombre
--------------	-----------	-------------	------------	--------------	----------

1	1	6	2	8	Esta columna no es obligatoria, pero puede ser útil para guiarnos al momento de programar
2	1	7	3	11	

Nota: Recuerde que esto es un ejemplo para que el estudiante se guíe al momento de preparar sus respectivos archivos.

Nuestro archivo SysForeignKey.csv nos queda de la siguiente forma:

ForeignKeyID,TablaHija,ColumnaHija,TablaPadre,ColumnaPadre,FKNombre

1,1,6,11,49,PasajeroUbicacion

2,2,12,3,14,EmpRol

3,2,13,9,44,AerolineaID

4,4,21,1,1,PasajeroID

5,4,22,5,23,VueloID

6,5,27,7,36,AvionID

7,5,28,10,46,AeroPuertoSalida

8,5,29,10,46,AeropuertoDestino

9,5,30,9,44,AerolineaID

10,6,32,5,23,VueloID

11,6,33,2,7,Emp1ID

12,6,34,2,7,Emp2ID

13,6,35,2,7,Emp3ID

14,7,38,10,46,AeropuertoID

15,8,42,1,1,PasajeroID

16,8,43,4,17,TiqueID

17,10,48,11,49,UbicacionID

18,11,51,12,52,PaisID

19,12,54,13,55,ProvincialID

20,13,57,14,58,MunicipioID

21,14,60,15,61,SectorID

22,15,63,16,64,CalleID

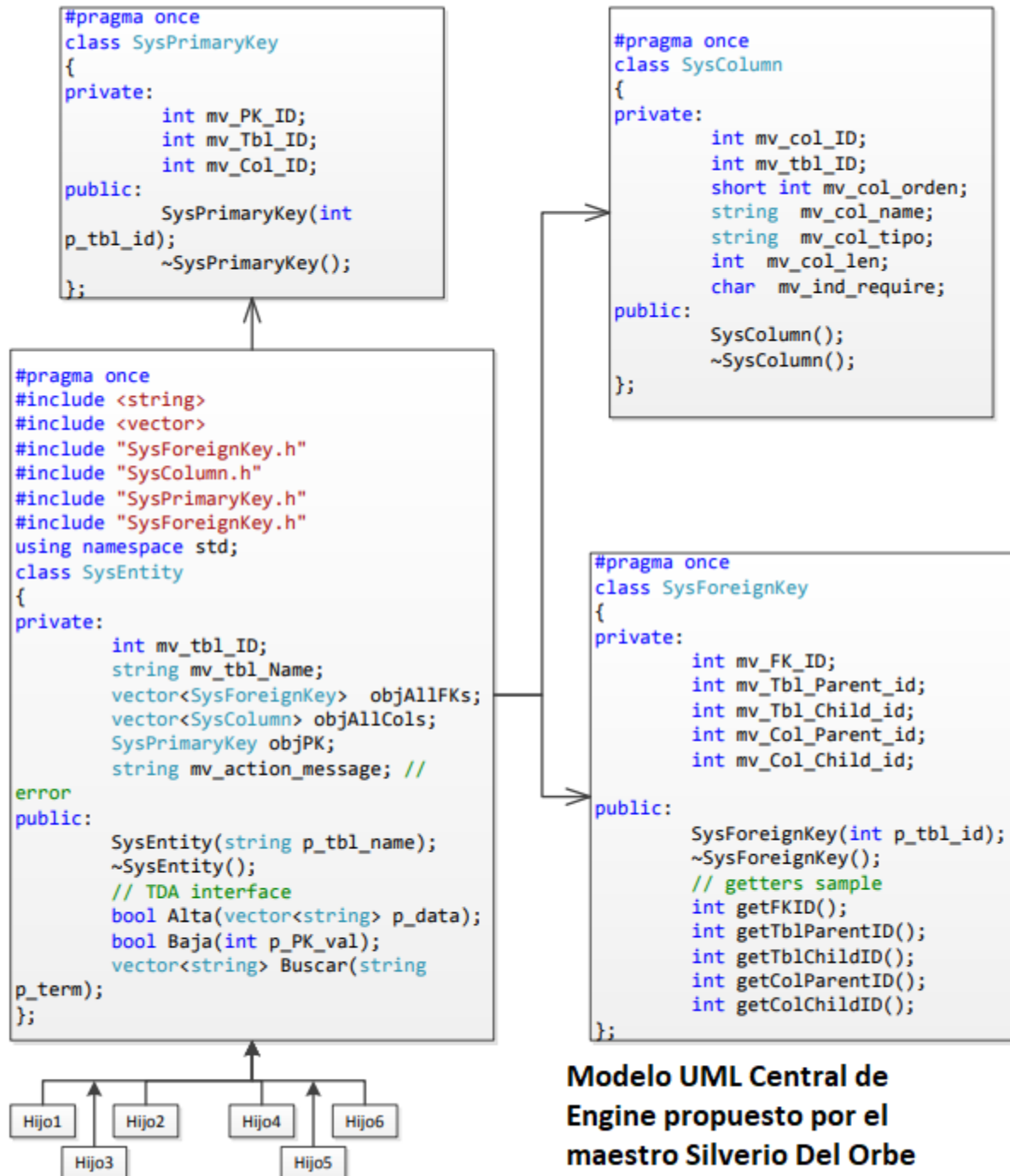
23,16,67,17,68,CasaID

Con estos 4 archivos principales procedemos a crear una versión reducida de un gestor de bases de datos que creará las estructuras de datos en memoria y permitirá la manipulación de estos (Alta, Baja y Búsqueda). Veamos esto en la siguiente sección.



