

UD4. SQL

Índice

Introducción.....	2
El SQL.....	2
Lenguaje de definición de datos (DDL).....	3
Lenguaje de manipulación de datos (DML).....	4
Lenguaje de consulta de datos (DQL).....	5
Lenguaje de control de datos y transacciones (DCL y TCL).....	6
Características del SQL.....	6
Como ejecutar comandos SQL.....	6
Comandos con SQLPLUS.....	7
Ingeniería inversa con el DataModeler.....	8
Convenciones de escritura de SQL.....	9
Tipos de datos.....	10
Create table.....	12
La foreign key.....	15
Claves con más de una columna.....	18
Check.....	20
Generalización y reflexión.....	24
Las relaciones reflexivas.....	24
Generalizaciones.....	25
Alter table.....	29
Añadir, eliminar o modificar columnas de la tabla.....	29
Añadir eliminar o modificar Constraints.....	31
Drop Table.....	33
Activar y desactivar constraints.....	34
Lenguaje de Manipulación de datos.....	35
Insert.....	35
Campos autonuméricos.....	37
GENERATED AS IDENTITY.....	38
Update.....	41
Delete.....	42

Introducción

En esta unidad, se profundiza en la primera etapa del diseño físico, traducción del modelo lógico al modelo físico usando como sistema gestor de bases de datos Oracle y un lenguaje universal para su gestión, **SQL**. Se abordarán los aspectos asociados con la creación de los objetos de una base de datos, en general, y de las tablas, en particular. Los comandos principales para tales acciones son **CREATE, ALTER y DROP**. La gestión se basa en estos procesos principales, el alta, la modificación o actualización y la eliminación de objetos. La gestión de datos no es más que el conjunto de acciones que permiten el alta de los datos, su modificación y su posible baja.

Otros comandos permiten introducir, modificar y eliminar datos en los objetos a través de registros en tablas. Los comandos que se usarán son **INSERT INTO, UPDATE SET WHERE y DELETE WHERE**. En la inserción de registros, es usual insertar valores autonuméricos en las columnas que son claves primarias. En Oracle, las secuencias son unas estructuras que permiten crear secuencias numéricas para aquellas columnas autonuméricas.

La creación de tablas conlleva la reserva de espacio físico en los sistemas de almacenamiento. La gestión del almacenamiento es una tarea muy importante para amortizar los sistemas de almacenamiento y repartir los recursos para aquellas tablas que lo necesitan más que otras. Además, el fin último es que los accesos a los archivos sean lo más eficientes posible, con el fin de acelerarlos, teniendo en cuenta su disposición lógica y física.

El SQL

Para la comunicación con los sistemas gestores de bases de datos, se utilizan programas informáticos que mandarán sentencias o comandos que un procesador de lenguaje interno al propio sistema gestor es capaz de interpretar y llevar acciones asociadas a cada uno de estos propósitos atómicos. Para dicha comunicación, se hace necesario un lenguaje compuesto de reglas léxicas, sintácticas y semánticas que determina cómo se deben escribir los comandos y asociarse entre sí. Este conjunto de comandos es interpretado por un parser o procesador de comandos que lleva a cabo las acciones asociadas a cada comando. El lenguaje más popular para la comunicación con sistemas gestores de bases de datos se denomina SQL.

De este modo, se puede decir que casi todos los sistemas gestores de bases de datos relacionales usan SQL como lenguaje de gestión de datos. De este modo, los conocimientos adquiridos con el estudio de este lenguaje son portables a la mayoría de ellos. Además, los programas escritos en SQL pueden ser ejecutados en cualquier motor, ya que los cambios entre estos es tarea fácil. Este lenguaje es usado tanto por programadores como por administradores y el fin último de este es ofrecer un conjunto de recursos para la gestión y administración completa de la base de datos y garantizar su integridad y consistencia. Oracle también dispone de herramientas muy cómodas que permite la gestión completa como Enterprise Manager.

Otras herramientas son SQL *plus, JDeveloper y SQL Workshop. La herramienta SQL *plus dispone de una interfaz de usuario basada en web. Para la programación en Java, servicios web y SQL, se puede usar JDeveloper, que proporciona una interfaz gráfica que integra código Java y un modelador de bases de datos, además de un depurador PL/SQL. Otra herramienta interesante es SQL Workshop que permite gestionar los objetos de la base de datos usando un procesador SQL con un repositorio muy amplio de scripts.

La mayoría de los sistemas gestores relacionales tienen incluido un parser que es capaz de traducir este lenguaje universal. Este lenguaje ha ido evolucionando desde los años setenta hasta la actualidad y se publican diferentes versiones. Este lenguaje se encuentra compuesto por un conjunto inmenso de comandos que se pueden clasificar según el propósito de ellos y englobarlos en sublenguajes de SQL.

Estos sublenguajes son principalmente cuatro: el lenguaje de definición de datos, el de manipulación de datos, el de control de datos y el de control de transacciones.

Lenguaje de definición de datos (DDL)

Este conjunto de comandos permite crear cada una de las tablas que compone la base de datos y definir las restricciones asociadas a ellas. Los comandos principales son **DESCRIBE**, **CREATE TABLE**, **ALTER TABLE** y **DROP TABLE**. El comando **DESCRIBE** permite mostrar la descripción de una tabla, mostrando el nombre de los campos, su orden, el tipo de datos y si el campo es opcional u obligatorio.

Su sintaxis es

DESC[RIBE] nombreTabla

No se necesita usar punto y coma al final, ya que se trata de un comando sin más cláusulas. Un ejemplo puede ser el comando DESC Proveedor. En la sintaxis del comando anterior, se han escrito las letras RIBE entre corchetes. Esto quiere decir que estas letras son opcionales,

por lo que el comando se puede escribir como DESC, DESCR, DESCRI, DESCRIB y DESCRIBE.

Todos los comandos tienen esta propiedad y el texto obligatorio que escribir es aquel que lo distingue del resto de los comandos. Por ejemplo, el comando CREATE TABLESPACE se puede escribir CREATE TABLES, CREATE TABLESP, CREATE TABLESPA, CREATE TABLESPAC y CREATE TABLESPACE. Es necesario escribir hasta la S con el fin de distinguirlo de CREATE TABLE.

El comando CREATE TABLE permite crear una tabla donde se define cada una de sus columnas y restricciones. El siguiente comando crea una tabla llamada Tienda:

CREATE TABLE Tienda(

IdTienda NUMBER(3) PRIMARY KEY,

NombreTienda VARCHAR2(20) NOT NULL,

Direccion VARCHAR2(56) NOT NULL);

Para añadir, modificar o eliminar cualquier columna o cualquier restricción en una tabla que ya está creada, se usa el comando **ALTER TABLE**. Este comando tiene una serie de cláusulas para añadir, modificar y eliminar columnas y restricciones. El siguiente ejemplo añade a la tabla Tienda la columna Telefono:

```
ALTER TABLE Tienda ADD Telefono NUMBER(9) NOT NULL;
```

Para eliminar una tabla, se usa

```
DROP TABLE nombreTabla [CASCADE CONSTRAINTS];
```

Este comando elimina la tabla, sus restricciones y todos los registros que existen en ella. El siguiente ejemplo elimina la tabla Tienda y sus restricciones:

```
DROP TABLE Tienda CASCADE CONSTRAINTS;
```

Lenguaje de manipulación de datos (DML)

Está compuesto por los comandos que permiten la gestión de los datos almacenados en las estructuras definidas con los comandos DDL, es decir, gestionan los registros de las tablas. Los comandos más importantes de este bloque son INSERT, UPDATE y DELETE.

Para insertar un nuevo registro en una tabla, se usa

```
INSERT INTO Tabla[(Columnas)] VALUES(Valores).
```

El siguiente ejemplo inserta un registro en la tabla Tienda:

```
INSERT INTO Tienda(idTienda, NombreTienda, Direccion, Telefono) VALUES (1, 'AdyD Informatica', 'C/La Perla, 43, 28661', 913143243);
```

El comando UPDATE permite modificar un conjunto de registros que se encuentran en una tabla y su sintaxis es

```
UPDATE Tabla SET campo=valor {WHERE condicion};
```

El comando permite indicar a través de una cláusula WHERE aquellos registros que modificar. Ejemplo:

```
UPDATE Tienda SET telefono=919143243 WHERE idTienda=1;
```

Para eliminar un registro de una tabla, se usa el comando DELETE con la sintaxis

DELETE [FROM] Tabla [WHERE condicion];

La condicion permite indicar qué registros se deben eliminar. El siguiente ejemplo elimina la tienda con idTienda = 1

```
DELETE from Tienda WHERE idTienda = 1;
```

Lenguaje de consulta de datos (DQL)

Realmente este es un lenguaje incluido en el anterior pero lo separamos porque realmente es muy importante. El lenguaje de consulta de datos está compuesto principalmente por el comando **SELECT**.

Este comando permite obtener información de la base de datos, por lo que es, quizá, el comando que más se ejecute en la vida de una base de datos. Generalmente, son las aplicaciones que acceden a las bases de datos las que suelen tener en su código operaciones de consultas de datos. También el administrador de la base de datos realiza constantemente consultas al diccionario de datos para desarrollar sus actividades de administración. La sintaxis reducida de este comando es

SELECT {columnas[*]} FROM tablas [WHERE condicion] [ORDER BY columnal];

Después de SELECT, se indican las columnas que se quieren proyectar de las tablas que se indican con FROM. La cláusula WHERE permite indicar el filtrado en relación con los registros de interés. Este filtrado se indica con una condición que, registro a registro, debe cumplir para que sea seleccionado. Por ejemplo, la siguiente consulta proyecta todas las columnas y muestra todos registros de la tabla Tienda, ya que no se ha usado ninguna condición de filtrado. Además, se ordenan por la columna nombreTienda:

```
SELECT * FROM tienda ORDER BY nombreTienda;
```

Lenguaje de control de datos y transacciones (DCL y TCL)

El lenguaje de control de datos o Data Control Language (DCL) está compuesto por los comandos que permiten la gestión de la seguridad en el acceso a los datos de la base de datos. Principalmente, son **REVOKE, GRANT, CREATE USER, CREATE ROLE y CREATE PROFILE**.

El lenguaje de control de transacciones o TCL está compuesto por los comandos que permiten gestionar las transacciones, es decir, los grupos de operaciones vistas como acciones atómicas para la base de datos. Los comandos principales son COMMIT, ROLLBACK y SAVEPOINT. El comando COMMIT procesa todas las transacciones pendientes de actualizar y las lleva a las tablas reales. Esta información es leída desde los ficheros de actualización, donde se almacenan todas las transacciones de actualización pendientes, hasta los ficheros de datos, donde se almacenan todos los registros de un conjunto de tablas.

