

Práctica 06

Integridad

1. Objetivo General

Conocer los diferentes tipos de integridad inherentes a las bases de datos.

2. Objetivos Secundarios

- Modelar las características singulares de la base de datos.
- Optimizar la congruencia y consistencia de los datos que se almacenarán en la base de datos.

3. Introducción

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:

1. Integridad de Entidad.
2. Integridad Referencial.
3. Integridad de Dominio.
4. Integridad de No Nulidad.

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

3.1. 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 6.1.

<i>Tipo</i>	<i>Nombre de la marca</i>	<i>Motor</i>	<i>Modelo</i>	<i>Capacidad</i>	<i>Placa</i>	<i>Color</i>	<i>Nombre</i>
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 6.1. 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 6.1 incluiremos registros con la intención de hacer registros duplicados. Ver Tabla 6.2.

<i>Tipo</i>	<i>Nombre de la marca</i>	<i>Motor</i>	<i>Modelo</i>	<i>Capacidad</i>	<i>Placa</i>	<i>Color</i>	<i>Nombre</i>
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

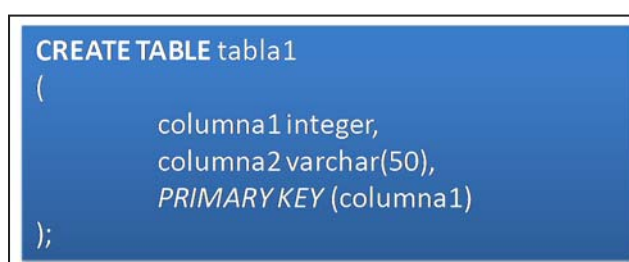
Tabla 6.2. Ejemplificación de duplicidad de registros.

En la Tabla 6.2 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 6.2) 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 6.1.



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

Figura 6.1 - Sintaxis de llave primaria.

La sintaxis de la llave primaria se integra a la sintaxis de CREATE TABLE presentada en el *Lenguaje de Definición de Datos DDL* (Práctica 05). 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 SDBD 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.

3.2. 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 éstas 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 6.2 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.