## CREANDO EL GESTOR DE DATOS EN C++

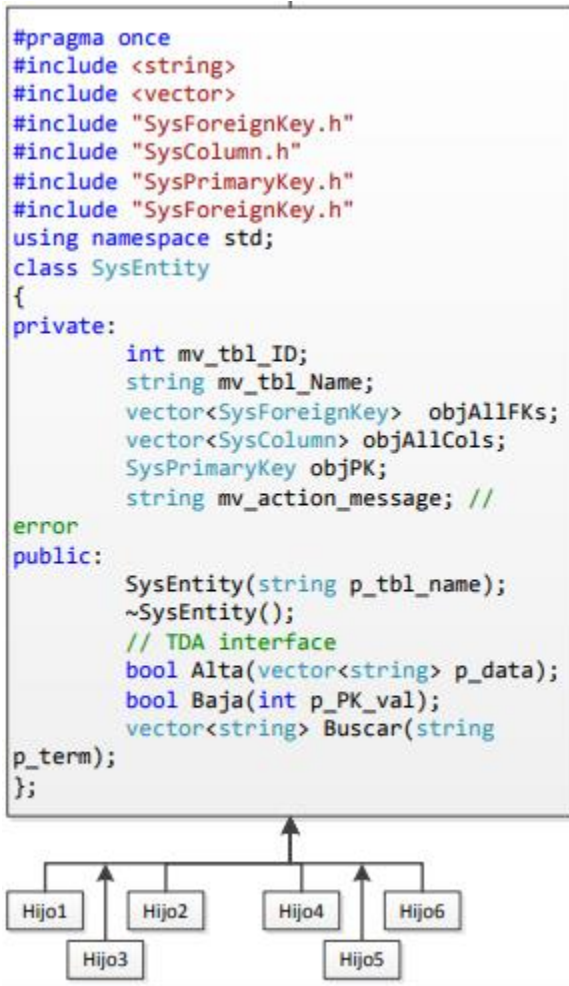
Antes de comenzar a codificar presentamos el Modelo UML central del Engine propuesto por el maestro Silverio Del Orbe. Este presenta las interfaces principales y a nosotros, los estudiantes nos corresponde escribir el código restante para que funcione.



Detalles de la funcionalidad del modelo en las siguientes páginas.



## Conociendo la interfaz SysEntity



En su presentación inicial tenemos la clase SysEntity que se encargara de poner todas las piezas juntas; es decir, el SysTable, el SysColumn, el SysPrimaryKey y el SysForeignKey (las 4 conforman nuestro SysCatalog) y crea la estructura de datos en memoria para ser manipulados.

Además, nuestras operaciones con los datos también serán manejadas por la entidad, como el motor del sistema aquí se hacen todas las verificaciones pertinentes para asegurar la integridad y la consistencia de los datos tomando en cuenta su independencia.

Nuestro boceto inicial de la clase empieza presentando el archivo de encabezado o header.

Ver en la siguiente pagina →→→

```

SysEntity.h* X SysPrimaryKey.cpp SysColumn.cpp SysEntity.cpp SysForeignKey.cpp TesterClas
SGBDMiguelSanchez (Global Scope)
1  /*
2  Miguel Sanchez
3  CH-7491
4  Asignatura Estructura de Datos
5  Semestre 2019-2
6  Universidad Autonoma de Santo Domingo UASD
7  para mas informacion visitar mi GitHub
8  https://github.com/MiguelSanchezCuello
9  me pueden contactar a traves del correo electronico
10 miguelsc10@hotmail.com
11 o a traves del sitio web
12 https://www.deolink.com
13 */
14
15
16 #pragma once
17 #include <string>
18 #include <vector>
19 #include "SysColumn.h"
20 #include "SysPrimaryKey.h"
21 #include "SysForeignKey.h"
22
23
24 using namespace std;
25
26 class SysEntity
27 {
28 public:
29     SysEntity();
30     SysEntity(string nombreTabla);
31     ~SysEntity();
32     bool TablaExiste(string nameTable);
33     bool ForeignKeyExiste(int tbl_id);
34     // verifica si un id existe en los registros de la Entidad.
35     // retorna el id del dato.
36     int datoExiste(int id);
37     // Verifica si un dato existe en los registros de la Entidad.
38     // retorna el id del dato.
39     int datoExiste(string dato);
40     SysColumn cols;

