Profesor
L. en C.C. Erick Orlando Matla Cruz
Ayudantes
L. en C.C. Efraín Hipólito Chamú
L. en C.C. Anahí Quiroz Jiménez
L. en C.C. Karen Monserrat Zavala Correa
Grupo: 9119

Práctica 05

Lenguaje de Definición de Datos (DDL) e Integridad

1. Objetivo General

Construir el esquema de una base de datos utilizando el Lenguaje Estructurado de Consultas (SQL por sus siglas en inglés). Así como conocer los diferentes tipos de integridad inherentes a las bases de datos.

2. Objetivos Secundarios

- Conocer la sintaxis y los parámetros del Lenguaje de Definición de Datos (DDL por sus siglas en inglés).
- Transformar un diagrama de Clases en el esquema de una base de datos.
- Optimizar la congruencia y consistencia de los datos que se almacenarán en la base de datos.

3. Introducción

Las bases de datos relacionales están conformadas por una colección de tablas o *relaciones*. Para poder construir estas tablas o relaciones se necesita primero crear a la base que las contendrá. La base de datos junto con las tablas que la componen serán creadas y administradas con ayuda de un SMBD, en este caso PostgreSQL y un lenguaje de programación de propósito específico, SQL.

En esta práctica se trabajará por una parte con SQL, uno de los primeros lenguajes comerciales y fue propuesto por Donald Chamberlin y Raymond Boyce. Por otra parte el modelo relacional creado por Edgar F. Codd y que el mismo describió como "Un modelo relacional de datos para grandes bancos de datos compartidos". A pesar de no apegarse íntegramente al modelo relacional tal como lo describe Codd , SQL se convirtió en el lenguaje de bases de datos más utilizado.

SQL se convirtió en un estándar de la American National Standards Institute (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987. Desde entonces, el estándar se ha mejorado varias veces con características añadidas pero aún no es completamente transferible entre sistemas de bases de datos diferentes, es decir, cada SMBD tiene una combinación específica del código, ya que los diferentes fabricantes no siguen estrictamente el estándar al añadir extensiones.

SQL fue originalmente basado en el Álgebra Relacional y el Cálculo Relacional de tuplas. Está constituido por el siguiente conjunto de sublenguajes:

- Lenguaje de definición de datos (DDL)
- Lenguaje de manipulación de datos (DML)
- Lenguaje de control (DCL)
- Lenguaje de transacciones (TCL)

El alcance de SQL incluye aspectos como la inserción de datos, consulta, actualización y supresión, creación y modificación de esquemas, y el control de acceso a datos. Para ésta práctica revisaremos la parte correspondiente a la definición de datos, es decir el DDL.

3.1. Creación de la base de datos

Para crear una base de datos sobre la cual construiremos las tablas, se necesitan algunas palabras reservadas de SQL. En la Figura 5.1 se presenta la sintaxis utilizada para la creación de una base de datos. Para fines didácticos, y a partir de esta práctica, se utilizarán mayúsculas al escribir aquellas palabras reservadas pertenecientes a SQL.

```
CREATE DATABASE mi_primera_base;
```

Figura 5.1 - Ejemplo básico de la sintaxis para creación de una base de datos.

Esta sintaxis está compuesta de diferentes partes:

1. CREATE DATABASE
   Palabra reservada que se utiliza cada vez que se necesite crear una base de datos.
2. mi_primera_base
   El nombre de la base de datos que se desea crear, es elegido por el diseñador de la base de datos. Toda nueva base que se cree dentro de un SMBD tendrá entonces un nombre diferente a los ya existentes.
3. ;
   Indica la finalización de la instrucción.

Lo anterior corresponde a la sintaxis básica y mínima necesaria para crear una base de datos. Sin embargo el DDL permite definir características avanzadas sobre una base de datos, tal como se muestra en la Figura 5.2.

```
CREATE DATABASE mi_primera_BD

WITH
        OWNER = postgres
        ENCODING = 'UTF8'
        TABLESPACE = pg_default
        CONNECTION LIMIT = -1
        ;
```

Figura 5.2 - Ejemplo avanzado de la sintaxis para creación de una base de datos.

La sintaxis anterior se explica a continuación:

1. OWNER
   Indica el nombre del usuario que será el propietario de la base de datos creada. En caso de no definirla o que se escriba la opción DEFAULT, el SMBD asignará como OWNER al usuario que ejecuta el comando.
2. ENCODING
   Conjunto de caracteres para utilizar en la base de datos creada. Se puede especificar una cadena constante, por ejemplo 'SQL_ASCII', un número entero de codificación o la opción DEFAULT para utilizar la codificación por defecto
   Otro ejemplo de codificación es: ISO 8859-15. Puede usarse para representar el alfabeto y otros caracteres importantes para almacenar textos en inglés, francés, alemán, español y portugués, entre otros idiomas de Europa occidental.
3. TABLESPACE
   Indica la ruta del equipo donde se guardarán los archivos de información de las bases de datos.
4. CONNECTION LIMIT
   Permite establecer cuántas conexiones simultáneas se pueden hacer a la base de datos creada. El número -1, escrito por defecto, significa que no hay límite.

Hay que tomar en cuenta que si no se personalizan los parámetros de las opciones de la base de datos, el SMBD aplicará la configuración por defecto.

3.2. **Creación de tablas**

Una vez que la base de datos ha sido creada, se procederá a construir las tablas necesarias, y una vez más, a través de SQL.

La construcción de las tablas requiere de comandos o palabras reservadas, además de variables y constantes en un determinado orden para que el SMBD lo ejecute correctamente. En la Figura 5.3 se muestra la sintaxis básica para crear una tabla.

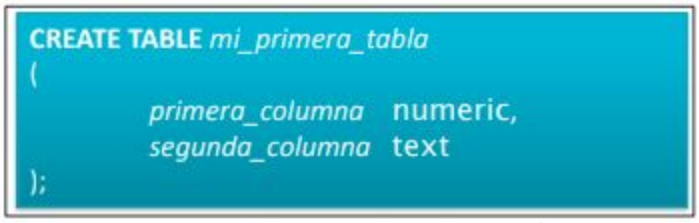

Figura 5.3 - Ejemplo básico de la sintaxis para creación de tablas.