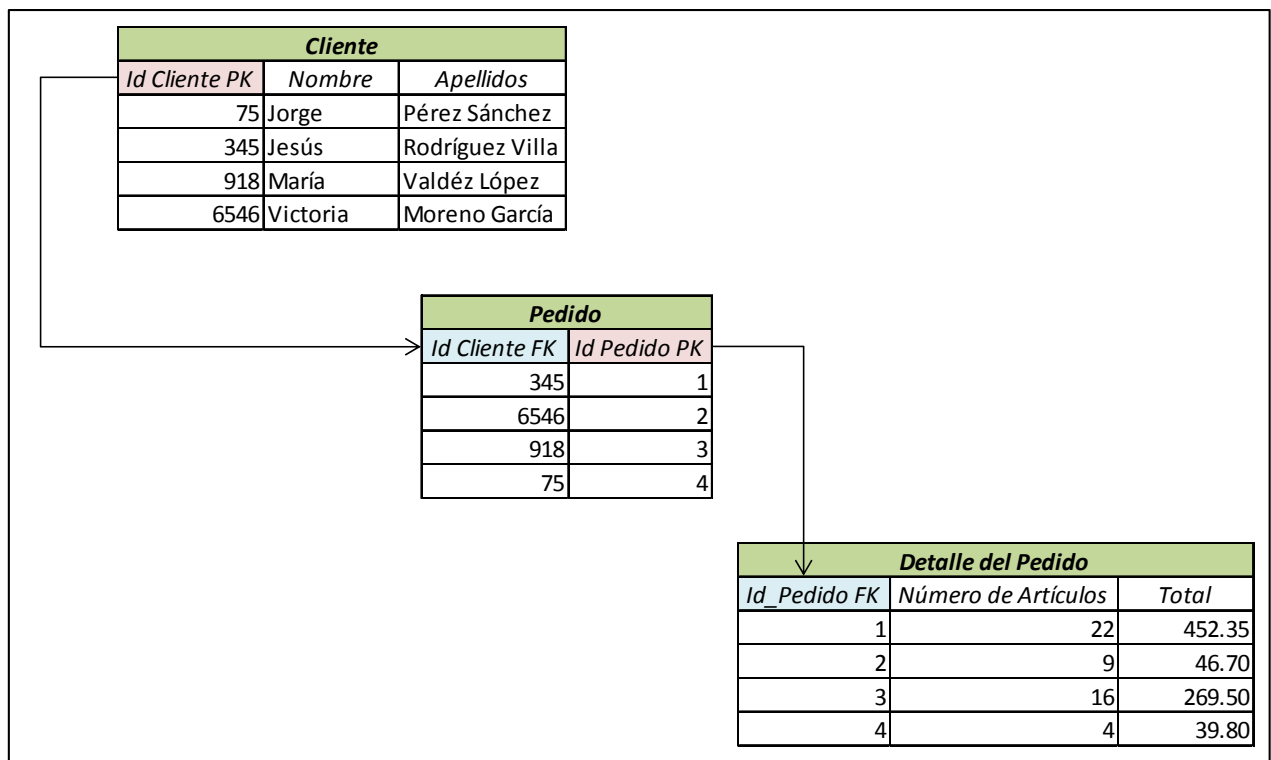


Figura 6.2 - Diagrama de tablas con PK y FK.

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 SMDB 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 6.3.

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

Figura 6.3 - Sintaxis de llave foránea.

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

1. **FOREIGN KEY**

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.

5. **columna4**

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

6. **)**

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

3. **REFERENCES**

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

4. **tabla1**

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

5. **(**

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

6. **columna1**

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

7. **)**

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.

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.

3.3. 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* (Práctica 05) 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 6.4 muestra la sintaxis de éste tipo para la inclusión de la clausula *CHECK*.

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

Figura 6.4. 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* (Práctica 05). 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 SMDB 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 SMDB 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 clausula *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: $(A = B)$, $(C <> D)$, $(E \text{ NOT IN } ('E', 'F', ..., 'X'))$, $(X < 0)$, etc. En particular, existe un comparativo que permite indicar un patrón establecido de caracteres. Ver Figura 6.5.



```
CHECK ( columnaX LIKE '[A-Z] [A-Z] [0-9] [0-9]' )
```

Figura 6.5 - Cláusula *CHECK* – *LIKE*.

En la Figura 6.5 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 clausula *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 6.6.



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

Figura 6.6 - Cláusula *CHECK* – *LIKE*.

3.4. 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 éste 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 6.7 muestra la sintaxis de éste tipo de integridad.

```
CREATE TABLE Cliente1(  
  ID_Cliente    INTEGER      NOT NULL,  
  Apellidos    VARCHAR(255) NOT NULL,  
  Nombre       VARCHAR(255) NOT NULL  
);
```

Figura 6.7 - 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 6.8 se ejemplifica la sintaxis que involucra a los cuatro tipos de integridades.

```
CREATE TABLE Cliente2(  
ID_Cliente SERIAL,  
Apellidos VARCHAR(255) NOT NULL,  
Nombre VARCHAR(255) NOT NULL,  
Genero CHAR(1),  
PRIMARY KEY (ID_Cliente),  
CHECK (Genero IN ('M','F'))  
);  
  
CREATE TABLE Pedido(  
ID_Pedido SERIAL,  
ID_Cliente INTEGER,  
Numero_Articulos INTEGER,  
Total INTEGER,  
PRIMARY KEY (ID_Pedido),  
FOREIGN KEY (ID_Cliente) REFERENCES Cliente2(ID_Cliente),  
CHECK (Numero_Articulos > 0),  
CHECK (Total > 0)  
);
```

Figura 6.8 - Ejemplo de Sintaxis de los cuatro tipos de Integridad.

4. 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 NFL-ONEFA parte 06 al final de esta práctica para realizarlos)

1. Comienza la implementación de **Integridad al esquema de la base de datos** identificando las llaves primarias (PK) de cada tabla.
2. Modifica el código DDL de la base de datos para incluir las PK identificadas.
3. Identifica las llaves foráneas (FK) de cada tabla.
4. Modifica el código DDL de la base de datos para incluir las FK identificadas.
5. Verifica el tipo de dato de todas las columnas e implementa cláusulas CHECK (si es necesario) para acotar el dominio.
6. 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.
7. ¿Por qué no se agregó la condición de No Nulidad (NOT NULL) a las columnas ID_cliente y Genero de la tabla Cliente de la Figura 6.8?
8. ¿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 6.8?