```

```

38     // retorna el id del dato.
39     int datoExiste(string dato);
40     SysColumn cols;
41     SysForeignKey forkeys;
42     void Alta(std::vector<std::string> data); // Enviar a Tabla.txt
43     vector<string> registros;
44
45
46 private:
47     string nombreEntidad;
48     int mv_tbl_ID; // Guarda el ID de la tabla
49     string mv_action_Msj;
50     vector<string> getSplit(const string &p_string, char p_delim);
51     void CargarColumnas();
52     // Almacena las columnas de la Entidad.
53     vector<SysColumn> objAllCols;
54     // Almacena todas las llaves foraneas de la Entidad.
55     vector<SysForeignKey> objAllFk;
56
57     vector<string> PrepararData(); // prepara la data a ingresar de alta
58     vector<string> dataAlta; // vector que contiene la data a la cual se dara alta.
59     void obtenerRegistros();// carga los registros al archivo
60     void verRegistros();
61     void CargarForeignKeys();
62     void CargarPrimaryKey();
63     // Representacion de la llave primaria de la Entidad.
64     SysPrimaryKey objPk;
65
66 };
67
68

```

Como podrá notar se agregaron más funciones o métodos para que fuera posible crear las estructuras y manipular los datos en memoria.

Recuerde que el código pronto estará disponible en mi GitHub:

<https://github.com/MiguelSanchezCuello>

El siguiente archivo para conocer es el archivo de encabezado para la clase SysColumn.

```
#pragma once
class SysColumn
{
private:
    int mv_col_ID;
    int mv_tbl_ID;
    short int mv_col_orden;
    string mv_col_name;
    string mv_col_tipo;
    int mv_col_len;
    char mv_ind_require;
public:
    SysColumn();
    ~SysColumn();
};
```

Esta clase se encarga de almacenar, como un objeto en memoria, los valores de la entidad correspondiente a las columnas, definiendo en si misma los atributos de las columnas de la entidad que se cargue en el sistema.

```
// Almacena las columnas de la Entidad.
vector<SysColumn> objAllCols;
```

Así pues, en la Entidad se tiene un vector de tipo SysColumn que almacenara, en cada índice, una columna con sus respectivos atributos que la describen y la hacen funcional.

En nuestro boceto inicial se presentan los atributos y métodos que fueron agregados a la clase SysColumn para que esta tuviera funcionalidad.

El archivo SysColumn.h se presenta en la siguiente pagina →→→

```

1  /*
2  Miguel Sanchez
3  CH-7491
4  Asignatura Estructura de Datos
5  Semestre 2019-2
6  Universidad Autonoma de Santo Domingo UASD
7  para mas informacion visitar mi GitHub
8  https://github.com/MiguelSanchezCuello
9  me pueden contactar a traves del correo electronico
10 miguelsc10@hotmail.com
11 o a traves del sitio web
12 https://www.deolink.com
13 */
14
15 #pragma once
16 #include <string>
17 using namespace std;
18 class SysColumn
19 {
20 public:
21     SysColumn();
22     ~SysColumn();
23     void SetTblID(int tbl_id);
24     int GetTblID();
25     void setColumnID(int col_id);
26     string GetColumnName();
27     void SetColumnName(string col_name);
28     string GetDataType();
29     void SetDataType(string DataType);
30     void SetLength(string length);
31     string GetLength();
32     void setRequerido(string requerido);
33     string GetRequerido();
34     int getColID();
35
36 private:
37     int mv_col_ID;
38     int mv_tbl_ID;
39     short int mv_col_orden;
40     string mv_col_name;
41     string mv_col_tipo;
42     int mv_col_len;
43     bool mv_ind_require;
44 };
  
```

107 %

El siguiente archivo para conocer es el archivo SysPrimaryKey.h

```
#pragma once
class SysPrimaryKey
{
private:
    int mv_PK_ID;
    int mv_Tbl_ID;
    int mv_Col_ID;
public:
    SysPrimaryKey(int
p_tbl_id);
    ~SysPrimaryKey();
};
```

Esta clase, genera un objeto PrimaryKey que maneja toda la data referente al campo, o los campos, que es la llave primaria de la Entidad que va a generar el sistema en memoria. (Podría ser mas de un primary key pero es una practica que no se recomienda, por eso limitaremos el alcance de este material a Entidades con una sola llave primaria).

```
63      // Representacion de la llave primaria de la Entidad.
64      SysPrimaryKey objPk;|
65
66  }
```

De igual forma que el objeto de columnas, en la Entidad existe también un objeto PrimaryKey.

En la página siguiente se muestra nuestro archivo SysPrimaryKey.h con las modificaciones iniciales para agregar funcionalidad ➡➡➡

```
TesterClass.h*  SysPrimaryKey.h*  SysColumn.h*  SysForeignKey.h*  SysEntity.h*
SGBDMiguelSanchez (Global Scope)

1  /*
2  Miguel Sanchez
3  CH-7491
4  Asignatura Estructura de Datos
5  Semestre 2019-2
6  Universidad Autonoma de Santo Domingo UASD
7  para mas informacion visitar mi GitHub
8  https://github.com/MiguelSanchezCuello
9  me pueden contactar a traves del correo electronico
10 miguelsc10@hotmail.com
11 o a traves del sitio web
12 https://www.deolink.com
13 */
14
15 #pragma once
16 class SysPrimaryKey
17 {
18
19 private:
20
21     //Primary key of table
22     int mv_PK_ID;
23     int mv_Tbl_ID;
24     int mv_Col_ID;
25
26 public:
27     void setPK_ID(int PK_ID);
28     void setTbl_ID(int col_tbl);
29     void setCol_ID(int mv_col);
30     int getTbl_ID();
31     SysPrimaryKey(int p_tbl_id);
32     SysPrimaryKey();
33     ~SysPrimaryKey();
34 };
35
36
```