Las partes que componen la sintaxis anterior se describen a continuación:

1. CREATE TABLE
   Palabra reservada que se utilizará cada vez que se necesite crear una nueva tabla.
2. mi_primera_tabla
   El nombre de la tabla que se quiere crear, es elegido por el diseñador de la base de datos. Toda nueva tabla que se cree dentro de un SMBD tendrá entonces un nombre diferente a los ya existentes.
3. (
   El símbolo de apertura de paréntesis, después del nombre de la tabla, sirve para indicar que comenzará con la descripción de las columnas de esa tabla.
4. primera_columna
   El nombre de la columna deberá describir el dato que se almacenará en ella. Este nombre lo elige el diseñador de la base de datos y cada columna deberá tener un nombre único dentro de la tabla, no así en toda la base de datos. Se aconseja evitar repetir nombres de columnas dentro de la misma base de datos. De ser necesario se recomienda utilizar la siguiente notación: *nombreColumna_nombreTabla*.
5. numeric
   Indica la naturaleza del dato que se va a almacenar, es decir, el tipo del dato. PostgreSQL soporta varios tipos de datos, como se presentó durante el *Modelado con Diagramas de Clases* (Práctica 03).
6. ,
   La coma, actúa como separador de columnas, indica que la descripción del nombre, tipo y otras características de la columna ha finalizado. Después de la coma se puede introducir la descripción de otra columna siguiendo el mismo procedimiento de los pasos 4 y 5.
7.)

El símbolo de cierre de paréntesis, después de la descripción de la última columna que integrará la tabla, indica al SMBD que ha terminado la definición de tabla.

8. ;

Indica la finalización de la instrucción.

SQL tiene la capacidad de personalizar ciertas características de las tablas para cumplir con las necesidades del diseñador. En la Figura 5.4 se muestra un ejemplo donde se declara un número mayor de columnas con diferentes tipos de datos.

```
CREATE TABLE   cliente
(
        id_unico SERIAL,
        nombre_cliente   VARCHAR(40),
        ap_paterno_cliente   VARCHAR(40),
        ap_materno_cliente   VARCHAR(40),
        clasificacion_de_cliente   BOOLEAN,
        status_activacion   BOOLEAN,
        fecha_ingreso   DATE
);
```

Figura 5.4 - Ejemplo avanzado de la sintaxis para creación de tablas.

La característica de *character varying (40)* expresa el tipo de la columna y la longitud máxima del dato. Es decir, que en esa columna tendremos la posibilidad de guardar cadenas de caracteres con una longitud variable pero máxima de 40 posiciones. Así como ésta restricción, existen algunos tipos de datos que requieren ciertos parámetros para determinar con mayor exactitud el tipo de dato que recibirán. La lista de los tipos de datos y sus características se definió durante el *Modelado con Diagramas de Clases* (Práctica 03).

4. Ejemplo de la pizzería

Recordando lo trabajado en *Modelado con Diagramas de Clases* (Práctica 03), se obtuvo el diagrama de Clases para la Pizzería. Ver Figura 5.5.

Profesor
L. en C.C. Erick Orlando Matla Cruz
Ayudantes
L. en C.C. Efraín Hipólito Chamú
L. en C.C. Anahí Quiroz Jiménez
L. en C.C. Karen Monserrat Zavala Correa
Grupo: 9119

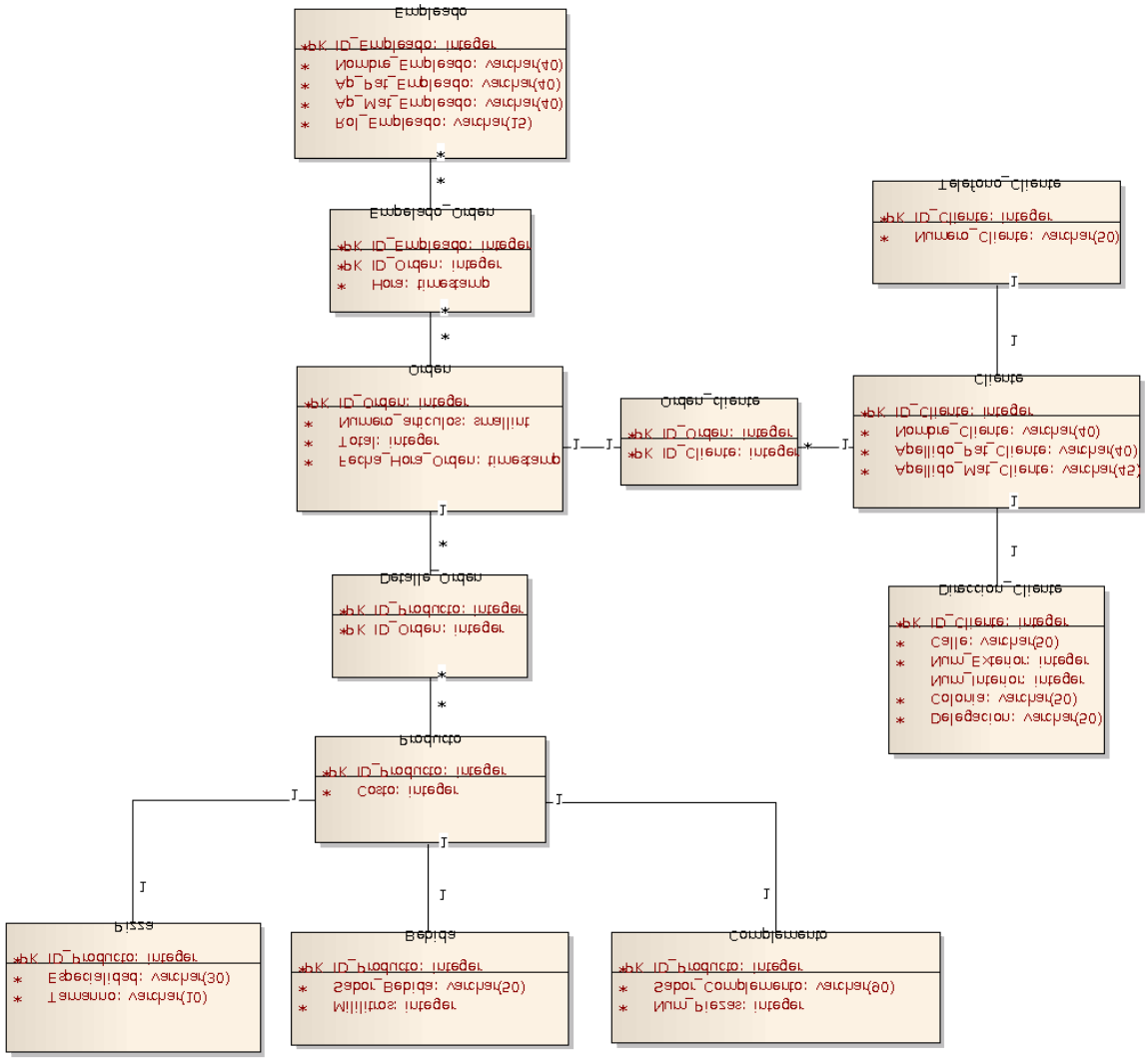

Figura 5.5 - Diagrama de Clases de la Pizzería.