Características del SQL

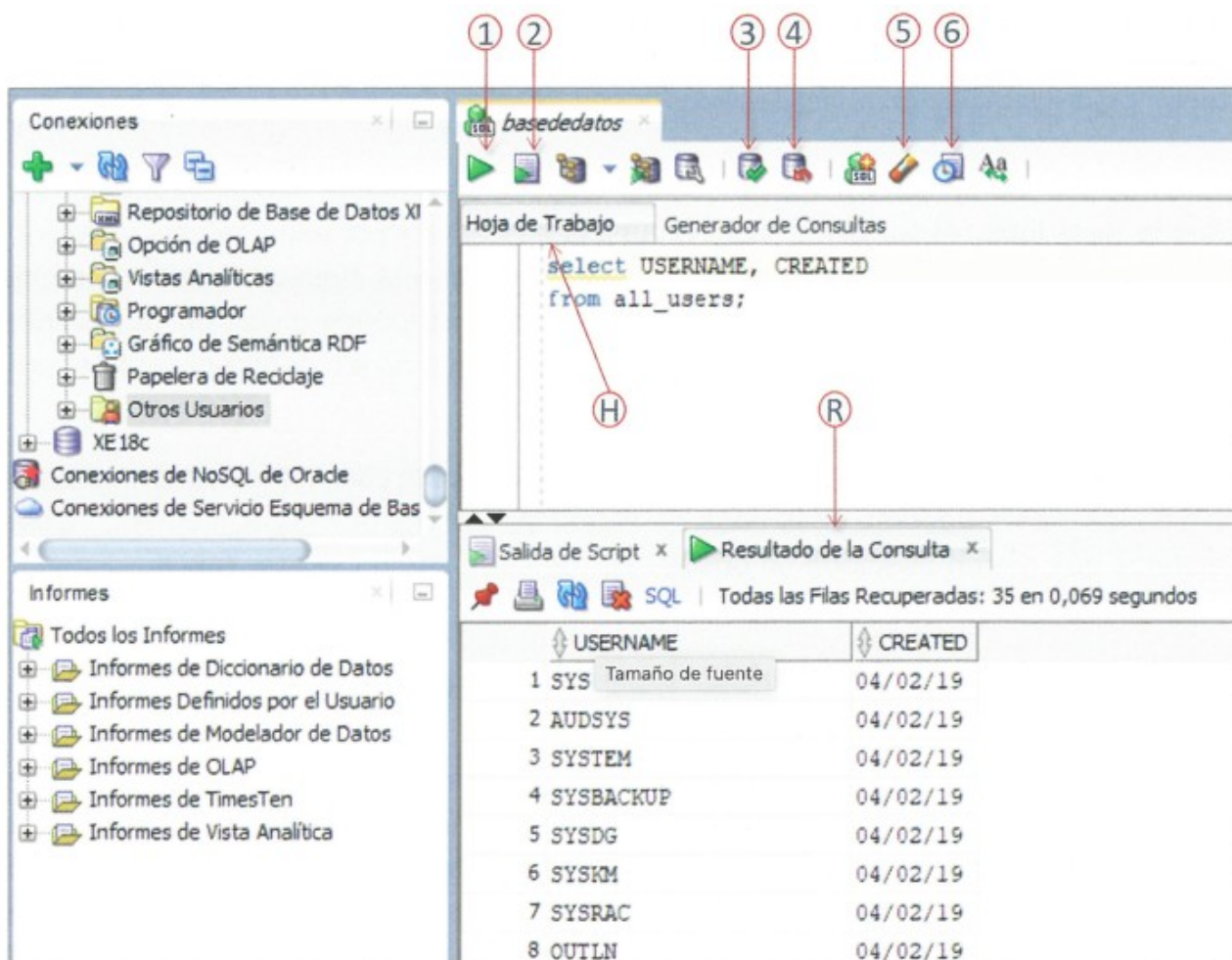
cuando se crea un objeto en el entorno gráfico de SQL Developer, este siempre dispone de alguna pestaña para obtener el comando asociado a la acción que se está llevando a cabo a través del entorno gráfico. Estos comandos pueden ser acumulados en un fichero plano con el fin de ejecutarlos para reproducir la creación de la misma base de datos en la misma máquina o en otra. Estos son los comandos que interpreta el motor de Oracle para gestionar la base de datos. La potencia de la ejecución de un comando en vez del uso del entorno gráfico tiene sus ventajas, sobre todo para la programación y la reproducción de gestiones ya efectuadas.

Como ejecutar comandos SQL

El conjunto de comandos escritos de forma ordenada y en un fichero plano se llama guión o script. De este modo, se pueden guardar los comandos que se ejecutan uno a uno en un único fichero para su posible ejecución en caso de necesitarlo. Por lo tanto, se entiende que los comandos se pueden ejecutar uno a uno o en su conjunto a través de un script.

Otro programa de Oracle que permite ejecutar comandos SQL es sqlplus. Este programa no dispone de entorno gráfico, sino que usa una consola de comandos en modo texto. El uso de SQL Developer para la ejecución de comandos y de scripts es muy cómodo. Veamos los elementos que permiten ejecutar los comandos SQL en SQL Developer.

La hoja de trabajo (H) permite ejecutar cualquier comando SQL. Para ejecutar un comando se pulsa el botón con un signo de play verde (1). Este botón ejecuta el comando donde está el cursor hasta el siguiente punto y coma. La misma acción tiene la pulsación a través del teclado usando las teclas Control e Intro a la vez. Sin embargo, para ejecutar todos los comandos de la hoja de trabajo, se pulsa el segundo botón con un signo play verde y una pequeña hoja de papel (2). Se debe prestar mucha atención y no ejecutar todo el conjunto de comandos en vez del último que se quiere ejecutar. Para borrar de la hoja de trabajo todos los comandos, se pulsa el icono goma (5) o Control y D.



Existen comandos en Oracle que se ejecutan directamente sobre la base de datos y no se pueden deshacer. Sin embargo, hay otros que si se pueden deshacer porque, en realidad, aún no se han hecho sobre la base de datos, sino que se ha registrado en un cuaderno de bitácoras. Estas acciones deben confirmarse o deshacerse. La acción **COMMIT** ejecuta en la base de datos todas las acciones pendientes y **ROLLBACK** las deshace dejando la base de datos en el estado en el que se encontraba en el último **COMMIT**. Los botones para hacer esto son Confirmar o tecla F11 (3) y Rollback o tecla F12 (4). El botón Historial SQL o tecla F8 (6) muestra todos los comandos ejecutados últimamente. En la sección Salida de Script o Salida de Comando (R), se muestra la salida que emite la ejecución del comando. Esta salida puede ser almacenada en un archivo con el icono Grabar o tecla Control+S. Para abrir un script .sql, se elige la opción Abrir en Archivo y se indica como tipo de archivo Script SQL (*.sql). El contenido del fichero se mostrará en la hoja de trabajo y se ejecutará pulsando el botón Ejecutar Script (2) o F5

Comandos con SQLPLUS

El programa sqlplus permite conectarse a un sistema gestor de bases de datos local o remoto y gestionar los objetos a través de una consola de comandos. Es posible ejecutar un script a través del comando start **nombrescript.sql**. La extensión de los scripts no tiene por que ser .sql, pero se recomienda para su organización.

Para ejecutar sqlplus, se abre Ejecutar o una consola de comando. Para ello, se pulsa la tecla Windows y R. En Ejecutar, se escribe cmd o directamente sqlplus. Si se ejecuta el comando cmd, se abre una consola de comandos.

Para ejecutar un script, solo hace falta conocer el path del directorio donde se encuentra el fichero .sql que se quiere ejecutar y usar el comando start. También funciona con la arroba. El path al script se pone entre comillas simples. También podemos arrastrar el script a la ventana y nos pone el path automáticamente.

```
SQL> start 'c:\oradata\scripts\DDLTiendaInformatica.sql'
```

```
SQL> @'c:\oradata\scripts\DDLTiendaInformatica.sql'
```

Cuando se escribe un comando SQL usando sqlplus, este puede ocupar varias líneas. Al pulsar la tecla Intro, el comando sigue en una línea nueva y por ellos sqlplus muestra el número de línea por el que va.

El procesador de sqlplus no ejecuta el comando hasta que no encuentra el carácter punto y coma, ;. Esto también ocurre para SQL Developer. Como se ha dicho antes, los comandos para deshacer es ROLLBACK. Para confirmar que las operaciones deben hacerse en la base de datos, se ejecuta el comando COMMIT. Los comandos DDL, como CREATE TABLE siempre son confirmados, no necesitan COMMIT y nunca se puede hacer rollback en ellos.

Ingeniería inversa con el DataModeler

A través de la herramienta DataModeler, se puede obtener el esquema relacional y también el conjunto de comandos para crear todos los objetos necesarios para la base de datos.

Una vez que se ha diseñado el modelo lógico, se pulsa el botón de ingeniería, obteniéndose el esquema relacional de la base de datos o modelo relacional. Una vez obtenido el esquema, pulsando el botón Obtener DDL, se obtiene el script DDL. Este script se llama así porque está compuesto por los comandos de definición de datos del tipo CREATE y ALTER. La ejecución de este script se puede llevar a cabo a través de SQL Developer o sqlplus.

No se debe ejecutar dos veces el mismo script sin eliminar previamente los objetos, ya que estos existen y no pueden ser creados de nuevo. Para eliminar una tabla, se usa el comando DROP TABLE nombreTabla CASCADE CONSTRAINTS. Se recomienda eliminar las restricciones asociadas a una tabla si se va a crear de nuevo, ya que los nombres de las restricciones provocarían un error, puesto que ya existen. Para ello, se usa en el comando DROP TABLE la cláusula CASCADE CONSTRAINTS.

Lo que suele hacerse es que los scripts de creación de la base de datos tengan el siguiente esquema:

- Sección DROP donde se borran las tablas

- Sección CREATE donde se crean las tablas en cualquier orden

- Sección ALTER donde se crean las constraints de las tablas

- Sección INSERT donde se insertan los datos precargados de las tablas

Convenciones de escritura de SQL

Cuando se escribe un comando en SQL, se debe tener en cuenta una serie de reglas de escritura que pueden afectar a su correcta ejecución. Todos los comandos terminan en un punto y coma. Hasta que el procesador del lenguaje no encuentra el símbolo ;, el comando no se ejecuta, lo haga individualmente o dentro de un script. Es muy importante conocer dónde se puede dividir un comando para escribirlo en varias líneas. El objetivo es escribir el comando lo más claro posible, ya que el procesador podría leer el comando incluso en una sola línea por muy extenso que sea este. Para ello, es importante conocer los conceptos de comando, cláusula y sección o subcláusula.

Un comando puede estar compuesto por un conjunto de cláusulas opcionales o no, que tienen diferentes partes llamadas sección. Algunas secciones son obligatorias y otras no dependiendo del comando. Una cláusula no debe escribirse en varias líneas, a menos que sea muy larga y existan mecanismos que permitan separarlas. Cada comando tendrá sus posibilidades.

Los comandos siguientes son todos equivalentes:

Ejemplo 1

```
select *  
from profesor  
where nombre='Dolores'
```

Ejemplo 2

```
select * from profesor  
where nombre='Dolores';
```

Ejemplo 3

```
Select * from profesor  
where nombre='Dolores'  
;
```

Los comandos se pueden escribir en minúsculas o en mayúsculas, pero hay que prestar mucha atención a los datos en sr'. Estos sí son diferentes si están escrito de una forma u otra. Por ejemplo, estos comandos son exactamente iguales:

```
CREATE TABLE ALUMNO(
```

```
    DNI NUMBER(8) PRIMARY KEY,
```

```
    NombreApellidos VARCHAR2(40) NOT NULL);
```

```
create table alumno(
```

```
    dni number(8) primary key,
```

```
    nombreaPELLIDOS varchar2(40) not null);
```

```
cReaTe Table ALUMNO(
```

```
    dni Number(8) Primary kEy,
```

```
    nOmbrEApEllIdOs vaRchaR2(40) nOt nUlL);
```

Incluso cuando se hace referencia al nombre del objeto, por ejemplo, la tabla Alumno, se puede escribir tanto en minúsculas como en mayúsculas. Esto tiene una excepción, cuando se accede al diccionario de datos, ya que los valores de las tablas y vistas que almacenan los metadatos están escritos en su mayoría en mayúsculas.

Las siguientes consultas son diferentes, ya que hacen referencia a valores distintos:

```
select * from profesor where nombre='lOlA';
```

```
select * from profesor where nombre='lola';
```

```
select * from profesor where nombre='Lola';
```

Los comandos para la definición de elementos de la base de datos no pueden ser anulados. Así, con los comandos CREATE, ALTER y DROP, no se puede usar el comando ROLLBACK para deshacerlos.

Tipos de datos

Los tipos de datos se usan para limitar el espacio que un campo necesita en memoria. Este tamaño se puede indicar a través una categoría, por ejemplo, número entero, o indicando los valores que puede tomar. Oracle permite declarar muchos tipos de datos, propios y de terceros con el fin de permitir compatibilidad con otros sistemas gestores de bases de datos. Los tipos de datos más importantes serán los numéricos, en sus diferentes tamaños; los alfanuméricos, compuestos por letras y números y que se denominan cadenas; los de tipo fecha y hora y otros especiales para contener datos de gran dimensión, como un documento XML, HTML, o incluso binarios de gran tamaño.