Entregables requeridos para prácticas subsecuentes:

- Integridad al esquema de la base de datos

5. Apéndice NFL-ONEFA parte 06

Siguiendo los supuestos que se encuentran en el Apéndice NFL-ONEFA 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 jugador (  
    id_jugador INT,  
    apellido_jugador VARCHAR(20),  
    nombre_jugador VARCHAR(20),  
    fecha_nacimiento DATE,  
    universidad VARCHAR(30)  
);
```

```
CREATE TABLE linea_ofensiva (  
    id_jugador INT,  
    posicion VARCHAR(2),  
    pancakes INT  
);
```

```
CREATE TABLE defensiva (  
    id_jugador INT,  
    posicion VARCHAR(2),  
    tackles INT,  
    sacks INT,  
    ff INT,  
    ints INT  
);
```

```
CREATE TABLE kicker (  
    id_jugador INT,  
    posicion VARCHAR(2),  
    fga INT,  
    fgm INT,  
    blocked INT,  
    longest INT  
);
```

```
CREATE TABLE punter (  
    id_jugador INT,  
    posicion VARCHAR(2),  
    punts INT,  
    net INT,  
    longest INT,  
    in20 INT  
);
```

```
CREATE TABLE quarterback (  
    id_jugador INT,  
    posicion VARCHAR(2),  
    tds INT,  
    ints INT,  
    yds INT,  
    att INT,  
    comp INT  
);
```

```
CREATE TABLE runner (  
    id_jugador INT,  
    posicion VARCHAR(2),  
    carries INT,  
    fumbles INT,  
    yds INT,  
    tds INT  
);
```

```
CREATE TABLE receiver (  
    id_jugador INT,  
    posicion VARCHAR(2),  
    rec INT,  
    yds INT,  
    fumbles INT,  
    tds INT  
);
```

```
CREATE TABLE equipo (  
    id_equipo VARCHAR(3),  
    nombre_equipo VARCHAR(20),  
    nickname_equipo VARCHAR(20),  
    division VARCHAR(10),  
    conferencia VARCHAR(10),  
    record_G INT,  
    record_P INT,  
    record_E INT,  
    anyo_fund INT  
);
```

```
CREATE TABLE entrenador (  
    id_entrenador INT,  
    apellido_entrenador VARCHAR(20),  
    nombre_entrenador VARCHAR(20),  
    cargo VARCHAR(2)  
);
```

```
CREATE TABLE ciudad (  
    id_ciudad INT,  
    nombre_ciudad VARCHAR(20),  
    estado_ciudad VARCHAR(2),  
    clima VARCHAR(20),  
    habitantes BIGINT,  
    indice_crecimiento INT,  
    region VARCHAR(30),  
    ingreso_promedio INT  
);
```

```
CREATE TABLE estadio (  
    id_estadio INT,  
    nombre_estadio VARCHAR(30),  
    capacidad INT,  
    grama VARCHAR(15),  
    condicion VARCHAR(10),  
    anyo_construccion INT,  
    id_ciudad INT  
);
```

```
CREATE TABLE jugador_equipo (  
    id_jugador INT,  
    id_equipo VARCHAR(3),  
    numero INT,  
    juegos_jugados INT,  
    juegos_iniciados INT,  
    anyo_adquirido INT  
);
```

```
CREATE TABLE equipo_ciudad (  
    id_equipo VARCHAR(3),  
    id_ciudad INT,  
    apoyo_ciudad INT  
);
```

```
CREATE TABLE equipo_estadio (  
    id_equipo VARCHAR(3),  
    id_estadio INT,  
    proyecto_nuevo VARCHAR(30)  
);
```

```
CREATE TABLE equipo_entrenador (  
    id_equipo VARCHAR(3),  
    id_entrenador INT,  
    anyo_contratado INT  
);
```

```
CREATE TABLE partido (  
    id_partido INT,  
    id_equipo_V VARCHAR(3),  
    id_equipo_L VARCHAR(3),  
    marcador_V INT,  
    marcador_L INT,  
    semana INT  
);
```