Como se revisó en el punto 3 de ésta práctica, para comenzar la construcción del esquema asociado a este diagrama, el primer paso será crear la base de datos en la que se cargarán todas las tablas. El código para lograr esto es el siguiente:

```
CREATE DATABASE pizzeria;
```

Una vez creada la base de datos, ingresaremos a ella ya sea mediante la consola de PostgreSQL o seleccionándola dentro de la pantalla principal de DataGrip. Al estar dentro de ésta, procederemos a construir cada una de sus tablas.

A continuación se muestra la sintaxis asociada a la clase Pizza que se convertirá en la tabla Pizza. Ver Figura 5.6.

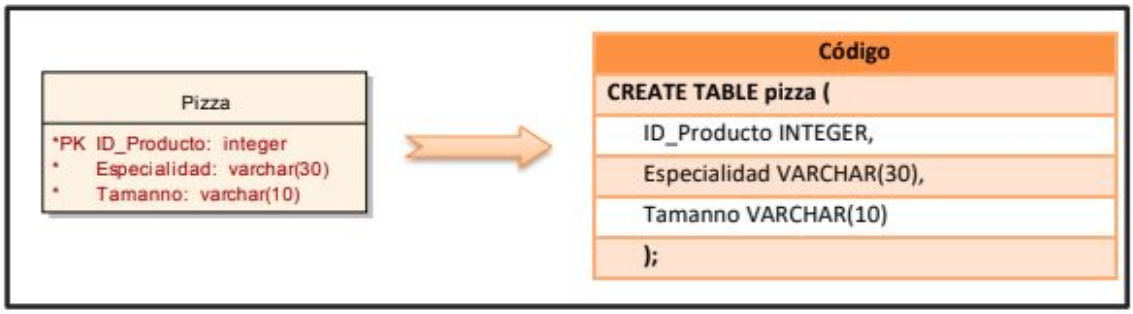

Figura 5.6 - Conversión de una clase a una tabla.

En la Figura 5.7 se muestra el código de cada una de las tablas asociadas a nuestra base de datos. Dicho código fue construido de la misma manera como se hizo con la tabla Pizza.

```
CREATE TABLE Producto (
                ID_Producto INTEGER,
                Costo INTEGER);
CREATE TABLE Bebida (
                ID_Producto INTEGER,
                Sabor_Bebida VARCHAR(50),
                Mililitros INTEGER);
CREATE TABLE Complemento (
                ID_Producto INTEGER,
                Sabor_Complemento VARCHAR(90),
                Num_Piezas INTEGER);
CREATE TABLE Pizza (
                ID_Producto INTEGER,
                Especialidad VARCHAR(35),
                Tammano VARCHAR(10));
CREATE TABLE Orden (
                ID_Orden INTEGER,
                Numero_Articulos SMALLINT,
                Total INTEGER);
CREATE TABLE Detalle_orden (
                ID_Producto INTEGER,
                ID_Orden INTEGER);
CREATE TABLE Empleado (
                ID_Empleado INTEGER,
                Nombre_Empleado VARCHAR(40),
                Ap_Pat_Empleado VARCHAR (40),
                Ap_Mat_Empleado VARCHAR (45),
                Area_Trabajo VARCHAR(15));
CREATE TABLE Empleado_Orden (
                ID_Empleado INTEGER,
                ID_Orden INTEGER,
                Hora TIMESTAMP);
CREATE TABLE Cliente (
                ID_Cliente INTEGER,
                Nombre_Cliente VARCHAR(40),
                Apellido_Pat_Cliente VARCHAR(40),
                Apellido_Mat_Cliente VARCHAR(45));
CREATE TABLE Orden_Cliente (
                ID_Orden INTEGER,
                ID_Cliente INTEGER);
CREATE TABLE Telefono_Cliente (
                ID_Orden INTEGER,
                ID_Cliente VARCHAR(20));
CREATE TABLE Direccion_cliente (
                ID_Cliente INTEGER,
                Calle VARCHAR(60),
                Num_Exterior INTEGER ,
                Num_Interior INTEGER,
                Colonia VARCHAR(70),
                Delegacion VARCHAR (65));
```

Figura 5.7 - Código DDL completo para la Pizzería.

Como podemos apreciar no se utilizan acentos ni caracteres especiales como la letra ñ, ya que el SMBD no los interpreta con precisión, lo cual puede ocasionar errores que al final se reflejarán en la eficiencia de la base de datos.

5. Integridad

La integridad dentro de las bases de datos puede ser vista como un conjunto de reglas, características y opciones que permiten al diseñador aproximar el modelo a la realidad. Es decir, la integridad es una herramienta para construir de manera más detallada la base de datos. Existen cuatro diferentes tipos de integridad:

a) Integridad de Entidad.
b) Integridad Referencial.
c) Integridad de Dominio.
d) Integridad de No Nulidad.

En las siguientes subsecciones describiremos con mayor detalle cada una de ellas.

1.1. Integridad de Entidad

Este tipo de integridad, como su nombre lo indica, hace referencia a las Entidades (Clases o Tablas) y a su llave primaria. El concepto de llave primaria se presentó durante el *Modelado con Diagramas Entidad – Relación* (Práctica 02), como breve recordatorio, la llave primaria de una Entidad es el conjunto mínimo de Atributos que identifican de manera única a una tupla de esa Entidad.

La integridad de Entidad protege a la base de datos de posibles errores debido al almacenamiento de tuplas iguales, además de permitir la identificación de manera única de cada una de las tuplas. Los errores que se pueden presentar debido a las duplicidades dentro de una tabla son amplios y pueden generar un fuerte impacto en la posterior explotación e interpretación de la base de datos.

Para explicar esta problemática retomaremos el ejemplo de los vehículos presentado durante el *Modelado con Diagramas Entidad – Relación* (Práctica 02). Ver Tabla 5.1.