Por ultimo y no menos importante conoceremos el archivo SysForeignKey.h

```
#pragma once
class SysForeignKey
{
private:
    int mv_FK_ID;
    int mv_Tbl_Parent_id;
    int mv_Tbl_Child_id;
    int mv_Col_Parent_id;
    int mv_Col_Child_id;

public:
    SysForeignKey(int p_tbl_id);
    ~SysForeignKey();
    // getters sample
    int getFKID();
    int getTblParentID();
    int getTblChildID();
    int getColParentID();
    int getColChildID();
};
```

Esta clase se encarga de crear objetos que contengan la información referente a una llave foránea de la entidad. Este es un archivo muy interesante y se recomienda al lector prestar cuidadosa atención al estudio del mismo.

```
54      // Almacena todas las llaves foraneas de la Entidad.
55      vector<SysForeignKey> objAllFk;
```

Como es de esperarse en la Entidad existe un vector que almacena todas las llaves foranes. Cuando la entidad se carga desde el Catalogo (los 4 archivos antes mencionado) se crea la estructura y maneja los datos en memoria.

Nuestro archivo SysForeignKey.h se presenta en la siguiente pagina →→→



```

TesterClass.h  SysPrimaryKey.h  SysColumn.h  SysForeignKey.h  Sy
SGBDMiguelSanchez
1  /*
2  Miguel Sanchez
3  CH-7491
4  Asignatura Estructura de Datos
5  Semestre 2019-2
6  Universidad Autonoma de Santo Domingo UASD
7  para mas informacion visitar mi GitHub
8  https://github.com/MiguelSanchezCuello
9  me pueden contactar a traves del correo electronico
10 miguelsc10@hotmail.com
11 o a traves del sitio web
12 https://www.deolink.com
13 */
14
15 #pragma once
16 #include <string>
17
18 using namespace std;
19 class SysForeignKey
20 {
21 private:
22     int mv_FK_ID;
23     int mv_Tbl_Parent_id;
24     int mv_Tbl_Child_id;
25     int mv_Col_Parent_id;
26     int mv_Col_Child_id;
27     string mv_FK_tbl_name;
28
29 public:
30     SysForeignKey();
31     SysForeignKey(int table_id);
32     int getFKID();
33     int getTblParentID();
34     int getTblChildID();
35     int getColParentID();
36     int getColChildID();
37     string getFKTblName();
38     void setFKID(int fk_id);
39     void setTblParentID(int id);
40     void setTblChildID(int id);
41     void setColParentID(int id);
42     void setColChildID(int id);
43     void setFKTblName(string tbl_name);
44     ~SysForeignKey();
45 };
46
47

```

# Funcionamiento del Gestor y Ejemplos

Ahora veamos el funcionamiento en acción de una Entidad.

Si cargamos una entidad con el nombre de la tabla →

```
int main() {
    //cout << "Hola mundo";
    SysEntity entidad("Pasajero");
}
```

Esta llamara a su constructor →

```
using namespace std;
// Constructor que recibe el nombre de la tabla para crear la Entidad.
SysEntity::SysEntity(string nombreTabla)
{
    // Lo primero que debe hacer es verificar si la tabla existe.
    if (TablaExiste(nombreTabla))
    {
        // Si existe carga los atributos de la Entidad.
        CargarColumnas();
        CargarPrimaryKey();

        if (ForeignKeyExiste(mv_tbl_ID)) {
            CargarForeignKeys(); // Carga las llaves Foraneas de la Entidad.
        }
    }
}
```

En dicho constructor, la primera función en ejecutarse es la que verifica si la tabla existe

```
// Metodo que verifica la existencia de la tabla.
bool SysEntity::TablaExiste(string nombreTabla)
{
    mv_action_Msj = "La tabla no existe"; // Hasta ahora la variable no se esta utilizando BORRAR MAS TARDE
    ifstream reads("../SGBDMiguelSanchez\\SysFiles\\SysTable.csv", ios::in);
    string temp;
    //Para leer el encabezado del archivo(Que no es la data)
    getline(reads, temp);
    ifstream sublector(temp); //A este punto la variable Temp tiene la primera linea.
}
```

Dicha función verifica en el archivo de sistema SysTable.csv que creamos con anterioridad y si encuentra la tabla retorna verdadero y continua con el método CargarColumnas →→

```
// Metodo que carga todas las columnas de la entidad.
void SysEntity::CargarColumnas()
{
    ifstream reader("../SGBDMiguelSanchez\\SysFiles\\SysColumn.csv", ios::in);
    string temp;
    //Read the headders of the files
}
```

Dicha función itera a través del archivo SysColumn para obtener todas las columnas correspondientes a la tabla cuya existencia ha sido comprobada.