Tipos de datos en Oracle		
Tipo y descripción	Ejemplo	Observación
NUMBER(E,D) Puede contener tanto valores enteros como decimales. Se indica el número de dígitos que compone el número, E. Si tiene decimales, de ese número, con D se indica cuántos dígitos son decimales	<pre>CREATE TABLE Profesor(dni NUMBER(8), sueldoMensual NUMBER(8,2));</pre>	Donde dni puede tomar hasta 8 dígitos numéricos dígitos enteros, desde 0 hasta 999999999, sueldoMensual puede tener hasta 8 dígitos, de los cuales 2 son para la parte decimal
BINARY_FLOAT Puede contener valores numéricos de coma flotante de hasta 32 bits. Se usa para números con decimales y el rango es desde 1,17E-38F hasta 3,40+38F	<pre>CREATE TABLE Producto(CodProd NUMBER(5), precio BINARY_FLOAT);</pre>	El campo precio es un número decimal

Tipos de datos en Oracle		
Tipo y descripción	Ejemplo	Observación
BOOLEAN Puede ser TRUE, FALSE o NULL	<pre>CREATE TABLE Venta(codVenta NUMBER(10), fecha DATE, online BOOLEAN);</pre>	El campo online puede valor TRUE o FALSE
BLOB, CLOB, BFILE Datos de gran tamaño	<pre>CREATE TABLE Perfil(codPerfil NUMBER(10), nombre VARCHAR2(40), foto BLOB);</pre>	El campo foto puede contener un dato de gran tamaño

Tipos de datos en Oracle		
Tipo y descripción	Ejemplo	Observación
BINARY_DOUBLE El doble que BINARY_FLOAT, es decir, hasta 64 bits. Se usa cuando son número con más decimales	<pre>CREATE TABLE Inmueble(CodInmueble NUMBER(4), valorBursatil BINARY_DOUBLE);</pre>	El campo valorBursatil es un número decimal con el doble de tamaño que BINARY_FLOAT
VARCHAR2(N) Puede contener una cadena de hasta N caracteres alfanuméricos. Puede contener cualquier carácter imprimible de la tabla ASCII. El tamaño máximo es de 32 KB	<pre>CREATE TABLE Alumno(dni NUMBER(8), nombre VARCHAR2(20), prApellido VARCHAR2(25), sgApellido VARCHAR2(25));</pre>	El campo Nombre puede tener hasta 20 caracteres alfanuméricos y prApellido y sgApellido hasta 25
CHAR(N) Contiene siempre N caracteres alfanuméricos. Los no introducidos se rellenan con espacios en blanco. El tamaño máximo es de 2000 bytes	<pre>CREATE TABLE Entrenador(codEntr CHAR(4), dni NUMBER(8), letradni CHAR(1));</pre>	El campo codEntr tiene 4 caracteres siempre. Si tiene menos, se rellena con espacios. Es de longitud fija. El campo letradni sería de un carácter alfanumérico
LONG Admite cadenas de hasta 2 GB	<pre>CREATE TABLE Perfil(codPerfil NUMBER(10), nombre VARCHAR2(40), foto LONG);</pre>	El campo foto puede llegar a ocupar hasta 2 GB
DATE Almacena la fecha y hora	<pre>CREATE TABLE Precio(codProd NUMBER(5), fechaPrecio DATE, precio NUMBER(7,2));</pre>	El campo fechaPrecio puede contener una fecha, hora y más información sobre la fecha
TIMESTAMP[(n)] Almacena con N dígitos la fecha y hora	<pre>CREATE TABLE Precio(codProd NUMBER(5), fechaPrecio TIMESTAMP(16), precio NUMBER(7,2));</pre>	El campo fechaPrecio almacenará una fecha usando 16 dígitos
INTERVAL YEAR(N) TO MONTH(M) DAY TO SECOND(S) Puede ser año y mes con N y M dígitos o día a segundo con S dígitos	<pre>CREATE TABLE Profesor(id NUMBER(3), experiencia INTERVAL YEAR TO MONTH); CREATE TABLE Sesion(Id Sesion(8), tiempoConexionTotal DAY TO SECONDS);</pre>	Una inserción para el campo experiencia podría ser INTERVAL '3-9' YEAR(1) TO MONTH(1), para indicar 3 años y 9 meses. Una inserción para el campo tiempoConexionTotal sería INTERVAL '8 20:56:10' DAY TO SECOND para indicar 8 días 20 horas, 56 minutos y 10 segundos

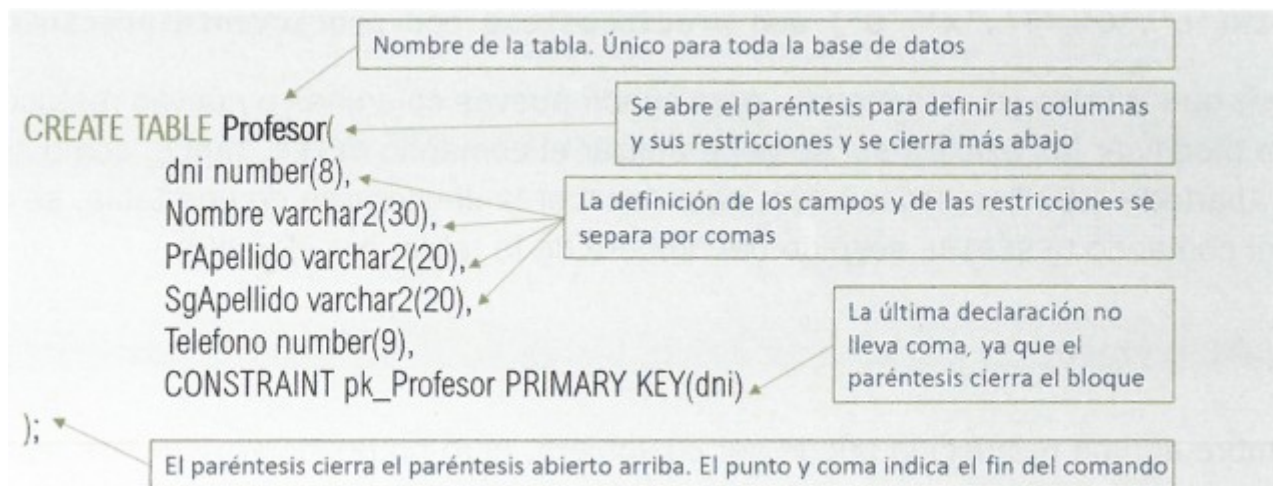
Create table

El comando CREATE TABLE permite crear una tabla indicando todas sus propiedades, ya sean atributos o restricciones asociadas a sus campos, y sus relaciones a través de sus claves ajenas. La sintaxis básica del comando es el siguiente:

```
CREATE TABLE [esquema.nombreTabla(  
    nombreColumna tipoDeDatos [DEFAULT valor] [clausula_restriccion],  
    ...  
    [clausula_restriccion]...  
)  
[TABLESPACE nombreEspacioTabla]  
[STORAGE (clausula_de_almacenamiento)];
```

La sintaxis del comando CREATE TABLE tiene más cláusulas, pero estas son las que se deben declarar para cualquier base de datos. Las cláusulas TABLESPACE y STORAGE se usan para gestionar el almacenamiento de las tablas en espacios de tablas y son opcionales.

Si no se indican la tabla tomará los valores por defecto definidos para el usuario o para la base de datos. Entre los paréntesis del comando, se indican, separados por comas, cada una de las columnas de las tablas y las restricciones sobre ellas. Las restricciones se pueden declarar en el mismo lugar donde se define la columna o definir las todas juntas después de la definición de todas las columnas, solo es cuestión de organización. En la definición de una restricción, se puede usar palabra CONSTRAINT con el fin de dar un nombre a la restricción. El uso de la palabra CONSTRAINT no es obligatorio, pero sí recomendable, ya que permite nombrar las restricciones para facilitar posibles gestiones que puedan surgir sobre ellas. Si no se le asigna un nombre, Oracle le asignará un número, lo que hace más difícil su edición posterior.



Se ha creado una tabla llamada Profesor y se han definido 5 columnas, dni de 8 dígitos numéricos, Nombre de 30 caracteres alfanuméricos, PrApellido y SgApellido de 20 caracteres alfanuméricos y Telefono de 9 dígitos numéricos. Tras definir las columnas, se ha definido la restricción clave primaria usando la cláusula **CONSTRAINT nombre PRIMARY KEY(columna)**.

Las restricciones que se definen son aquellas relacionadas con las columnas de la tabla en las que están. Dentro de una tabla, ninguna columna se puede llamar igual que otra, así, si hay dos claves ajenas que se llaman dni, por ejemplo, se hace necesario darles nombres diferentes. Las restricciones que se pueden declarar son las mismas que las estudiadas en la fase de diseño y son las siguientes:

- **PRIMARY KEY**: se indicarán entre paréntesis la columna o columnas que forma la clave primaria de la tabla. Solo se debe escribir una restricción PRIMARY KEY dentro de una tabla, aunque esta tenga más de una columna. La escritura de una restricción de clave primaria es **CONSTRAINT nombreRestriccion PRIMARY KEY(columna/s)**.
- **FOREIGN KEY**: se indicará una clave foránea por cada interrelación que apunte a la tabla. Si una clave foránea tiene más de una columna, se indicarán dentro del paréntesis. La escritura de una restricción de clave ajena es **CONSTRAINT nombreRestriccion FOREIGN KEY(columna/s) REFERENCES TablaALaQueApunta(columna/s)**.
- **NOT NULL**: se indicará para cada columna que es obligatoria. Como son tantas, generalmente, no se les da un nombre y se especifican en la definición de la columna.
- **UNIQUE**: se indicará con una sola restricción la columna o columnas que son atributos secundarios. Si hay demasiadas columnas, se puede declarar por separado. Se puede omitir también la palabra CONSTRAINT. La escritura de una restricción de clave única es **CONSTRAINT nombreRestriccion UNIQUE(columna/s)**.
- **CHECK**: esta restricción se usa para limitar los valores que puede tomar una o más columnas. La condición asociada a una restricción CHECK se evalúa en cada inserción de un registro en la tabla. Si los valores del registro no cumplen la condición, el registro no se inserta. Generalmente, se escribe una sola restricción CHECK con todas las condiciones, pero también se pueden separar si interesa. Un ejemplo de restricción CHECK es el siguiente: **CONSTRAINT ck_producto CHECK(Tipo IN('L','C','T','X','O') and precioCoste>9 and precioVenta>precioCoste)**.

Una vez que la tabla ya está creada, para añadir nuevas columnas o nuevas restricciones o para modificar las existentes, se debe utilizar el comando ALTER TABLE. Recuerdese que, para conocer la descripción de una tabla, se puede usar el comando DESCRIBE seguido del nombre de la tabla, por ejemplo:

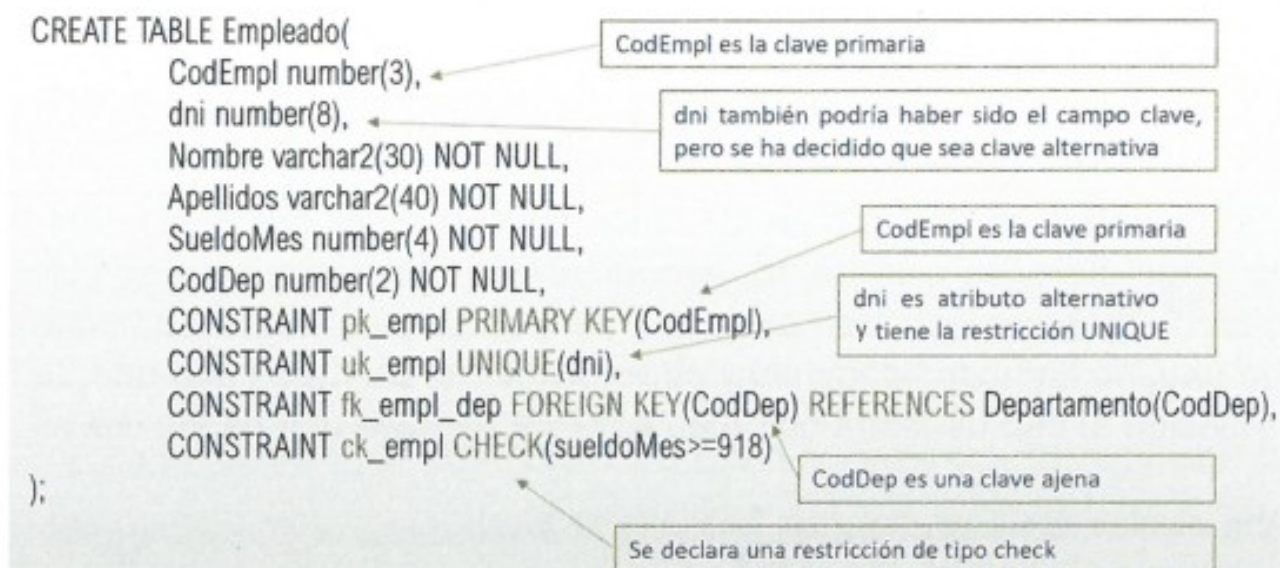
DESCRIBE Profesor

El nombre de una restricción puede ser cualquiera, pero no pueden existir dos restricciones con el mismo nombre. Se puede emplear una notación que muchos diseñadores de bases de datos usan y es utilizar dos letras para indicar el tipo de restricción y añadir el nombre de la tabla dueña de la restricción. Estas letras son pk para primary key, fk para foreign key, uk para unique y ck para check.