Tipo	***Nombre de la marca***	***Motor***	***Modelo***	***Capacidad***	***Placa***	***Color***	***Nombre***
Terrestre	Volvo	4 cilindros	2012	5 pasajeros	472 YBZ	Rojo	Automóvil
Acuático	Harley Davidson	2 cilindros	2008	2 pasajeros	511 WAS	Rojo	Moto acuática
Terrestre	Mercedez - Benz	12 cilindros	1998	20 toneladas	925 VOK	Verde	Trailer
Aéreo	Boeing	Doble Turbina	2002	524 pasajeros	106 XCC	Azul	Avión
Terrestre	Italika	Eléctrico	2002	1 pasajero	723 UHI	Amarillo	Motoneta

Terrestre	Granja San Marcos	Sin motor	2007	80 kilogramos	250 MKL	Café	Yegua
Terrestre	Bennotto	Sin motor	2005	1 pasajero	776 WMB	Naranja	Bicicleta
Acuático	Volvo	Fuera de borda	2000	15 pasajeros	666 ZIO	Blanco	Bote
Aéreo	Boeing	Doble Turbina	2003	524 pasajeros	872 YMC	Azul	Avión

Tabla 5.1. Tabla Transporte

En la tabla anterior se puede apreciar que todos los renglones contienen combinaciones únicas de valores (es decir, que no existe un renglón que sea idéntico a otro en totalidad), a pesar de que existen columnas donde se repite más de una vez el mismo valor. Por ejemplo, la columna tipo, solo ha almacenado tres valores diferentes: Terrestre, Acuático y Aéreo, repitiéndose más de una vez cada uno en la columna correspondiente, pero si a ésta supuesta duplicidad agregamos los valores de todas las columnas, entonces la combinación resultante es única, concluyendo que no existe duplicidad de registros.

Modificando la Tabla 5.1 incluiremos registros con la intención de hacer registros duplicados. Ver Tabla 5.2.

Tipo	***Nombre de la marca***	***Motor***	***Modelo***	***Capacidad***	***Placa***	***Color***	***Nombre***
Terrestre	Volvo	4 cilindros	2012	5 pasajeros	472 YBZ	Rojo	Automóvil
Terrestre	Italika	Eléctrico	2002	1 pasajero	723 UHI	Amarillo	Motoneta
Acuático	Harley Davidson	2 cilindros	2008	2 pasajeros	511 WAS	Rojo	Moto acuática
Terrestre	Mercedez - Benz	12 cilindros	1998	20 toneladas	925 VOK	Verde	Trailer
Aéreo	Boeing	Doble Turbina	2002	524 pasajeros	106 XCC	Azul	Avión
Terrestre	Italika	Eléctrico	2002	1 pasajero	723 UHI	Amarillo	Motoneta
Terrestre	Granja San Marcos	Sin motor	2007	80 kilogramos	250 MKL	Café	Yegua
Terrestre	Bennotto	Sin motor	2005	1 pasajero	776 WMB	Naranja	Bicicleta
Acuático	Volvo	Fuera de borda	2000	15 pasajeros	666 ZIO	Blanco	Bote
Acuático	Harley Davidson	2 cilindros	2008	2 pasajeros	511 WAS	Rojo	Moto acuática
Aéreo	Boeing	Doble Turbina	2003	524 pasajeros	872 YMC	Azul	Avión

Tabla 5.2. Ejemplificación de duplicidad de registros.

En la Tabla 5.2 a simple vista se dificulta el apreciar la duplicidad de los registros, ya que la estructura de la tabla muestra columnas cuya información se repite en otros renglones con frecuencia. Si revisamos con cuidado el renglón 3 y el último renglón de la tabla podremos constatar que es un registro duplicado. De la misma manera para el renglón 2 y el renglón 6.

En dado caso que tuviéramos una tabla con cientos de miles de registros, el proceso de revisar registros duplicados manualmente sería complicado y costoso.

Por esta razón, la integridad de Entidad nos asegura que éste problema se evitará si se implementa de manera correcta la llave primaria (PK) a cada una de las Entidades (Clases o Tablas). Hay que tener en cuenta que la selección de una llave primaria está íntimamente ligada con la naturaleza de la entidad a la cual hace referencia y su elección depende íntegramente del diseñador. Para el ejemplo anterior (Tabla 5.2) se propone como PK el atributo Placa, ya que es único por cada transporte de la tabla.

Para integrar la llave primaria (PK) al código SQL de definición de datos (DDL), se utiliza la sintaxis mostrada en la Figura 5.8.

```
CREATE TABLE tabla1
(
        columna1 integer,
        columna2 varchar(50),
        PRIMARY KEY (columna1)
);
```

Figura 5.8 - Sintaxis de llave primaria.

La sintaxis de la llave primaria se integra a la sintaxis de CREATE TABLE presentada en el punto 3.2 del DDL. Como se puede observar, en la última línea de la descripción de las columnas, se encuentra:

1. PRIMARY KEY
   Palabras reservadas para añadir la característica de llave primaria a una columna (o conjunto de columnas) en específico. La columna nombrada columna1, ha adquirido la restricción de ser único y no nulo. Es decir, que cualquier registro debe, forzosamente contener un valor, el cual debe ser único, en la columna columna1.
2. (
   El símbolo de apertura de paréntesis sirve para indicar que comenzará la descripción de las columnas de esa tabla que formarán parte de la llave primaria.
3. columna1
   Es el nombre o listado de nombres separados por comas de los atributos que formarán parte de la llave primaria.
4.)
   El símbolo de cierre de paréntesis indica al SMBD que ha terminado la definición de la llave primaria.

De esta manera se ha implementado la llave primaria de una tabla, logrando así la integridad de esa Entidad.

1.2. **Integridad Referencial**

Las bases de datos relacionales basan su estructura y funcionalidad en las relaciones de las tablas que la componen. Estas relaciones son detectadas en las primeras etapas del diseño de la base de datos.

La integridad referencial utiliza como herramienta a las Llaves Foráneas (FK por sus siglas en inglés) para lograr y garantizar que las relaciones que se detectaron y modelaron, se cumplan dentro de la estructura de la base de datos. La ventaja de garantizar el uso de estas relaciones se verá reflejada en la congruencia y consistencia de los datos almacenados, es decir, se proveerá a la base de datos de herramientas automáticas para asegurar que los datos que se guardarán serán veraces reduciendo la posibilidad de errores.

Una FK es un conjunto mínimo de atributos que identifican de manera única un registro en otra tabla. Es decir, la FK es una regla de correspondencia, ya que, si una columna es FK, quiere decir que los datos que se almacenarán en esa columna provendrán de una columna de otra tabla; en esa otra tabla, la columna a la cual hace referencia la FK, deberá de ser forzosamente PK.

Para explicar gráficamente este concepto, en la Figura 5.9 se muestran tres tablas que contienen PK y FK. Como ejemplo, este diagrama puede modelar los pedidos de una tienda, sus clientes y el detalle de cada pedido.