Y cada columna que sea cargada es almacenada en el vector definido **objAllCols** ➔

```

114         if (this->mv_tbl_ID == cols.GetTblID())
115         {
116             objAllCols.push_back(cols);
117         }
    
```

La data que se va almacenando es precisamente la que se encuentra en el archivo SysColumn donde se indica el id, el nombre, el tipo de dato, etc... puede regresar a la sección de definición del catalogo de sistemas para retroalimentación.

Luego se ejecuta el método CargarPrimaryKey() ➔

```

void SysEntity::CargarPrimaryKey()
{
    ifstream reader("../SGBDMiguelSanchez\\SysFiles\\SysPrimaryKey.csv", ios::in);
    string temp;
    //Read the headders of the files
    ifstream subLector(temp); //read actual line
    getline(reader, temp);
    string auxiliar;
    //Read of files
    while (reader)
    {
        reader >> temp;
        vector<string> aux;
        aux = getSplit(temp, ',');

        objPk.setTbl_ID(stoi(aux[1]));

        if (this->mv_tbl_ID == objPk.getTbl_ID())
        {
            objPk.setPK_ID(stoi(aux[0]));
            objPk.setCol_ID(stoi(aux[2]));
            break;
        }
    }
    subLector.close();
} // Fin del metodo que carga la llave primaria.
    
```

Este método carga desde el Catálogo, toda la información referente a la llave primaria de la clase.

Luego ejecutaremos el método para cargar las llaves foranes de la entidad, pero como no todas las Entidades podrían tener claves foranes lo hacemos a través de una condicional.

```

if (ForeignKeyExiste(mv_tbl_ID)) {
    CargarForeignKeys(); // Carga las llaves Foraneas de la Entidad.
}

```

Esta es la función mas interesante de todo el proyecto, puesto que lee el Catalogo y toma del archivo ForeignKeys.csv la data donde La entidad aparece como TablaHija, si la Entidad se encuentra como hija en al menos una tabla esto quiere decir que la entidad posee al menos una clave foránea, y mientras mas concurrencias de hija tenga en ese archivo más claves foráneas tendrá ➔

```

153 // Metodo que verifica la existencia de llaves foraneas en la tabla.
154 bool SysEntity::ForeignKeyExiste(int tbl_id)
155 {
156     mv_action_Msj = "Esta tabla no tiene llaves foraneas."; // Hasta ahora la variable no se esta utilizando BORRAR MAS TARDE
157     ifstream reads("../SGBDMiguelSanchez\\SysFiles\\SysForeignKey.csv", ios::in);
158     string temp;
159     //Para leer el encabezado del archivo(Que no es la data)
160     getline(reads, temp);
161     ifstream subLector(temp); //A este punto la variable Temp tiene la primera linea.
162     vector<string> allForeignKeys;
163
164     // Lee desde la segunda linea del archivo, separa los campos y va agregando
165     // al vector todos las llaves foraneas (allForeignKeys).
166     while (reads) {
167         reads >> temp;
168         vector<string> aux;
169         aux = getSplit(temp, ',');
170         allForeignKeys.push_back(aux[1]);
171         // Recordemos que en cada linea del archivo
172         // 0 = FKID, 1 = TablaHija, 2 = ColHija, 3= TablaPadre, 4= ColPadre, 5= FKNombre.
173         // Tomamos el campo tabla hija para comparar y si es hija en alguna otra tabla significa
174         // que el primarykey de la otra tabla esta en esta tabla como foreignkey.
175     }
176

```

Así pues, una vez verificada la existencia de claves foraneas el programa agrega, usando la función **CargarForeignKeys()**, datos del registro en SysForeignKey como datos que pasan a ser parte del vector allForeignKeys perteneciente a la entidad.

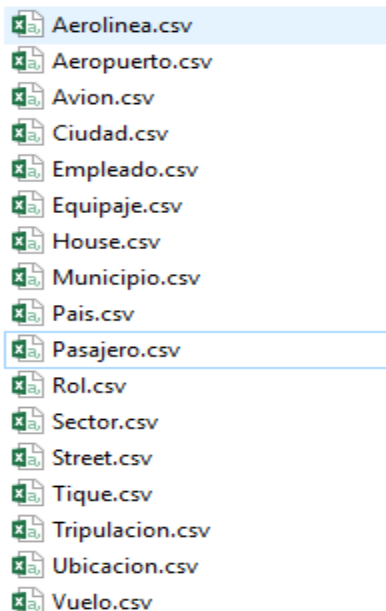
```

260 // Metodo que carga todas las llaves foraneas de la entidad.
261 void SysEntity::CargarForeignKeys() {
262
263     ifstream reader("../SGBDMiguelSanchez\\SysFiles\\SysForeignKey.csv", ios::in);
264     string temp;
265     //Read the headders of the files
266     ifstream subLector(temp); //read actual line
267     getline(reader, temp);
268     string auxiliar;
269     //Read of files
270     while (reader)
271     {
272         reader >> temp;
273         vector<string> aux;
274         aux = getSplit(temp, ',');
275
276         forkeys.setFKID(stoi(aux[0]));
277         forkeys.setTblChildID(stoi(aux[1]));
278         forkeys.setColChildID(stoi(aux[2]));
279         forkeys.setTblParentID(stoi(aux[3]));
280         forkeys.setColParentID(stoi(aux[4]));
281         forkeys.setFKTblName(aux[5]);
282
283         // Recordemos que en cada linea del archivo
284         // 0 = FKID, 1 = TablaHija, 2 = ColHija, 3= TablaPadre, 4= ColPadre, 5= FKNombre.
285         // Tomamos el campo hija para comparar y si es hija en alguna otra tabla significa
286         // que el primarykey de la otra tabla esta en esta tabla como foreignkey.
287
288         if (this->mv_tbl_ID == forkeys.getTblChildID())
289         {
290             objAllFk.push_back(forkeys);
291         }
292         //break;
293     }
294     subLector.close();
295 } // Fin del metodo que carga las llaves foraneas.

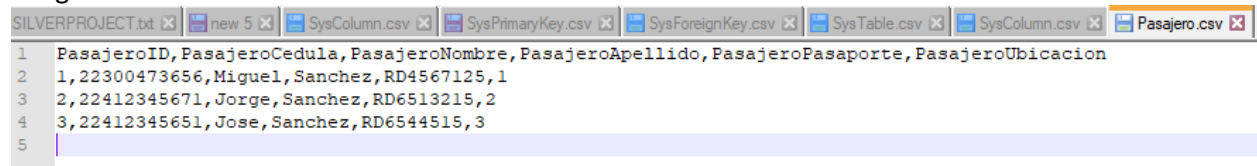
```