A continuación se crea la tabla empleado. En este caso, primero, se han declarado todas las columnas y, luego, sus restricciones usando la cláusula CONSTRAINT.

En esta tabla, la clave primaria es CodEmpl, tal como se indica en la restricción con nombre pk_empl. Es de tipo NUMBER(3), asique se estima que existirán como máximo 1000 registros en la tabla Empleado. Los códigos que tomará el campo clave irán desde 0 hasta 999.

El atributo dni también podría haber sido la clave primaria, pero se ha decidido que sea un atributo alternativo, por lo que tendrá la restricción UNIQUE, tal como aparece en la restricción con nombre uk_empl. La columna CodDep es para almacenar el código del departamento al que pertenece el empleado y se trata de una clave ajena que apunta a la tabla Departamento, tal como aparece en la restricción con nombre fk_empl_dep. Se ha añadido una restricción de chequeo para controlar que el campo SueldoMes nunca sea inferior a 918 euros. Esta restricción tiene el nombre ck_empl.



Las restricciones tipo NOT NULL deben declararse en la misma definición de la columna. En este caso los campos Nombre, Apellidos, SueldoMes y CodDep son obligatorios y tienen la restricción NOT NULL. En el caso de que fuesen opcionales, se escribiría NULL o no se indicaría nada, ya que este es el valor por defecto. Las columnas que tienen las restricciones PRIMARY KEY y UNIQUE tienen implícita la restricción NOT NULL, por eso no se ha indicado en las columnas CodEmpl y dni, aunque, si se declarase NOT NULL, no causaría ningún error.

Las restricciones se pueden declarar en el momento en el que se define cada columna siempre que una clave no esté compuesta por más de una columna. Además, igual que antes, se puede añadir la palabra CONSTRAINT para darle un nombre a la restricción. En el ejemplo se muestran dos comandos análogos para crear la tabla Empleado. En el primero, se declara la restricción en la definición de la columna, dándole nombre a la restricción. En el segundo, no se le asigna nombre, ya que no se usa la palabra CONSTRAINT.

```

CREATE TABLE Empleado(
    CodEmpl number(3) CONSTRAINT pk_empl PRIMARY KEY(CodEmpl),
    dni number(8) CONSTRAINT uk_empl UNIQUE(dni),
    Nombre varchar2(30) NOT NULL,
    Apellidos varchar2(40) NOT NULL,
    SueldoMes number(4) NOT NULL CONSTRAINT ck_empl CHECK(sueldoMes>=918),
    CodDep number(2) NOT NULL CONSTRAINT fk_empl_dep FOREIGN KEY(CodDep) REFERENCES Departamento(CodDep)
);

CREATE TABLE Empleado(
    CodEmpl number(3) PRIMARY KEY(CodEmpl),
    dni number(8) UNIQUE(dni),
    Nombre varchar2(30) NOT NULL,
    Apellidos varchar2(40) NOT NULL,
    SueldoMes number(4) NOT NULL CHECK(sueldoMes>=918),
    CodDep number(2) NOT NULL FOREIGN KEY(CodDep) REFERENCES Departamento(CodDep)
);

```

PRÁCTICA PROUESTA

SCRIPT DE CREACIÓN DE LA BASE DE DATOS I

Las siguientes son tablas de una base de datos relativa a una academia. Las tablas que se adjuntan están aisladas y, por tanto, aún no se ha usado ninguna restricción de clave ajena. Usando SQL Developer, crea cada una de las tablas a través de comandos. Para ello, crea un usuario llamado Academia y autentícalo con él. Crea cada una de las tablas una a una y decide el tipo de datos que más se ajusta a cada una de las claves primarias. Guarda el conjunto de comandos en un script de creación de la base de datos .sql

Profesor(P-**dni**, nombre, apellidos, direccion, fechaAlta, codDep)
 Alumno(P-**codAlumno**, nombre, apellidos, direccion)
 Asignatura(P-**codAsig**, nombre, numHoras, codCiclo)
 Ciclo(P-**codCiclo**, nombreCiclo, sigla)
 Departamento(P-**codDep**, nombre)

La foreign key

La restricción Foreign Key es la que permite establecer las conexiones entre las tablas. Se usa para indicar en una tabla cuál o cuáles de sus columnas comportan una clave ajena que apunta a otra tabla. Se define una cláusula FOREIGN por cada clave ajena que posea la tabla. Si una clave ajena está compuesta por más de una columna, se indicará en la restricción y nunca se usará más de una cláusula FOREIGN KEY para una clave ajena.

CONSTRAINT nombre_restriccion FOREIGN KEY(columnas) REFERENCES tabla[(columnas)] [ON DELETE CASCADE | ON DELETE SET NULL]

La palabra REFERENCES se usa para indicar cuál es la tabla a la que apunta la clave foránea.

Después del nombre de la tabla, de forma opcional, se puede indicar cada columna con el nombre usado en la clave primaria de la tabla que apunta, separados por coma y entre paréntesis. Esta se hace debido a que el nombre de las columnas de la clave ajena puede ser diferente al de la clave primaria.

Las opciones **ON DELETE CASCADE** y **ON DELETE SET NULL** se usan para definir las restricciones de integridad referencial. Estas restricciones se crean para controlar el borrado o puesta a nulo de una clave ajena cuando se elimina el registro al que hace referencia en la otra tabla. Por ejemplo:

ESTUDIANTE					MATRICULA		ASIGNATURA		
P dni	Nombre	prApellido	sgApellido	FechaNac	PF dni	PF codAsig	P codAsig	Nombre	numHoras
28910123	Antonio	Montero	Escudero	'03/02/2001'	28910123	1	1	Fundamentos de hardware	96
43910674	Maria del Carmen	Aragón	Sanz	'11/11/1992'	28910123	3	2	Lenguajes de marcas	128
98143987	Carolina	Blanco	Mesa	'08/05/1998'	28910123	4	3	Gestión de bases de datos	192
77845234	Dolores	Palacios	Borrero	'25/09/1994'	77845234	2	4	Planificación y administración de redes	224
85490003	Iván	Marín	Herrera	'14/02/2000'	77845234	3	5	Implantación de sistemas operativos	224
55894493	Miguel Ángel	Barea	Vidal	'19/01/2003'	77845234	5	6	Administración de sistemas operativos	160
77800392	Laura	Verde	Moreno	'01/08/1998'	43910674	3	7	Seguridad y alta disponibilidad	80

La tabla matricula relaciona los estudiantes y las asignaturas en las que están matriculados. En la asignatura Gestión de bases de datos, cuyo código es 3, se han matriculado los alumnos con DNI 28910123, 77845234 y 43910674. ¿Qué ocurriría con los registros que hay en Matricula si se eliminase dicha asignatura de la tabla Asignatura? Cuando se declara una clave ajena, se dispone de tres opciones en cuanto a la restricción de integridad referencial: no especificar ninguna restricción, indicar la restricción **ON DELETE CASCADE** o indicar la restricción **ON DELETE SET NULL**.

Si, junto a una clave ajena, no se indica ninguna restricción de integridad referencial, entonces, no es posible eliminar un registro cuya clave primaria está relacionada con una clave ajena. Para el ejemplo no se permitiría eliminar el registro Gestión de bases de datos, ya que su clave primaria, 3, está relacionada. Sin embargo, si se usa la restricción **ON DELETE CASCADE**, el registro si se elimina y, además, también se eliminan todos los registros donde exista relación.

ESTUDIANTE					MATRICULA		ASIGNATURA		
P dni	Nombre	prApellido	sgApellido	FechaNac	PF dni	PF codAsig	P codAsig	Nombre	numHoras
28910123	Antonio	Montero	Escudero	'03/02/2001'	28910123	1	1	Fundamentos de hardware	96
43910674	Maria del Carmen	Aragón	Sanz	'11/11/1992'	28910123	4	2	Lenguajes de marcas	128
98143987	Carolina	Blanco	Mesa	'08/05/1998'	77845234	2	4	Planificación y administración de redes	224
77845234	Dolores	Palacios	Borrero	'25/09/1994'	77845234	3	5	Implantación de sistemas operativos	224
85490003	Iván	Marín	Herrera	'14/02/2000'	77845234	5	6	Administración de sistemas operativos	160
55894493	Miguel Ángel	Barea	Vidal	'19/01/2003'			7	Seguridad y alta disponibilidad	80
77800392	Laura	Verde	Moreno	'01/08/1998'					

Si se usa **ON DELETE SET NULL**, se elimina el registro, pero sus relaciones se sustituyen por NULL.

ESTUDIANTE					MATRICULA		ASIGNATURA		
P dni	Nombre	prApellido	sgApellido	FechaNac	PF dni	PF codAsig	P codAsig	Nombre	numHoras
28910123	Antonio	Montero	Escudero	'03/02/2001'	28910123	1	1	Fundamentos de hardware	96
43910674	Maria del Carmen	Aragón	Sanz	'11/11/1992'	28910123	NULL	2	Lenguajes de marcas	128
98143987	Carolina	Blanco	Mesa	'08/05/1998'	28910123	4	4	Planificación y administración de redes	224
77845234	Dolores	Palacios	Borrero	'25/09/1994'	77845234	2	5	Implantación de sistemas operativos	224
85490003	Iván	Marín	Herrera	'14/02/2000'	77845234	NULL	6	Administración de sistemas operativos	160
55894493	Miguel Ángel	Barea	Vidal	'19/01/2003'	77845234	5	7	Seguridad y alta disponibilidad	80
77800392	Laura	Verde	Moreno	'01/08/1998'	43910674	NULL			

```
CREATE TABLE Estudiante(
    dni number(8),
    Nombre varchar2(20) NOT NULL,
    prApellido varchar2(20) NOT NULL,
    sgApellido varchar2(20),
    fechaNac DATE NOT NULL,
    CONSTRAINT pk_estudiante PRIMARY KEY(dni));

CREATE TABLE Asignatura(
    codAsig number(3),
    Nombre varchar2(30) NOT NULL,
    numHoras number(3) NOT NULL,
    CONSTRAINT pk_asig PRIMARY KEY(codAsig));
```

Para relacionar estas dos tablas, se usa la tabla Matricula. La primera forma de declarar la clave ajena es sin usar ON DELETE CASCADE/SET NULL.

```
CREATE TABLE Matricula(
    dni number(8),
    codAsig number(8),
    CONSTRAINT pk_matr PRIMARY KEY(dni, codAsig),
    CONSTRAINT fk_matr_est FOREIGN KEY(dni) REFERENCES Estudiante(dni),
    CONSTRAINT fk_matr_asig FOREIGN KEY(codAsig) REFERENCES Asignatura(codAsig)
);
```

Si quisiésemos poner la restricción ON DELETE CASCADE/SET NULL

```
CONSTRAINT fk_matr_est FOREIGN KEY(dni) REFERENCES Estudiante(dni) ON DELETE CASCADE,
CONSTRAINT fk_matr_asig FOREIGN KEY(codAsig) REFERENCES Asignatura(codAsig) ON DELETE SET NULL
```

PRÁCTICA PROPUESTA

SCRIPT DE CREACIÓN DE LA BASE DE DATOS II

Partiendo del script de la práctica anterior, usa las restricciones de FOREIGN KEY para completar el script.