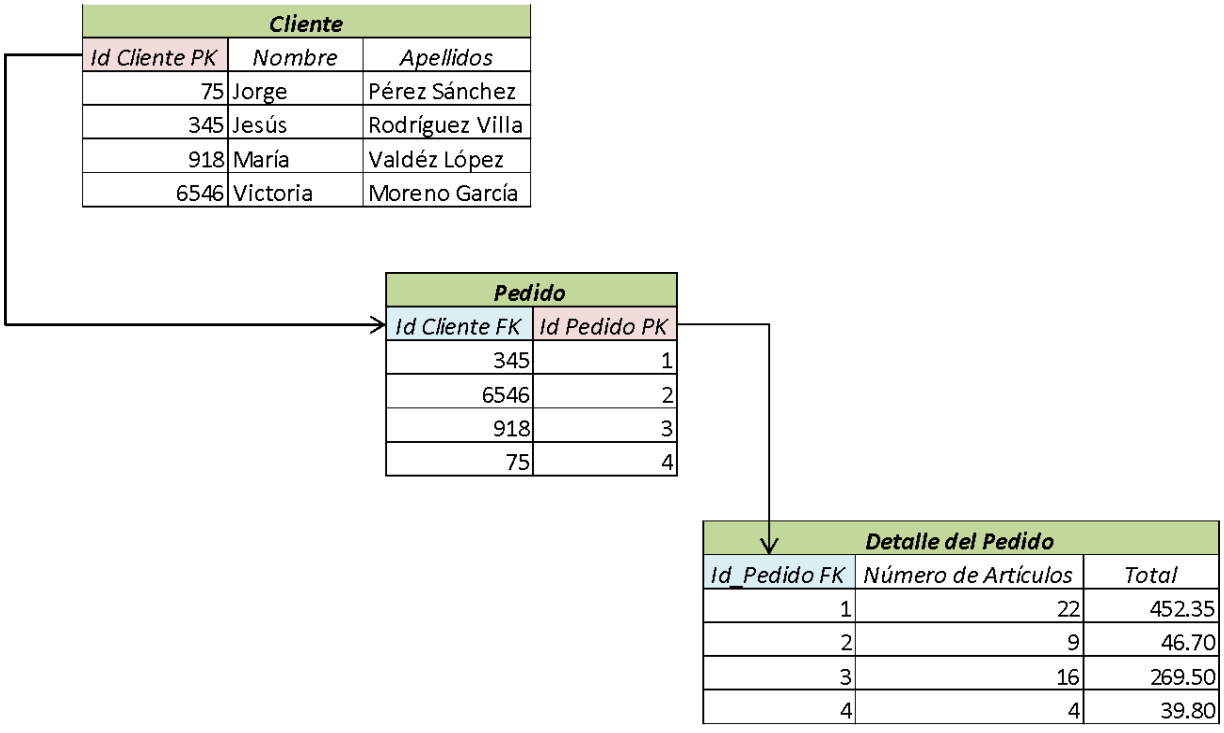

Figura 5.9 - Diagrama de tablas con PK y FK.

Podemos apreciar que la columna *Id Cliente* de la tabla *Cliente*, está marcada como **PK**, esto nos ofrece la seguridad de que solo existen registros únicos dentro de esa tabla, es decir, que no existe un mismo número de identificación para dos clientes.

En la tabla *Pedido*, como la columna *Id Cliente* está marcada como **FK** y con ayuda de la flecha, podemos inferir que la columna *Id Cliente* de la tabla *Cliente* le otorgará a ésta (*Id Cliente* de *Pedidos*) sus datos.

Esta regla de integridad nos garantiza que todos los datos que se guardarán en la columna FK provienen de una tabla segura, donde no existen duplicidades. Si no tuviéramos este tipo de seguridad, podríamos generar ambigüedades en las tablas que se encuentran involucradas en una relación. Por ejemplo, al generar un nuevo pedido, no sabríamos a que cliente asociarlo puesto que pudieran existir dos clientes con el mismo número de identificación.

De manera análoga a la relación de las tablas *Cliente* y *Pedido*, podemos ver la relación entre *Pedido* y *Detalle de Pedido*, teniendo en la tabla *Pedido* la columna denominada **PK** y en *Detalle Pedido* la columna con **FK**.

La manera de actuar de ésta **FK** es la siguiente. Una vez que el diseñador ha dispuesto que una columna sea llave foránea de otra tabla el SMBD pondrá en marcha un protocolo de verificación de datos cada vez que se pretenda insertar, modificar o eliminar un registro. Este protocolo inicia tomando el valor que se pretende insertar, modificar o eliminar para buscarlo en la columna de la tabla donde es **PK**, si lo encuentra, permite al usuario realizar los cambios pertinentes, en caso contrario, prohíbe su inserción, modificación o eliminación según sea el caso.

La sintaxis en lenguaje SQL para añadir las características de FK al código DDL es mostrada en la Figura 5.10.

```
CREATE TABLE tabla2
(
        columna3 integer,
        columna4 integer,
        columna5 varchar(15),
        FOREIGN KEY (columna4) REFERENCES tabla_1(columna1)
);
```

Figura 5.10 - Sintaxis de llave foránea.

Como podemos observar, en la última línea de la descripción de las columnas, tenemos:

1. FOREIGN KEY
    a. Palabras reservadas para añadir la característica de llave foránea a una columna (o conjunto de columnas) en específico. La columna nombrada columna4, ha

adquirido la restricción de solo poder insertar valores que existan en una columna específica de otra tabla.

2. (

   El símbolo de apertura de paréntesis sirve para indicar que comenzará la descripción de los atributos de esa tabla que formarán parte de la llave foránea.
3. Columna4

   Es el atributo o listado de atributos separados por comas que formarán parte de la llave foránea.
4. REFERENCES

   Este comando indica que comenzará la definición del atributo o conjunto de atributos a las que se hará referencia.
5. tabla1

   Es el nombre de la tabla donde se encuentra el atributo o conjunto de atributos a los que se hará referencia.
6. (

   El símbolo de apertura de paréntesis sirve para indicar que comenzará la descripción de los atributos a los que hará referencia la llave foránea.
7. columna1

   Es el atributo o listado de atributos separados por comas que serán referenciados por la llave foránea.
8.)

   El símbolo de cierre de paréntesis indica al SMBD que ha terminado la definición de los atributos a los que hará referencia la llave foránea.
9.)

   El símbolo de cierre de paréntesis indica al SMBD que ha terminado la definición de la llave foránea.