Hay que recordar que, aunque la Entidad, o Entidades, es cargada en memoria esto no significa que la data es manipulada directamente al cargar la entidad. Nuestra entidad es quien regirá las normas para la manipulación y el acceso a los datos.

Todo nuestro sistema se maneja a través de archivos



Cada archivo representa, tiene, un conjunto de registros de una tabla con sus respectivos campos. Así por ejemplo la tabla Pasajero.csv contiene registros de pasajeros como se muestra en la siguiente imagen →



PasajeroID	PasajeroCedula	PasajeroNombre	PasajeroApellido	PasajeroPasaporte	PasajeroUbicacion
1	22300473656	Miguel	Sanchez	RD4567125	1
2	22412345671	Jorge	Sanchez	RD6513215	2
3	22412345651	Jose	Sanchez	RD6544515	3

Para hacer altas en el sistema se utiliza la función Alta que se encuentra en la Entidad.

```
41 // System entity for keys,  
42 void Alta(std::vector<std::string> data); // Enviar a Tabla.csv  
43
```

Esta se auxilia de la función PrepararData y de el atributo dataAlta.

```
57 vector<string> PrepararData(); // prepara la data a ingresar de alta  
58 vector<string> dataAlta; // vector que contiene la data a la cual se dara alta.
```

El método PrepararData() se encarga de hacer todas las validaciones correspondientes a la data ingresada y también verifica si hay llaves foráneas asociadas a la data introducida. Una vez completadas las verificaciones la data preparada es enviada al vector dataAlta para ser ingresado a la tabla correspondiente.

```
298 // Metodo para dar alta al vector de data  
299 void SysEntity::Alta(vector<string> data)  
300 {  
301     ofstream archivo;  
302     string fileLocation = "..\\SGBDMiguelSanchez\\SysFiles\\Tablas\\";  
303     archivo.open(fileLocation + this->nombreEntidad + ".csv", std::ios::out | std::ios::app);  
304  
305     string auxString;  
306     if (!archivo.fail()) {  
307         for (int i = 0; i < data.size(); ++i) {  
308             auxString += data[i] + ",";  
309         }  
310         auxString.pop_back();  
311         archivo << auxString << "\n";  
312     }  
313     archivo.close();  
314 }
```

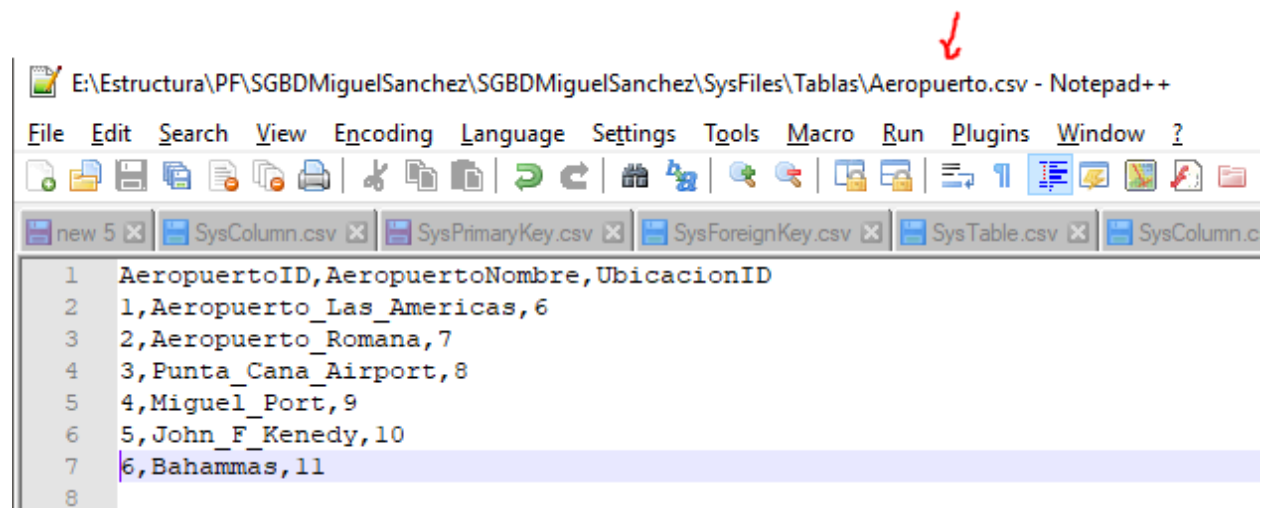
Esta función de alta recibe la data ya preparada de una forma genérica toma el nombre de la entidad y busca el archivo con ese nombre y almacena el registro. Todas las verificaciones se hacen en la función PrepararData.