PRÁCTICA RESUELTA

SCRIPT DE PLATAFORMA DE VIDEOJUEGOS ONLINE

Partiendo de la base de datos PLATAFORMA DE VIDEOJUEGOS ONLINE realiza el script de creación de la base de datos haciendo:

1. crea el usuario
2. conéctate con el usuario y realiza el script
3. Corre el script creando la base de datos.

PRÁCTICA PROPUESTA

SCRIPT DDL para el caso TIENDA DE MÓVILES

Para el caso de Tienda de móviles, crea el usuario TiendaMovil, autentícate con el y crea cada una de las tablas procedentes del esquema relacional.

Claves con más de una columna

Una clave primaria de una tabla que procede de un tipo de entidad puede estar formada por más de un campo. Por tanto, esta clave primaria puede ser clave ajena en otra tabla. Las tablas procedentes de entidades débiles tienen siempre esta situación. Cuando esta clave primaria se propaga a otra tabla para relacionarla, convirtiéndose en una clave ajena, esta debe tener el mismo número de columnas y tipos.



Supón este esquema relacional

Hotel(P-**codHotel**, Nombre, Direccion, precDesay, qMascotas, qAerop)

Habitacion(P-F-**CodHotel** -> Hotel, P-**codHab**, Superficie, Banio, Thab)

TipoCamas(P-**codTipoCamas**, Descripcion)

HabTipoCama(P-F-**codHotel**, P-F-**codHab** -> Habitacion, P-F-**codTipoCamas** -> TipoCamas)

Se tiene una tabla Habitacion que es débil de Hotel. Supóngase que no se dispone de más de 1000 hoteles en la base de datos, por lo que el tipo de datos del campo clave de Hotel puede ser NUMBER(3). Para codificar las habitaciones de un hotel, se va a utilizar el propio número del hotel añadiéndole el código del hotel. Las habitaciones se suelen numerar atendiendo a la planta en la que están y, para hoteles de más de diez plantas, se hace necesario usar cuatro dígitos para codificar la habitación más el código del hotel, por ejemplo, habitación 102 2303, que sería del hotel con número 102. Para esta base de datos, se puede crear el usuario Hotel.

Para crear la tabla Hotel, se ejecutaria el comando:

```
CREATE TABLE Hotel(  
    CodHotel NUMBER(3) CONSTRAINT pk_hotel PRIMARY KEY(CodHote1),  
    Nombre VARCHAR2(49) NOT NULL,  
    Direccion VARCHAR2(59) NOT NULL,  
    precDesay NUMBER(4,2) NOT NULL,  
    qMascotas BOOLEAN NOT NULL,  
    qAerop BOOLEAN NOT NULL  
);
```

Ahora, para crear Habitación, se debe añadir la restricción FOREIGN que apunta a la tabla Hotel. El comando sería:

```
CREATE TABLE Habitacion(  
    CodHotel NUMBER(3),  
    CodHab NUMBER(4),  
    Superficie NUMBER(3) NOT NULL,  
    Banio CHAR(1) NOT NULL,  
    THab CHAR(2) NOT NULL,  
    CONSTRAINT pk_habitacion PRIMARY KEY(CodHotel, CodHab),  
    CONSTRAINT fk_hab_hotel FOREIGN KEY(CodHotel) REFERENCES Hotel(CodHotel)  
);
```

El campo clave de esta tabla está compuesto por CodHotel que se añade desde la tabla Hotel, por lo que debe tener el mismo dominio, NUMBER(3). Ahora, se quieren registrar las posibles combinaciones de camas que permiten cada habitación. Se requiere crear las tablas TipoCamas y HabTipoCama. Esta última tabla es la que almacena, para cada habitación, sus posibles tipos de camas de todas las posibles que existen en tipoCamas. Para esta tabla, el campo clave son las dos claves primarias que une, es decir, codHotel y coHabitacion de la tabla Habitación y codTipoCamas de la tabla TipoCamas. Los tipos deben ser los mismos de las tablas que proceden. Los comandos para crear estas dos tablas son:

```
CREATE TABLE TipoCamas(  
    CodTipoCamas NUMBER(2) CONSTRAINT pk_tipocamas PRIMARY KEY(CodTipoCamas),  
    Descripcion VARCHAR2(39) NOT NULL  
);  
  
CREATE TABLE HabTipoCama(  
    CodHotel NUMBER(3),  
    CodHab NUMBER(4),  
    CodTipoCamas NUMBER(2),  
    CONSTRAINT pk_habtipocama PRIMARY KEY(CodHotel,CodHab,CodTipoCamas),  
    CONSTRAINT Fk_habtc_habitacion FOREIGN KEY(CodHotel,CodHab) REFERENCES  
Habitacion(CodHotel,CodHab),  
    CONSTRAINT fk_habtc_tipocama FOREIGN KEY(CodTipoCamas) REFERENCES  
TipoCamas(CodTipoCamas));
```

PRÁCTICA PROPUESTA

SCRIPT DDL para el caso TIENDA DE MÓVILES

Para el caso siguiente crea un usuario llamado CentroComercial y crea el script de creación de la base de datos y ejecútalo.

CComercial(P-codCC, Nombre, Direccion, numPl)

Seccion(P-F- **CodCC**->CComercial, P-codSeccion, nombre, planta)

Producto(P-F-(**CodCC**, **codSeccion**)->Seccion, P-codProd, EAN13, Descripcion)

Precio(P-F-(**CodCC**, **codSeccion**,**codProd**)->Producto, P-orden, precio, fecha)

Check

Las restricciones de validación se convierten en una herramienta muy útil para controlar el dominio de los valores de las columnas de una tabla. Estas restricciones solo pueden usar las columnas definidas dentro de una tabla y no combinando columnas de diferentes tablas. Para esto último, se debe programar un procedimiento o usar un disparador, TRIGGER.

La definición de una restricción de validación usa la palabra CHECK y, entre paréntesis, se declara la condición que deben cumplir las columnas de la tabla. Por ejemplo, para la tabla Hotel, se puede limitar el valor del campo precio del desayuno en el rango [5-50] escribiendo

```
CONSTRAINT ck_prDes CHECK(precDesay>=5 AND precDesay<=59)
```

Por ejemplo, el valor del campo THab, tipo de habitación, está en la lista explícita 'ES','SU','SJ','S','DE' escribiendo

```
CONSTRAINT ck_tHab CHECK(THab IN ('ES','SU','SJ','S','DE')).
```

La expresión que se evalúa entre paréntesis se denomina expresión lógico-relacional. Este tipo de expresiones están compuestas por una serie de operadores que permiten evaluar si una expresión es cierta o no. Estos operadores son los lógicos y los relacionales.

Las expresiones relacionales son aquellas que comparan valores para evaluar si son iguales, diferentes, mayores que, menores que, mayores o iguales que... Los símbolos que se usan para representar estas operaciones son =, !=, >, <, >=, <=

Operador relacional	Representación	Ejemplo
Igual a	=	Having count(*) = 10
Distinto de	!=, <>	Having count(*)!=0
Mayor que	>	Having count(*)>99
Menor que	<	Having count(*)<10
Mayor o igual que	>=	Having count(*)>=1000
Menor o igual que	<=	Having count(*)<=100

Las expresiones lógicas son aquellas que, tras su evaluación, devuelven un valor cierto, true, o un valor falso, false. También pueden contener valores lógicos, es decir, un dato puede contener en su interior un valor cierto o un valor falso. Los datos lógicos o binarios, también llamados datos booleanos, se pueden representar también a través de los números 0 y 1. Para operar con datos lógicos, se usa un modelo matemático llamado álgebra de Boole. Este modelo opera con expresiones lógicas y con expresiones relacionales. Los operadores lógicos que se usan son NOT, OR y AND, con sus equivalentes en español, NO, O e Y.

Operación	Resultado	Descripción	Ejemplo
NOT true	<i>False</i>	No verdadero es falso	NOT(edad>100) Será verdad cuando edad sea menor de 100
NOT false	<i>True</i>	No falso es verdadero	NOT(edad>100)
False OR False	<i>False</i>	Si algo es falso o algo es falso, todo es falso	a>5 or b<10 es falso cuando a <= 5 y b >= 10
False OR true	<i>True</i>	Si algo es falso o algo es verdadero, resulta todo verdad	A=1 or a=2 es verdad cuando o A es 1 o a es 2
True OR false	<i>True</i>	Si algo es verdadero o algo es falso, resulta todo verdad	B=1 or a=3 es verdad cuando o B = 1 o a = 3
True OR true	<i>True</i>	Si algo es verdadero o algo es verdadero, todo es verdad	A=1 or b=2 es verdad cuando o A = 1 o b = 2
False AND false	<i>False</i>	Si algo es falso y algo es falso, todo es falso	A=1 and b=2 es falso si b = 3
False AND true	<i>False</i>	Si algo es falso y algo es verdad, todo es falso	A=1 and b=2 es falso si A = 2
True AND false	<i>False</i>	Si algo es verdad y algo es falso, todo es falso	A=1 and b=2 es falso si b = 3
True AND true	<i>True</i>	Si algo es verdad y algo es verdad, todo es verdad	A=1 and b=2 es verdadero si A = 1 y si b = 2

- Operador **NOT**: también se escribe usando el simbolo ! El operador NOT devuelve un valor false si su contenido es cierto y devuelve true cuando su contenido es falso. Por ejemplo, la expresión NOT(codigo=10) devuelve true siempre que codigo no sea igual a 10, lo que equivale a codigo!=10.
- Operador **OR**: el operador OR devuelve cierto siempre que alguna de sus dos expresiones sea cierta. Por ejemplo, la expresión (codigo=10 OR codigo=20) devuelve true siempre que codigo sea 10 o codigo sea 20. Será false si no vale ni 10 ni 20.
- Operador **AND**: el operador AND solo devuelve true cuando todas sus expresiones son ciertas. La expresión codigo=5 AND nombre='Antonio' AND tipo='C' solo es verdad si las tres expresiones relacionales son ciertas, es decir, cuando código vale 5, nombre contiene 'Antonio' y tipo es igual a 'C'. Si algunos de estos valores no son ciertos, toda la expresión es falsa.
- Operador **BETWEEN**: este operador permite comprobar si un valor está comprendido en un rango, por ejemplo, edad BETWEEN 18 AND 59.
- Operador **IN**: este operador comprueba si un valor está en una lista de valores, por ejemplo, IVA IN (4,19,21) o también ciudad IN ('Sevilla', 'Barcelona', 'Madrid', 'Valencia', 'Bilbao').

Cuando los operadores lógicos se anexan, se debe tener en cuenta que el orden de las expresiones si importa. Al igual que en aritmética, el cálculo $5 + 4 \times 6$ no es lo mismo que $(5 + 4) \times 6$, en lógica, también se usa paréntesis para agrupar cálculos lógicos. Así, la expresión $\text{numero1}=10 \text{ OR } \text{numero1}=29 \text{ AND } \text{numero2}=39$ es cierta cuando el numero1 es 10 o el numero1 es 20 y además el numero2 es 30. La expresión $(\text{numero1}=10 \text{ OR } \text{numero1}=29) \text{ AND } \text{numero2}=39$ es cierta cuando numero2 es 30 y el numero1 es 10 o 20.