Es importante notar que la columna FK deberá de ser parte de la tabla que se declaró en el punto 4. Cabe destacar que el orden en el que se declaren las tablas es trascendental ya que la tabla que sirve de referencia, es decir donde se declaró como PK, tiene que ser declarada antes que la tabla que hace referencia, donde se declara como FK.

1.3. **Integridad de Dominio**

En matemáticas, un dominio es un conjunto de elementos para los cuales, mediante una función dada, se le asocia un único elemento de otro conjunto. Retomando esta línea de pensamiento, un dominio en base de datos será el conjunto de los valores posibles que acepte una columna de una tabla.

Este concepto no es del todo nuevo, ya que durante el *Lenguaje de Definición de Datos DDL* fue presentado el concepto de tipos de datos. Los datos que soportan los diferentes SMBD son de naturaleza general, es decir, si una columna va a recibir el número de hijos de una persona, el tipo de dato podría ser modelado como un entero, es decir, *integer*. Pero esto no modela de manera exacta la realidad, ya que en esa columna probablemente solo deberíamos de tener la posibilidad de insertar valores mayores o iguales a cero, sin embargo, podríamos insertar números negativos lo que claramente sería incongruente.

La integridad de dominio otorga la capacidad al diseñador de poder acotar el conjunto de posibles valores de un atributo, y en consecuencia el tipo de dato, en particular este conjunto de valores es conocido como Dominio. Esto se logra utilizando la palabra reservada *CHECK* dentro de la sintaxis de la creación de las tablas, donde se detallan las características de cada columna.

La Figura 5.11 muestra la sintaxis de éste tipo para la inclusión de la cláusula CHECK.

```
CREATE TABLE Cliente
(
        ID_cliente integer,
        Apellidos varchar(255),
        Nombre varchar(255),
        Genero char(1),
        CHECK (Genero IN ('M', 'F'))
);
```

Figura 5.11. Sintaxis de cláusula CHECK.

La sintaxis de la cláusula CHECK se integra a la sintaxis de CREATE TABLE vista durante el *Lenguaje de Definición de Datos DDL*. Como podemos observar, en la última línea de la descripción de las columnas, se encuentran:

1. CHECK
   Palabra reservada para comenzar con la restricción del dominio de una columna en específico.
2. (
   La apertura de paréntesis indica el comienzo de los parámetros que utilizará el SMBD para llevar a cabo la restricción de dominio.
3. Genero
   Nombre de la columna sobre la cual se hará la validación de los valores a insertar.
4. IN
   Palabra reservada para incluir un conjunto de elementos que será el dominio de la columna descrita anteriormente. También se puede usar su negación NOT IN.
5. (

El símbolo de apertura de paréntesis sirve para indicar que comenzará la descripción de los valores que conformarán el dominio del atributo.

6. 'M', 'F'
   Es el listado de valores del dominio, separados por comas, que permitirá el SMBD para el atributo en cuestión.
7.)
   El símbolo de cierre de paréntesis indica el término de la lista de valores que formarán el dominio.
8.)
   El símbolo de cierre de paréntesis indica el término de la cláusula CHECK.

La cláusula *CHECK* actúa revisando si la condición es verdadera o falsa, es decir, si el valor que se pretende insertar en esa columna cumple o no con la condición descrita anteriormente. Al resultar verdadera la condición, la cláusula CHECK permite la inserción, en otro caso la restringe.

Existen otros casos de condición como: (A = B), (C <> D), (E *NOT IN* ('E', 'F',...,'X')), (X < 0), etc. En particular, existe un comparativo que permite indicar un patrón establecido de caracteres. Ver Figura 5.12.

Figura 5.12 - Cláusula CHECK – LIKE.

En la Figura 5.8 se muestra un ejemplo donde el dominio de la columna llamada *columnaX* ha quedado reducido a elementos que cumplen con el patrón establecido, es decir, valores de la siguiente forma:

AF56 HR34 LO89 DE08

Las cláusulas CHECK pueden ser sensibles a cambios durante el tiempo. Por ejemplo, si modeláramos una columna que almacena los tipos de pago de un pedido, es posible que al principio solo se pudiera pagar con efectivo o tarjeta de crédito; pero al avanzar en el tiempo, el negocio decida también recibir pagos electrónicos como transferencias o instrumentos digitales. Por lo tanto nuestra primera cláusula CHECK se vería de la siguiente forma:

CHECK (tipo_de_pago IN ('Efectivo', 'Tarjeta_Credito'))

Para agregar las nuevas formas de pago se requeriría de reescribir la clausual CHECK de la siguiente manera:

CHECK (tipo_de_pago IN ('Efectivo', 'Tarjeta_Credito','Transferencia','Paypal'))

Para lograr este cambio es necesario eliminar la primera cláusula y después agregar la nueva. La sintaxis que logra esto, se encuentra en la Figura 5.13.

```
ALTER TABLE nombre_tabla
DROP CONSTRAINT nombre_de_la_cláusula;

ALTER TABLE nombre_tabla
ADD CHECK (nombre_columna IN ('M','F'));
```

Figura 5.13 - Cláusula CHECK – LIKE.

1.4. **Integridad de No Nulidad.**

Un problema frecuente en el diseño de las bases de datos es el tratamiento adecuado de los campos nulos. En bases de datos, así como en las matemáticas, el cero y el vacío no son lo mismo. De esta manera en el diseño de la base de datos que se pretende construir, se deben de identificar los posibles valores nulos y los posibles valores vacíos.

Para ejemplificar esto, podríamos pensar en una persona de origen ruso. En ese país, a los niños se les registra únicamente con el apellido del padre. Si pensamos que este personaje tiene relaciones comerciales con una empresa mexicana donde se le ha pedido que registre sus datos personales, un correcto tratamiento sería que registrara en el campo apellido materno la palabra NULL (nulo en inglés), ya que no existe esa información, es decir, el vacío. Si el campo quedara en blanco, significaría que la información existe, pero se desconoce.

Tomando en cuenta la existencia de valores nulos o vacíos, se deberá tomar en cuenta antagónicamente la prohibición de la nulidad o inexistencia de un valor. Es decir, limitar la posibilidad de una columna de no ser llenada. Siguiendo con el ejemplo de la persona de nacionalidad rusa, un requisito de no nulidad podría ser el número de cuenta bancaria donde se le depositarán sus pagos, ya que, sin este número, no se podrían llevar a cabo las operaciones de la empresa.