Por ejemplo, en la tabla Aeropuerto se hace un alta y se verifican los datos a través de la Entidad

```
E:\Estructura\PF\SGBDMiguelSanchez\Debug\SGBDMiguelSanchez.exe
Bienvenido al sistema de vuelos aereos
En esta fase de pruebas vamos a dar un alta a Aeropuerto:
Ingrese el ID del aeropuerto:
6
Ingrese el Nombre del aeropuerto:
Bahamas
Ingrese el ID de la Ubicacion del aeropuerto:
11
ingresando los datos en la tabla de registros de la entidad.

AHORA SE MUESTRAN LOS REGISTROS ALMACENADOS
AeropuertoID    AeropuertoNombre    UbicacionID
1      Aeropuerto_Las_Americas  6
2      Aeropuerto_Romana        7
3      Punta_Cana_Airport      8
4      Miguel_Port              9
5      John_F_Kenedy          10
6      Bahamas                11
```

La data queda registrada en la tabla



```
E:\Estructura\PF\SGBDMiguelSanchez\SGBDMiguelSanchez\SysFiles\Tablas\Aeropuerto.csv - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 5 x SysColumn.csv x SysPrimaryKey.csv x SysForeignKey.csv x SysTable.csv x SysColumn.c
1  AeropuertoID,AeropuertoNombre,UbicacionID
2  1,Aeropuerto_Las_Americas,6
3  2,Aeropuerto_Romana,7
4  3,Punta_Cana_Airport,8
5  4,Miguel_Port,9
6  5,John_F_Kenedy,10
7  6,Bahamas,11
8
```

**Todas las operaciones de gestión y manipulación de datos se realizan desde la Entidad.**

```
using namespace std;
```

```
int main() {
    //cout << "Hola mundo";
    vector<string> dataTest;
    string data;
    SysEntity entidad("Aeropuerto");

    cout << "Bienvenido al sistema de vuelos aereos" << endl;
    cout << "En esta fase de pruebas vamos a dar un alta a Aeropuerto: " << endl;
    cout << "Ingrese el ID del aeropuerto: " << endl;;
    getline(cin, data);
    dataTest.push_back(data);
    cout << "Ingrese el Nombre del aeropuerto: "<< endl;
    getline(cin, data);
    dataTest.push_back(data);
    cout << "Ingrese el ID de la Ubicacion del aeropuerto: " << endl;
    getline(cin, data);
    dataTest.push_back(data);

    cout << "ingresando los datos en la tabla de registros de la entidad." << endl;
    entidad.Alta(dataTest);

    cout << endl << "AHORA SE MUESTRAN LOS REGISTROS ALMACENADOS" << endl;
    entidad.verRegistros();
}
```

En las siguientes paginas mostraremos mas ejemplos de altas que van directo a las tablas de la entidad correspondiente. → →

Alta de Pasajeros y visualización de registros a través de la entidad

```
E:\Estructura\PF\SGBDMiguelSanchez\Debug\SGBDMiguelSanchez.exe
Bienvenido al sistema de vuelos aereos
En esta fase de pruebas vamos a dar un alta a Pasajero:
Ingrese el ID del Pasajero:
5
Ingrese la Cedula:
555441231
Ingrese el Nombre:
Silvestre
Ingrese el Apellido:
Stalone
Ingrese el Pasaporte:
US55123
Ingrese el ID de la Ubicacion(direccion) del Pasajero:
5
ingresando los datos en la tabla de registros de la entidad.

AHORA SE MUESTRAN LOS REGISTROS ALMACENADOS
```

PasajeroID	PasajeroCedula	PasajeroNombre	PasajeroApellido	PasajeroPasaporte	PasajeroUbicacion
1	22300473656	Miguel	Sanchez	RD4567125	1
2	22412345671	Jorge	Sanchez	RD6513215	2
3	22412345651	Jose	Sanchez	RD6544515	3
4	0014735546	Antony	Santos	RD45132	4
5	555441231	Silvestre	Stalone	US55123	5



```

E:\Estructura\PF\SGBDMiguelSanchez\SGBDMiguelSanchez\SysFiles\Tablas\Pasajero.csv - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 5 x SysColumn.csv x SysPrimaryKey.csv x SysForeignKey.csv x SysTable.csv x SysColumn
1 PasajeroID,PasajeroCedula,PasajeroNombre,PasajeroApellido,PasajeroPasajero
2 1,22300473656,Miguel,Sanchez,RD4567125,1
3 2,22412345671,Jorge,Sanchez,RD6513215,2
4 3,22412345651,Jose,Sanchez,RD6544515,3
5 4,0014735546,Antony,Santos,RD45132,4
6 5,555441231,Silvestre,Stalone,US55123,5
7
    
```