Por ejemplo, para la tabla Habitación que se ha creado anteriormente, se podrían haber definido las dos restricciones de chequeo siguientes:

```
CREATE TABLE Habitacion(  
  CodHotel NUMBER(3),  
  CodHab NUMBER(4),  
  Superficie NUMBER(3) NOT NULL,  
  Banio CHAR(1) CONSTRAINT ck_banio CHECK(Banio IN ('D','B','A')),  
  THab CHAR(2) CONSTRAINT ck_tipoHab CHECK(THab IN ('ES','SU','SJ','S','DE')),  
  CONSTRAINT pk_habitacion PRIMARY KEY(CodHotel, CodHab),  
  CONSTRAINT fk_hab_hotel FOREIGN KEY(CodHotel) REFERENCES Hotel(CodHotel)  
);
```

Con esta restricción sobre la columna Banio, se define que solo puede tomar el valor D, B o A, donde se podría codificar con D el baño con ducha, B con bañera, A con ambos.

Para la columna Thab, se define el tipo de habitación codificando con ES, una habitación estándar, con SU, una habitación superior, con SJ una suite junior, con S una suite y con DE una habitación deluxe. Así, con la restricción check, se han limitado los valores que pueden tomar estos dos campos. En caso de una inserción en la tabla con valores diferentes para algunas de las dos columnas, esta será rechazada.

La restricción check se podría haber creado sin nombre sin usar la cláusula CONSTRAINT, de la forma:

```
CREATE TABLE Habitacion(  
  CodHotel NUMBER(3),  
  CodHab NUMBER(4),  
  Superficie NUMBER(3) NOT NULL,  
  Banio CHAR(1) CHECK(Banio IN ('D','B','A')),  
  THab CHAR(2) CHECK(THab IN ('ES','SU','SJ','S','DE')),  
  CONSTRAINT pk_habitacion PRIMARY KEY(CodHotel, CodHab),  
  CONSTRAINT fk_hab_hotel FOREIGN KEY(CodHotel) REFERENCES Hotel(CodHotel));
```

O se podría haber escrito en una sola línea check, como se ejemplifica a continuación:

```
CREATE TABLE Habitacion(  
  CodHotel NUMBER(3),  
  CodHab NUMBER(4),  
  Superficie NUMBER(3) NOT NULL,  
  Banio CHAR(1),  
  THab CHAR(2),  
  CONSTRAINT pk_habitacion PRIMARY KEY(CodHotel, CodHab),  
  CONSTRAINT ck_habitacion CHECK(Banio IN ('D','B','A') AND Thab IN ('ES','SU','SJ','S','DE')));
```


PRÁCTICA RESUELTA

SCRIPT DEL CASO TIENDA DE INFORMÁTICA

Partiendo de la base de datos diseñada antes y con el fin de compararla con el que se obtuvo usando el SQLDeveloper, se crea una cuenta de usuario TiendaInformatica, así como el script de creación de la base de datos.

PRÁCTICA PROPUESTA

SCRIPT DEL CASO REVISTA DE VIAJES

Para el diagrama conceptual que se ha obtenido en la actividad propuesta de la REVISTA DE VIAJES, escribe cada uno de los comandos para crear las tablas y sus restricciones, incluye todos los comandos en un script con el nombre de RevistaDDL.sql. Usando el SQLDeveloper crea la cuenta de usuario Revista con contraseña revista. Autentícate con esa cuenta y ejecuta el script.

Generalización y reflexión

En las estructuras asociadas a interrelaciones reflexivas y en las de una generalización, se deben tener en cuenta algunos aspectos de diseño físico. Para las reflexivas, se debe tener en cuenta que no se puede añadir una clave ajena a una tabla que aún no está creada. Para la generalización, se deben estudiar los valores que puede tomar el campo discriminador atendiendo si existe totalidad o parcialidad y exclusividad o solapamiento.

Las relaciones reflexivas

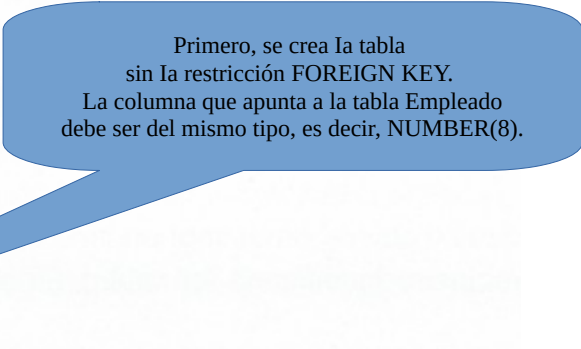
Cuando se crean las tablas procedentes de interrelaciones reflexivas, se deben tener en cuenta ciertas consideraciones. Como la tabla aún no está creada, no se puede definir la clave ajena hacia ella. En estos casos, primero, se crea la tabla con las columnas que proceden y, a continuación, se añade la restricción FOREIGN KEY a la columna o columnas concernientes con la interrelación reflexiva. Continuando con la base de datos del caso «Tienda de informática», supóngase que se quieren registrar aquellos empleados que son jefes. Para ello, se añade una columna dniJefe a la tabla Empleado para almacenar el DNI del jefe de cada uno. Aquellos que sean jefes tendrán un valor NULL en dicha columna.

Dicha columna debe estar puesta a NULL, en caso contrario, no se podría insertar ningún registro. En caso de que dicha columna por requisitos que cumplir deba estar a NOT NULL, se hace necesario crear una tabla nueva como si fuera una interrelación N:M.

La tabla que habría que crear tendría la siguiente descripción:

Empleado(P-dniEmpl, Nombre, prApellido, N-sgApellido, Cuenta, Domicilio, Telefono, Correo, F-dniJefe -> Empleado)

```
CREATE TABLE Empleado(  
    dniEmpl NUMBER(8),  
    Nombre VARCHAR2(20) NOT NULL,  
    prApellido VARCHAR2(20) NOT NULL,  
    sgApellido VARCHAR2(20),  
    Cuenta NUMBER(20) NOT NULL,  
    Domicilio VARCHAR2(60) NOT NULL,  
    Telefono NUMBER(9) NOT NULL,  
    Correo VARCHAR2(128) NOT NULL,  
    DniJefe NUMBER(8),  
    CONSTRAINT pk_empleado PRIMARY KEY(dniEmpl),  
    CONSTRAINT fk_empl_jefe FOREIGN KEY(dniJefe) REFERENCES Empleado(dniEmpl));
```



Primero, se crea la tabla sin la restricción FOREIGN KEY. La columna que apunta a la tabla Empleado debe ser del mismo tipo, es decir, NUMBER(8).

A continuación, se usaría el comando ALTER TABLE para modificar la tabla y añadir la restricción que falta. Este comando se estudia en el apartado siguiente. El comando sería:

```
ALTER TABLE Empleado ADD CONSTRAINT fk_empl_jefe FOREIGN KEY(dniJefe)  
REFERENCES Empleado(dniEmpl);
```

PRÁCTICA PROPUESTA

SCRIPT DEL CASO EQUIPOS DE PROGRAMADORES

Obtén el script DDL para el diagrama que se ha obtenido en la práctica de EQUIPOS DE PROGRAMADORES. Crea un usuario llamado pogramadores para crear esta base de datos. Añade las restricciones que consideres oportunas.

Generalizaciones

Con las tablas de una generalización, se deben tener en cuenta ciertos aspectos técnicos, sobre todo, en relación con el campo discriminador.

Cuando la generalización es incompleta, es decir, puede ocurrir que un registro que está en el supertipo no tenga ninguna relación con ninguno de los subtipos, el valor discriminador puede tomar el valor NULL. También se puede codificar con otro valor, pero generalmente se hace con NULL. Este valor NULL indicaría que se trata de una entidad que no es de ningún tipo de los subtipos posibles.

A cada uno de los posibles subtipos se les indica con un valor del discriminador. Estos valores suelen codificarse con caracteres y están asociados siempre a una restricción CHECK en la que se incluyen todos los valores posibles. Si son demasiados, se puede usar una tabla que contenga los valores explícitos del campo discriminado. Esto se hace si se prevé que en un futuro pueden existir diferentes subtipos a los iniciales, aunque se debe evitar en la medida de lo posible.

Cuando una generalización es exclusiva, no existen más complicaciones, pero, cuando existe solapamiento, se debe tener en cuenta que existen nuevos valores para el discriminador. Estos nuevos valores resultan de la posibilidad que existe que una entidad pertenezca a más de un subtipo, por lo que es necesario realizar todas las combinaciones posibles o incluir más de un campo discriminador.

Siguiendo con la base de datos del caso «Tienda de informática», supóngase que las tablas Empleado y Cliente se quieren especializar en un supertipo llamado Persona en el que se incluyen los campos comunes. Para añadir alguna estructura más, supóngase que se quiere almacenar no solo un teléfono y un correo, sino varios de estos. Este nuevo requisito modificaría las tablas Empleado y Cliente desde las descripciones:

Empleado(P-**dniEmpl**, Nombre, prApellido, N-sgApellido, Cuenta, Domicilio, Telefono, Correo)
Cliente(P-**dniCl**, Nombre, prApellido, N-sgApellido, Domicilio, Telefono, Correo)

hasta la siguientes tablas:

Persona(P-**codPersona**, dni, Nombre, prApellido, N-sgApellido, Domicilio, C-Tipo)
Telefono(P-**Telefono**, F-**codPersona->Persona**)
Correo(P-**Correo**, F-**codPersona->Persona**)
Empleado(P-**F-codPersona->Persona**, cuenta)
ClientelP-**F-codPersana -> Persona**)

Para este caso, que Persona es, o Empleado, o Cliente, y se trata de una generalización total, el valor del discriminador es fácil. La restricción check que añadir a Persona es CHECK(Tipo IN ('E','C')). Se ha codificado con los caracteres E para un registro que es empleado y C para un registro que es cliente. Aquellas tablas en las que exista alguna clave ajena que apunta a Empleado o a Cliente deben ser modificadas.

En este caso, que para la tabla Cliente no existe más campo que su identificador, se puede tomar una nueva consideración técnica: eliminar la tabla y, cuando se haga referencia a ella, hacerla a su supertipo añadiendo un procedimiento TRIGGER que se asocie a eventos y que se cree para controlar que el registro que se selecciona como cliente deba tener el valor C en su campo Tipo. De esta manera, se disminuye la complejidad de la estructura de tablas asociadas a la generalización. Se evaluará si el procedimiento TRIGGER consume tiempo que pudiera disminuir la eficiencia temporal de las operaciones asociadas a las tablas de la generalización. Generalmente, se gana eficiencia temporal al cruzar las tablas, ya que existen menos tablas. De cualquier forma, debe ser evaluado en fases posteriores a través de técnicas asociadas a auditorías y monitorización de tablas.

El esquema relacional de la nueva base de datos tendría los siguientes cambios:

Persona(P-**codPersona**, dni, Nombre, prApellido, N-sgApellido, Domicilio, C-Tipo)
Telefono(P-**Telefono**, F-codPersona -> Persona)
Correo(P-**Correo**, F-codPersona -> Persona)
Empleado(P-F-**codPersona** -> Persona, Cuenta)

Y la tabla Venta quedaría:

Venta(P-**CodVenta**, fechaHora, F-codEmpl -> Empleado, F-codCl -> Persona)

Esta tabla tiene un TRIGGER cuya condición es Venta.codCl = Persona.codPersona AND Persona.Tipo='C', que se debe interpretar como que el código de la persona que se introduce en la columna codCl debe tener en su campo tipo el valor C.