La integridad de No Nulidad ofrece al SMBD la capacidad de prohibir que los valores de una columna específica queden vacíos o nulos. Esta herramienta se implementa en la sintaxis de la creación de tablas, en el detalle de las características de cada columna.

La Figura 5.14 muestra la sintaxis de este tipo de integridad.

Figura 5.14 - Sintaxis NOT NULL.

La sintaxis de la integridad de no nulidad es simple, ya que solo es necesario añadir a la sintaxis original las palabras reservadas NOT NULL después del tipo de columna, para cada columna que así lo amerite.

En la Figura 5.15 se ejemplifica la sintaxis que involucra a los cuatro tipos de integridades.

Figura 5.15 - Ejemplo de Sintaxis de los cuatro tipos de Integridad.

Profesor
L. en C.C. Erick Orlando Matla Cruz
Ayudantes
L. en C.C. Efraín Hipólito Chamú
L. en C.C. Anahí Quiroz Jiménez
L. en C.C. Karen Monserrat Zavala Correa
Grupo: 9119

6. Ejercicios

(NOTA: Resuelve los siguientes ejercicios en relación al proyecto que realizarás durante el curso, en dado caso que no tengas un proyecto, utiliza la información en el apéndice Seguros parte 05 al final de esta práctica para realizarlos)

1. Crea la base de datos de tu proyecto.
2. Utilizando código DDL, construye las tablas que integrarán el **Esquema la base de datos** de acuerdo al diagrama de Clases de tu proyecto.
3. Integra las llaves primarias (PK) y llaves foráneas a las tablas del punto 2.
4. Genera un listado en donde describas el nombre de cada tabla y lo que representa cada una.
5. Dentro del listado del punto 4, agrega para cada tabla a todos los atributos que la conforman, lo que representa cada uno de ellos y el tipo de dato asociado a éstos.
6. De acuerdo al tipo de dato de las columnas, implementa cláusulas CHECK (si es necesario) para acotar el dominio de tus tablas.
7. Verifica la existencia de datos nulos en las tablas y añade la condición de No Nulidad (NOT NULL) para todas las columnas que así lo ameriten.
8. ¿Por qué no se agregó la condición de No Nulidad (NOT NULL) a las columnas ID_cliente y Genero de la tabla Cliente2 de la Figura 5.15?
9. ¿Por qué no se agregó la condición de No Nulidad (NOT NULL) a las columnas ID_pedido, ID_cliente, Numero_articulos y Total de la tabla Pedido de la Figura 5.15?.

Entregables requeridos para prácticas subsecuentes:

- Esquema de la base de datos
- Integridad al esquema de la base de datos

2. Apéndice Seguros parte 05 DDL

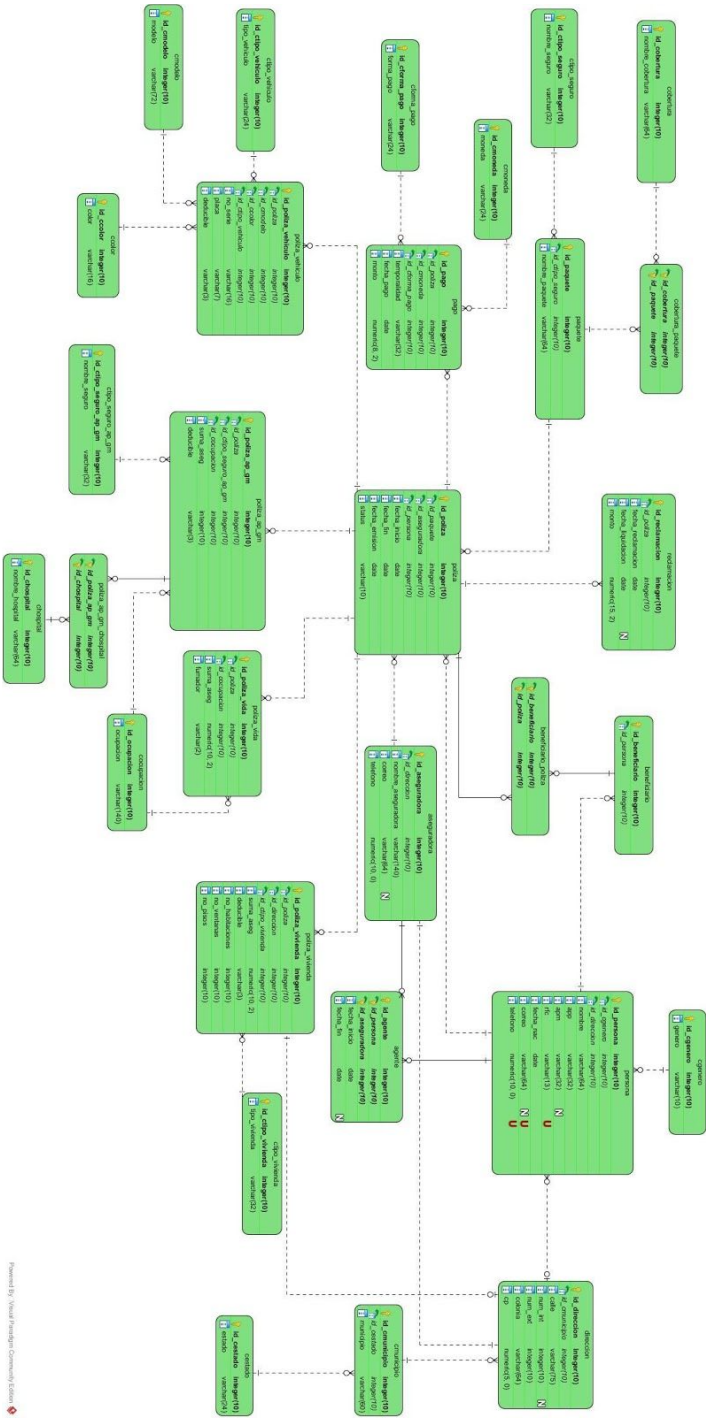

Figura 5.16 - Diagrama de Clases UML del proyecto Seguros

Profesor
L. en C.C. Erick Orlando Matla Cruz
Ayudantes
L. en C.C. Efraín Hipólito Chamú
L. en C.C. Anahí Quiroz Jiménez
L. en C.C. Karen Monserrat Zavala Correa
Grupo: 9119