Dando altas de Tiques y mostrando los registros a través de la Entidad

```

E:\Estructura\PF\SGBDMiguelSanchez\Debug\SGBDMiguelSanchez.exe
Bienvenido al sistema de vuelos aereos
En esta fase de pruebas vamos a dar un alta a Tique:
Ingrese el ID del Tique:
1
Ingrese La fecha del tique:
12/09/2019
Ingrese el numero de asiento:
15b
La Clase:
A
Ingrese el ID del pasajero:
1
Ingrese el ID del vuelo:
1
Ingresando los datos en la tabla de registros de la entidad.
AHORA SE MUESTRAN LOS REGISTROS ALMACENADOS
TiqueID TiqueFecha TiqueAsiento TiqueClase PasajeroID VueloID
1 12/09/2019 15b A 1 1
    
```

```
E:\Estructura\PF\SGBDMiguelSanchez\Debug\SGBDMiguelSanchez.exe
Bienvenido al sistema de vuelos aereos
En esta fase de pruebas vamos a dar un alta a Tique:
Ingrese el ID del Tique:
2
Ingrese La fecha del tique:
12/09/2019
Ingrese el numero de asiento:
25C
La Clase:
B
Ingrese el ID del pasajero:
2
Ingrese el ID del vuelo:
1
ingresando los datos en la tabla de registros de la entidad.

AHORA SE MUESTRAN LOS REGISTROS ALMACENADOS
TiqueID TiqueFecha TiqueAsiento TiqueClase PasajeroID VueloID
1 12/09/2019 15b A 1 1
2 12/09/2019 25C B 2 1
```

```
E:\Estructura\PF\SGBDMiguelSanchez\Debug\SGBDMiguelSanchez.exe
Bienvenido al sistema de vuelos aereos
En esta fase de pruebas vamos a dar un alta a Tique:
Ingrese el ID del Tique:
3
Ingrese La fecha del tique:
12/03/2019
Ingrese el numero de asiento:
11A
La Clase:
A
Ingrese el ID del pasajero:
4
Ingrese el ID del vuelo:
1
ingresando los datos en la tabla de registros de la entidad.

AHORA SE MUESTRAN LOS REGISTROS ALMACENADOS
TiqueID TiqueFecha TiqueAsiento TiqueClase PasajeroID VueloID
1 12/09/2019 15b A 1 1
2 12/09/2019 25C B 2 1
3 12/03/2019 11A A 4 1
```

Así pues, se puede utilizar la Entidad como el **Sistema Gestor de la Base de Datos**, donde el método PrepararData hará todas las verificaciones pertinentes antes de ingresar la data en los registros. Este método utiliza las funciones datoExiste sobrecargadas para verificar en la clave foránea que el registro existe y en caso de que no exista proporciona la opción de agregar la data, si no se quiere agregar el registro se cancela la operación de Alta (dentro de la cual se prepara la data).

Recordemos que este es un proyecto en desarrollo, aun hay muchas funciones y datos e incluso hasta el mismo análisis y diagrama necesitan tener una segunda revisión, la mayoría de los datos y casos se simplificaron para poder tener un material presentable dentro del tiempo acordado. Si desea dar seguimiento al proyecto recuerde que puede acceder a mi GitHub donde se publicara la información de los avances una vez terminado el semestre.

Información de contacto:

```
/*  
Miguel Sanchez  
CH-7491  
Asignatura Estructura de Datos  
Semestre 2019-2  
Universidad Autonoma de Santo Domingo UASD  
para mas informacion visitar mi GitHub  
https://github.com/MiguelSanchezCuello  
me pueden contactar a traves del correo electronico  
miguelscl0@hotmail.com  
o a traves del sitio web  
https://www.deolink.com  
*/
```

## Sobre el autor:

Nombre: Miguel Arturo Sanchez Cuello.

Matricula: CH-7491.

Materia: Estructura de Datos.

Profesor Asignado: Silverio del Orbe.

Lema del autor: Siempre hay espacio para mejorar.



El autor es estudiante de la carrera Licenciatura en Informática de la Universidad Autónoma de Santo Domingo, UASD. Consta con una vasta experiencia laboral en el ámbito del servicio al cliente bilingüe donde desempeñó funciones de Analista de Calidad por 8 años, también desempeñó funciones de Supervisor y Entrenador y Moderador de Calibraciones, entre otras varias funciones administrativas y gerenciales. Apasionado por la forma correcta de hacer las cosas ha presentado este material, no como una forma definitiva, sino como un boceto que deja abierta la puerta para todo aquel que quiera adentrarse en la investigación del funcionamiento interno de un sistema gestor de base de datos y como se utilizan las Estructuras de Datos para que este haga su trabajo.