La tabla Compra queda como sigue:

Compra(P-**CodCompra**, fechaHora, F-codPersona -> Empleado, F-CodProv -> Proveedor)

El nombre de la columna en la clave ajena puede cambiar, sobre todo, cuando existen más de una clave ajena que apunta a la misma tabla, como ocurre en la tabla Venta y Compra. Por esta razón, también se puede cambiar el nombre de las claves primarias de los subtipos y que apunten al supertipo. El siguiente cambio hace que la gestión de estas columnas sea más cómoda:

Persona(P-**codPersona**, dni, Nombre, prApellido, N-sgApellido, Domicilio, C-Tipo)
Telefono(P-**Telefono**, F-codPersona → Persona)
Correo(P-**Correo**, F-codPersona → Persona)
Empleado(PF-**codEmpl** → Persona, Cuenta)
Venta(P-**CodVenta**, fechaHora, F-codEmpl → Empleado, F-codCl → Persona)
Compra(P-**CodCompra**, fechaHora, F-codEmpl → Empleado, F-CodProv → Proveedor)

Para crear la tabla persona:

```
CREATE TABLE Persona(  
    codPersona NUMBER(4),  
    dni NUMBER(8) NOT NULL,  
    Nombre VARCHAR2(20) NOT NULL,  
    PrApellido VARCHAR2(20) NOT NULL,  
    SgApellido VARCHAR2(20),  
    Tipo CHAR(1) NOT NULL,  
    CONSTRAINT pk_persona PRIMARY KEY(codPersona),  
    CONSTRAINT ck_person CHECK(Tipo IN ('C','E'))  
);
```

Para las tablas nuevas Telefono y Correo, que se han creado para poder almacenar más de un teléfono y correo para los empleados y los clientes, su comando de creación queda como sigue:

```
CREATE TABLE Telefono(  
    Telefono NUMBER(9),  
    codPersona NUMBER(4) NOT NULL,  
    CONSTRAINT pk_telefono PRIMARY KEY(Telefono),  
    CONSTRAINT fk_telefono_persona FOREIGN KEY(codPersona) REFERENCES  
Persona(codPersona)  
);
```

```
CREATE TABLE Correo(  
    Correo VARCHAR2(128),  
    codPersona NUMBER(4) NOT NULL,  
    CONSTRAINT pk_correo PRIMARY KEY(Correo),  
    CONSTRAINT fk_correo_persona FOREIGN KEY(CodPersona) REFERENCES  
Persona(codPersona)  
);
```

Ambas claves ajenas se consideran NOT NULL, ya que, para registrar el teléfono o el correo, es obligatorio indicar el código de su propietario a través de codPersona. Para la tabla Empleado, el comando queda como sigue. Se debe prestar atención al nombre de la columna clave ajena en Empleado y el nombre que tiene en la tabla a la que apunta. En la tabla Empleado, se llama codEmpl, pero, realmente, apunta a codPersona de la tabla Persona. El tipo debe ser el mismo que este, es decir, NUMBER(4).

```
CREATE TABLE Empleado(  
    codEmpl NUMBER(4),  
    cuenta NUMBER(20) NOT NULL,  
    CONSTRAINT pk_empleado PRIMARY KEY(codEmpl),  
    CONSTRAINT fk_empleado_persona FOREIGN KEY(codEmpl) REFERENCES  
Persona(codPersona)
```

Para la tabla Venta, se continúa con los mismos criterios

```

Create table Venta(
    codVenta NUMBER(7),
    fechaHora DATE NOT NULL,
    codEmpl NUMBER(4) NOT NULL,
    codCL NUMBER(4) NOT NULL,
    CONSTRAINT pk_venta PRIMARY KEY(codVenta),
    CONSTRAINT fk_venta_empl FOREIGN KEY(codEmpl) REFERENCES
Empleado(CodEmpl),
    CONSTRAINT fk_Venta_cliente FOREIGN KEY(codCL) REFERENCES
Persona(CodPersona)
);

```

Los campos codEmpl y codCl se han definido como NOT NULL porque, para cualquier venta, es Obligatorio indicar el código del empleado que hace la venta y el código del cliente que compra.

Para la tabla Compra, es análogo. El comando es como sigue:

```

CREATE TABLE Compra(
    codCompra NUMBER(7),
    fechaHora DATE NOT NULL,
    codEmpl NUMBER(4) NOT NULL,
    codProveedor NUMBER(3) NOT NULL,
    CONSTRAINT pk_compra PRIMARY KEY(codCompra),
    CONSTRAINT fk_compra_empl FOREIGN KEY(codEmpl) REFERENCES
Empleado(CodEmpl),
    CONSTRAINT fk_compra_prov FOREIGN KEY(codProveedor) REFERENCES
Proveedor(codProveedor)
);

```

PRÁCTICA PROPUESTA

SCRIPT DEL CASO VENTA DE VEHÍCULOS

Creando el usuario Concesionario, realiza el script y ejecútalo, para el caso de VENTA DE VEHÍCULOS. Añade las restricciones que creas oportunas.

Alter table

Para modificar un objeto que ya está creado, se usa el comando ALTER. Este comando es global para cualquier tipo de Objeto, es decir, si se quiere modificar una tabla ya creada, se usa **ALTER TABLE**; si es un usuario ya creado, se usaría ALTER USER; si es un espacio de tablas ya creado, sería ALTER TABLESPACE, y asisucesivamente. Esto se expande cuando se quiere eliminar un objeto ya creado, usando para ello el comando DROP. Asi, DROP TABLE elimina una tabla ya creada, DROP USER un usuario ya creado o DROP TABLESPACE un espacio de tablas ya creado. Estos comandos, que pertenecen al lenguaje de definición de datos, Data Definition Language o DDL, no tienen vuelta atrás, es decir, no permiten la acción ROLLBACK.

Cuando se modifican las estructuras de las bases de datos, es importante analizar cómo están los datos almacenados en las tablas y, si es posible, modificarlas sin que les afecten, en cuyo caso, será necesario realizar procesos de exportación e importación posterior. La alteración de la descripción de una tabla debe hacerse, en la medida de lo posible, antes de incluir algún registro en las tablas que correspondan. Si las tablas contienen datos y se quieren añadir nuevas columnas obligatorias, se hace necesario algún proceso técnico anterior. Las modificaciones de columnas no obligatorias son sencillas, ya que no acarrear gran problema. Cuando se añade una clave ajena a una tabla que ya contiene datos, también se debe prestar atención. Una clave ajena que se añade se debe crear de tal forma que acepte nulos, ya que, de lo contrario, no sería válida su modificación porque, para la nueva columna, aún no habría datos. Estas y otras consideraciones deben llevarse a cabo con alto detenimiento para no acarrear pérdidas o modificaciones indeseadas.

Cuando se añaden nuevas columnas, así como nuevas filas, estas no se insertan entre columnas o filas ya existentes, sino que siempre se añaden al final. En caso de que se quiera colocar una columna nueva en una posición que no sea la final, se hace necesario llevar este proceso a través de una serie de comandos intermedios.

Añadir, eliminar o modificar columnas de la tabla

Para modificar una tabla ya creada cuando se quiere añadir una columna, redefinir una columna ya creada o eliminarlas, se usa el comando ALTER TABLE. Existe otro comando que se usa para modificar los registros de una tabla, el comando UPDATE. No se deben confundir estos dos comandos, ya que ALTER TABLE modifica la descripción de una tabla ya creada, mientras que, UPDATE modifica uno o más registros del interior de una tabla. La sintaxis del comando ALTER TABLE para las columnas es:

```
ALTER TABLE tabla {ADD columnaNueva Definicion  
| MODIFY nombreColumnaYaExiste DefinicionNueva  
| DROP columna  
| RENAME [COLUMN] nombreExiste TO nombreNuevo};
```


Cuando se añade una nueva columna, el nombre de esta no debe existir en la tabla donde se añade. Para ello, se usa la cláusula ADD seguida del nombre de la columna y su definición. La definición de esta columna es exactamente igual que cuando se define en el comando CREATE TABLE.

Por ejemplo, se ha observado que en la base de datos del caso «Tienda de informática», las tablas Venta y Compra no contienen un campo llamado Cantidad. Sin este campo, si el cliente compra el mismo producto diez veces, se hace necesario introducir diez registros con el código de la venta y el código del producto. Como esto no es posible, ya que el campo clave se repetiría, esta estructura obliga a que se tenga que hacer una nueva venta con un código nuevo para cada producto que se repite.

Se quiere, por tanto, añadir una nueva columna a ambas tablas. Este comando quedaria como sigue:

```
ALTER TABLE LineaVenta ADD Cantidad NUMBER(2) NOT NULL CHECK(Cantidad>0);  
ALTER TABLE LineaCompPa ADD Cantidad NUMBER(2) NOT NULL CHECK(Cantidad>0);
```

Se recomienda repetir toda la declaración, ya que dependiendo del sistema gestor, la columna podría tener una nueva definición. Para consultar información sobre las restricciones, se puede usar las columnas SEARCH_CONDITION, TABLE_NAME, CONSTRAINT_NAME y CONSTRAINT_TYPE de la vista del diccionario de datos llamada DBA_CONSTRAINTS. La consulta podría ser la siguiente:

```
SELECT SEARCH_CONDITION, TABLE_NAME, CONSTRAINT_NAME, CONSTRAINT_TYPE FROM  
DBA_CONSTRAINTS WHERE TABLE_NAME='LINEAVENTA';
```

Para eliminar una o más columnas, operación con la que se debe tener mucho cuidado, se usa la cláusula DROP en ALTER TABLE. Los siguientes comandos eliminan la columna Cantidad en las tablas Venta y Compra:

```
ALTER TABLE LineaVenta DROP Cantidad;  
ALTER TABLE LineaCompra DROP Cantidad;
```

Si la columna que se elimina tuviera datos, la columna quedaria totalmente eliminada. Cuando se modifica una columna que es clave primaria, esto conlleva consecuencias en las claves foráneas. Dichos cambios solo podrán ser efectuados si los datos que están en la columna que es clave ajena cumplen las nuevas condiciones de la clave primaria. Si se aumenta el número de dígitos, también se debe hacer en todas las tablas que estén como clave ajena. Si se disminuye o se cambia su tipo, los datos contenidos en las tablas deben ser compatibles con la nueva definición, de lo contrario, la operación no se llevará a cabo. En estos casos, se recomienda modificar primero las claves ajenas y sus datos y luego las claves primarias. También puede ser necesario desarrollar procedimientos extras para volcar estos datos en objetos auxiliares para una actualización posterior. Por ejemplo, supóngase que, para la base de datos del caso «Tienda de informática», se quiere ampliar el tamaño de los campos claves de Compra y Venta en dos dígitos más. Los comandos que ejecutar serían, para cambiarla clave ajena en las tablas LineaCompra y LineaVenta, los siguientes:

```
ALTER TABLE LineaCompra MODIFY CodCompra NUMBER(9);  
ALTER TABLE LineaVenta MODIFY CodVenta NUMBER(9);
```

Para las claves primarias de Compra y Venta, serían los siguientes:

```
ALTER TABLE Compra MODIFY CodCompra NUMBER(9);  
ALTER TABLE Venta MODIFY CodVenta NUMBER(9);
```

Añadir eliminar o modificar Constraints

En estos casos, aparece la utilidad de la cláusula CONSTRAINT en los comandos CREATE TABLE. Cuando se crea una restricción sin nombre, es decir, sin la palabra CONSTRAINT, Oracle le da un número, siendo necesario su uso cuando se quiere modificar o eliminar. Para hacer esta tarea más sencilla, lo que se recomienda es dar un nombre coherente a cada una de las restricciones de una tabla. Estos nombres deben ser únicos para toda la base de datos.

Las cláusulas del comando ALTER TABLE para gestionar las restricciones son exactamente iguales que las que se han usado para las columnas, con la salvedad de que a cada una de ella se le añade la palabra CONSTRAINT. De este modo, el comando ALTER TABLE acepta las cláusulas ADD CONSTRAINT, MODIFY CONSTRAINT y DROP CONSTRAINT para la gestión de las restricciones de una tabla. La sintaxis del comando ALTER TABLE para esta gestión es:

```
ALTER TABLE tabla {ADD CONSTRAINT restriccionNueva  
| MODIFY CONSTRAINT nombreRestriccionYaExiste DefinicionNueva  
| DROP CONSTRAINT nombreRestrccion  
| RENAME nombreExiste TO nombreNuevo};
```

Supóngase que la tabla Empleado se ha creado con el siguiente comando:

```
CREATE TABLE Empleado(  
    codEmpl NUMBER(4),  
    cuenta NUMBER(20) NOT NULL,  
    CONSTRAINT pk_empleado PRIMARY KEY(codEmpl),  
    CONSTRAINT fk_empleado_persona FOREIGN KEY(codEmpl) REFERENCES  
Persona(codPersona)  
);
```

También se quiere añadir el código del jefe de cada uno. Para ello, se añade la columna dniJefe y la restricción fk_empleado_jefe con los comandos siguientes:

```
ALTER TABLE Empleado ADD dniJefe NUMBER(4);  
ALTER TABLE Empleado ADD CONSTRAINT Fk_empleado_jefe FOREIGN KEY(dniJefe)  
REFERENCES Empleado(codEmpl);
```

Supóngase ahora que un mismo empleado puede tener más de un jefe. Por tanto, la columna dniJefe no puede estar en la tabla Empleado, ya que se debe crear una nueva tabla que soporte la interrelación de multiplicidad N:M que se quiere obtener. Para ello, se debe crear una nueva tabla que se llama EmpleadoJefe y que relaciona un empleado con su jefe. Esta tabla sería EmpleadoJefe(codEmpl Empleado, codEmleefe Empleado).