3. Apéndice Seguros parte 06 Integridad

Siguiendo los supuestos que se encuentran en el Apéndice Seguros parte 01, realiza todos los cambios de Integridad de Entidad (PK), Integridad Referencial (FK), Integridad de Dominio (CHECK y tipos de datos) e Integridad de No Nulidad (NOT NULL) al código DDL siguiente.

```
CREATE TABLE poliza (
        id_poliza  INTEGER,
        id_paquete INTEGER,
        id_aseguradora INTEGER,
        id_persona INTEGER,
        fecha_inicio DATE,
        fecha_fin  DATE,
        fecha_emision DATE,
        status VARCHAR(10)
);

CREATE TABLE monto
(
        id_monto INTEGER,
        id_poliza  INTEGER,
        id_cmoneda INTEGER,
        id_cforma_pago INTEGER,
        temporalidad VARCHAR(32),
        fecha_pago DATE
);

CREATE TABLE cforma_pago
(
        id_cforma_pago INTEGER,
        forma_pago VARCHAR(24)
);

CREATE TABLE cobertura
(
        id_cobertura INTEGER,
        nombre_cobertura VARCHAR(64)
);

CREATE TABLE agente
(
        id_agente INTEGER,
        id_persona INTEGER,
        id_aseguradora INTEGER,
        fecha_inicio DATE,
        fecha_fin  DATE
);


CREATE TABLE reclamacion (
        id_reclamacion INTEGER,
        id_poliza INTEGER,
        fecha_reclamacion  DATE,
```

Profesor
L. en C.C. Erick Orlando Matla Cruz
Ayudantes
L. en C.C. Efraín Hipólito Chamú
L. en C.C. Anahí Quiroz Jiménez
L. en C.C. Karen Monserrat Zavala Correa
Grupo: 9119

```
        fecha_liquidacion DATE,
        monto NUMERIC(15,2)
);

CREATE TABLE cmoneda
(
        id_cmoneda INTEGER,
        moneda VARCHAR(24)
);

CREATE TABLE paquete
(
        id_paquete INTEGER,
        id_ctipo_seguro INTEGER,
        nombre_paquete VARCHAR(64)
);

CREATE TABLE cobertura_paquete
(
        id_cobertura INTEGER,
        id_paquete INTEGER
);

CREATE TABLE ctipo_seguro
(
        id_ctipo_seguro INTEGER,
        nombre_seguro VARCHAR(32)
);

CREATE TABLE aseguradora
(
        id_aseguradora INTEGER,
        id_direccion INTEGER,
        nombre_aseguradora VARCHAR(140),
        teléfono NUMERIC(10,0),
        correo VARCHAR(64)
);

CREATE TABLE persona
(
        id_persona INTEGER,
        id_cgenero INTEGER,
        id_direccion INTEGER,
        nombre VARCHAR(64),
        app VARCHAR(32),
        apm VARCHAR(32),
        rfc VARCHAR(13),
        fecha_nac DATE,
        correo VARCHAR(64),
        telefono NUMERIC(10,0)
);

CREATE TABLE cestado
(
        id_cestado INTEGER,
        estado VARCHAR(24)
```

Profesor
L. en C.C. Erick Orlando Matla Cruz
Ayudantes
L. en C.C. Efraín Hipólito Chamú
L. en C.C. Anahí Quiroz Jiménez
L. en C.C. Karen Monserrat Zavala Correa
Grupo: 9119

```
);

CREATE TABLE cmunicipio
(
        id_cmunicipio INTEGER,
        id_cestado INTEGER,
        municipio VARCHAR(60)
);

CREATE TABLE cgenero
(
        id_cgenero INTEGER,
        genero VARCHAR(10)
);

CREATE TABLE ctipo_vivienda
(
        id_ctipo_vivienda INTEGER,
        tipo_vivienda VARCHAR(32)
);

CREATE TABLE poliza_vivienda
(
        id_poliza_vivienda INTEGER,
        id_poliza  INTEGER,
        id_direccion INTEGER,
        id_ctipo_vivienda INTEGER,
        suma_aseg NUMERIC(10,2),
        deducible VARCHAR(3),
        no_habitaciones INTEGER,
        no_ventanas INTEGER,
        no_pisos  INTEGER
);

CREATE TABLE direccion
(
        id_direccion INTEGER,
        id_cmunicipio INTEGER,
        calle VARCHAR(75),
        num_int INTEGER,
        num_ext  INTEGER,
        colonia VARCHAR(64),
        cp NUMERIC(5,0)
);

CREATE TABLE poliza_vida
(
        id_poliza_vida INTEGER,
        id_poliza  INTEGER,
        id_cocupacion INTEGER,
        suma_aseg NUMERIC(10,2),
        fumador VARCHAR(2)
);

CREATE TABLE cocupacion
(
```

```
        id_cocupacion INTEGER,
        ocupación VARCHAR(140)
);

CREATE TABLE poliza_ap_gm
(
        id_poliza_ap_gm INTEGER,
        id_poliza INTEGER,
        id_ctipo_seguro_ap_gm INTEGER,
        id_cocupacion INTEGER,
        deducible VARCHAR(3),
        suma_aseg INTEGER
);

CREATE TABLE chospital
(
        id_chospital INTEGER,
        nombre_hospital VARCHAR(64)
);

CREATE TABLE poliza_ap_gm_chospital
(
        id_poliza_ap_gm INTEGER,
        id_chospital INTEGER
);

CREATE TABLE ctipo_seguro_ap_gm
(
        id_ctipo_seguro_ap_gm INTEGER,
        nombre_seguro VARCHAR(32)
);

CREATE TABLE poliza_vehiculo
(
        id_poliza_vehiculo  INTEGER,
        id_poliza  INTEGER,
        id_cmodelo INTEGER,
        id_ccolor INTEGER,
        id_ctipo_vehiculo INTEGER,
        no_serie VARCHAR(16),
        placa VARCHAR(7),
        deducible VARCHAR(3)
);

CREATE TABLE ctipo_vehiculo
(
        id_ctipo_vehiculo INTEGER,
        tipo_vehiculo VARCHAR(24)
);

CREATE TABLE cmodelo
(
        id_cmodelo INTEGER,
        modelo VARCHAR(72)
);
```

Profesor
L. en C.C. Erick Orlando Matla Cruz
Ayudantes
L. en C.C. Efraín Hipólito Chamú
L. en C.C. Anahí Quiroz Jiménez
L. en C.C. Karen Monserrat Zavala Correa
Grupo: 9119

```
CREATE TABLE ccolor
(
        id_ccolor INTEGER,
        color VARCHAR(16)
);

CREATE TABLE beneficiario
(
        id_beneficiario INTEGER,
        id_persona INTEGER
);

CREATE TABLE beneficiario_poliza
(
        id_beneficiario INTEGER,
        id_poliza INTEGER
)
```