Para llevar esto a cabo, se procede, a modo de ejemplo, a eliminar de la tabla Empleado la columna dniJefe y la restricción fk_empleado_jefe. Los comandos para esto serian:

```
ALTER TABLE Empleado DROP dniJefe;  
ALTER TABLE Empleado DROP CONSTRAINT fk_empleado_jefe;
```

Para crear la tabla EmpleadoJefe, se debe crear primero con sus columnas y, luego, se les añaden sus restricciones. De lo contrario, al definir la clave ajena que apunta a la misma tabla, si esta aún no se ha creado, el comando emitiría un error.

```
CREATE TABLE EmpleadoJefe(  
    CodEmpl NUMBER(4),  
    CodEmplJefe NUMBER(4),  
    CONSTRAINT pk_empljefe PRIMARY KEY(CodEmpl,CodEmplJefe));
```

Ahora, si se les puede añadir la restricción clave ajena:

```
ALTER TABLE EmplJefe ADD CONSTRAINT fk_empljefe FOREIGN KEY(CodEmpl)  
REFERENCES Emleefe(CodEmpl);  
ALTER TABLE EmplJefe ADD CONSTRAINT fk_empljefe_jefe FOREIGN KEY(CodEmplJefe)  
REFERENCES Emleefe(CodEmpl);
```

Esto es lo que ocurre con las relaciones reflexivas de multiplicidad N:M.

Si se quisiera eliminar una clave primaria y las claves ajenas asociadas a ella, se usaría el comando ALTER TABLE Tabla DROP PRIMARY KEY CASCADE; o también el nombre de la restricción de la forma ALTER TABLE Tabla DROP CONSTRAINT nombreRestriccion CASCADE;.

PRÁCTICA PROPUESTA

ALTERACION BASE DE DATOS ACADEMIA

Lógate con el usuario Academia y crea las dos tablas siguientes:

Matricula(PF-codAlumn->Alumno,PF-codAsignatura->Asignatura, P-curso, calificacion)

Imparte(PF-dni->Profesor, PF-codAsig->Asignatura, P-curso)

A continuación, modifica cada una de las tablas con el fin de relacionarlas y cumpliendo los siguientes criterios:

- Se quiere conocer el ciclo al que pertenece cada asignatura.

- Se quiere conocer el departamento al que pertenece cada profesor.

- Se quieren registrar las asignaturas en las que se matricula cada alumno en cada curso y la calificación que obtiene.

- Se quieren registrar las asignaturas que imparte cada profesor en cada curso.

Indica todos los comandos ALTER TABLE necesarios para modificar la base de datos.

Para cambiar el nombre de una columna o de una restricción, se usa la cláusula RENAME O RENAME COLUMN dentro del comando ALTER TABLE. Su sintaxis es la siguiente

ALTER TABLE tabla RENAME [COLUMN] nombre_actual TO nombre__nuevo;.

Por ejemplo, para cambiar el nombre de la columna cantidad de Compra, se escribiría ALTER TABLE Compra RENAME COLUMN Cantidad TO Cantg. Para cambiar de nombre la restricción fk_empljefe_jefe por fk_empleadojefe_conjefe, se escribiría ALTER TABLE EmplJefe RENAME fk_empljefe_jefe TO fk_empleadojefe_conjefe,.

Drop Table

La operación de eliminar una tabla no debe tomarse a la ligera. Esta operación se lleva a cabo cuando las tablas se han volcado en copias de seguridad o se está experimentando en la preproducción de una base de datos. Eliminar una tabla conlleva eliminar sus registros, pero no la estructura; eliminar sus restricciones debido a que se van a implementar otras, eliminar alguna columna o eliminarla completamente. Para eliminar columnas y restricciones, se usa el comando que ya se ha estudiado ALTER TABLE y su cláusula DROP. Para eliminar los registros de una tabla, se usa el comando DELETE [FROM]. Este comando permite eliminar ciertos registros de toda la tabla. Para ello, se usa su cláusula WHERE. Por ejemplo, para eliminar todos los registros de la tabla EmpleadoJefe cuyo jefe es el 13, se ejecutaria el comando DELETE FROM EmpleadoJefe WHERE CodEmplJefe=13;.

Más énfasis tiene el comando DROP TABLE, que permite eliminar toda la tabla, sus registros y su estructura completa. La sintaxis de este comando es

DROP TABLE tabla [CASCADE CONSTRAINTS | PURGE];

Con este comando, se eliminan todos sus registros y su estructura, es decir, se elimina por completo. Además, no hay vuelta atrás, asique es importante prestar mucha atención cuando se ejecuta.

Además, se debe tener en cuenta si los registros que contiene están siendo referenciados en otras tablas. Si es así, la tabla no puede ser eliminada, a menos que añadamos la cláusula CASCADE CONSTRAINTS, en cuyo caso, se eliminará la tabla completa y las referencias que existan a ella desde otras tablas (claves ajenas). Por ejemplo, para eliminar la tabla EmpleadoJefe con todos sus registros y todas las referencias a ellas, se ejecutaria el comando:

DROP TABLE EmpleadoJefe CASCADE CONSTRAINTS;

Cuando se elimina una tabla, el espacio que ocupa no se libera inmediatamente, ya que este es un proceso que puede tardar, dependiendo de la arquitectura del servidor. Para exigir que el espacio se libere lo antes posible, se usa la opción PURGE. Así, con el comando:

DROP TABLE LineaVenta CASCADE CONSTRAINTS PURGE;

La tabla y sus relaciones se eliminan completamente y el espacio que ocupa es liberado ara uso posterior.

Con el comando **TRUNCATE**, se eliminan los registros de la tabla, pero no la estructura en si, por lo que la información almacenada en el diccionario de datos queda intacta. La diferencia respecto a usar el comando DELETE FROM Tabla es que este no libera el espacio que ocupa dichos registros, sin embargo, TRUNCATE si. También es importante tener cuidado con su uso, ya que no se puede deshacer usando el comando ROLLBACK. El comando **TRUNCATE EmpleadoJefe** elimina todos los registros de la tabla EmpleadoJefe liberando el espacio que ocupa.

Activar y desactivar constraints

Para gestiones asociadas a pruebas y rediseño de tablas, a veces, es interesante desactivar alguna restricción, por ejemplo, los campos NOT NULL a la hora de insertar registros que aún faltan valores por conocer. Los casos pueden ser muchos, pero el resultado es simple: desactivar una restricción para testear la base de datos. Para realizar esta acción, se usa la opción **ENABLE | DISABLE CONSTRAINT** del comando ALTER TABLE. Solo es necesario conocer el nombre de la restricción o el número que Oracle le haya asignando. El siguiente ejemplo desactiva la restricción de chequeo que tiene asignada la tabla Producto llamada ck_producto:

```
ALTER TABLE DISABLE CONSTRAINT Ck_producto;
```

Si se usa la opción CASCADE, también se desactivarán aquellas restricciones que dependen de ella.

```
ALTER TABLE DISABLE CONSTRAINT pk_producto CASCADE;
```

Para volver a activar una restricción desactivada, se usa la opción ENABLE CONSTRAINT. El siguiente ejemplo vuelve a activar la restricción de clave primaria de la tabla Producto:

```
ALTER TABLE ENABLE CONSTRAINT pk_producto CASCADE;
```

PRÁCTICA PROPUESTA

CAMBIANDO EL ESQUEMA DE UNA BASE DE DATOS

Dado el esquema de la parte superior, indica la secuencia de comandos para obtener el esquema inferior.

```
DatosPersonales(P·dni, Nombre, PrApellido, N· SgApellido, Domicilio, N·Telefono, N·Correo, C·tipo)
Director(PF·dni → DatosPersonales)
Comercial(PF·dni → DatosPersonales, comision)
Oficina(P·CodOficina, Direccion, Telefono, F·dniD → Director, F·dniC → Comercial)
Vendedor(PF·dni → DatosPersonales, C·Turno, F·CodOficina → Oficina)
Cliente(PF·dni → DatosPersonales, numTarjeta)
Venta(P·CodVenta, C·Tipo, F·dniV → Vendedor, F·dniCl → Cliente)
Editorial(P·CIF, RazonSocial, Direccion, SitioWeb, Correo, Telefono)
Libro(P·ISBN, Descripcion, N· NumPag, Precio, Autor, C·Tematica, F·CIF → Editorial)
LineaVenta(PF·ISBN → Libro, PF·fecha, PF·CodVenta → Venta, Cantidad)

TipoDP(P·CodTipo, Descripcion)
DatosPersonales(P·dni, Nombre, PrApellido, N· SgApellido, Domicilio, Telefono, Correo, F·tipo → TipoDP)
Director(PF·dni → DatosPersonales, anios)
Comercial(PF·dni → DatosPersonales)
Oficina(P·CodOficina, Direccion, Telefono, F·dniD → Director, F·dniC → Comercial, comision)
Turno(P·CodTurno, Descripcion)
Vendedor(PF·dni → DatosPersonales, F·turno → Turno, F·CodOficina → Oficina)
Cliente(PF·dni → DatosPersonales, numTarjeta)
Venta(P·CodVenta, C·Tipo, F·dniV → Vendedor, F·dniCl → Cliente)
Editorial(P·CIF, RazonSocial, Direccion, SitioWeb, Correo, Telefono, Direccion)
TematicaLibro(P·CodTem, NombreTematica)
Libro(P·ISBN, Descripcion, N· NumPag, PrecioActual, Autor, precio, F·CodTem → TematicaLibro, F·CIF → Editorial)
LineaVenta(PF·ISBN → Libro, PF·fecha, PF·CodVenta → Venta, Cantidad)
```


Lenguaje de Manipulación de datos

El lenguaje de manipulación de datos está compuesto por aquellos comandos que permiten la gestión de los datos que están en las tablas; a diferencia del lenguaje de definición de datos, que permite crear las tablas donde se almacenan los registros. De este modo, el lenguaje de manipulación de datos, o más comúnmente llamada lenguaje DML, está provisto de mecanismos para insertar datos en las tablas, modificarlos y eliminarlos. Estos comandos son **INSERT, UPDATE y DELETE**.

Insert

Cuando una tabla ya está creada, la mayoría de las operaciones que se hacen en ella consiste en insertar datos. La inserción de datos no está solo relacionada con el proceso que un operador puede llevar a cabo a través del comando INSERT, más bien, las aplicaciones informáticas, a través de mecanismos de conexión a servidores de bases de datos, insertan la información que los clientes elaboran. Las inserciones que se hacen en una tabla tienen que cumplir todas sus restricciones. Si no es así, la inserción emitirá un error en la máquina del servidor que el cliente podrá recoger y tomar decisiones para su corrección. Los mecanismos de control en el lado del servidor deben ser lo más completos posible, así, se evita que aplicaciones informáticas mal desarrolladas inserten datos incoherentes o, incluso, que tengan mal intención en los accesos.

La inserción de datos debe controlar el valor del campo clave, ya que este no se puede repetir a lo largo de los registros de una tabla. Para hacer esto óptimo, se puede usar lo que se llama campos autonuméricos. La elección del campo clave de una tabla no es inmediato. Existe la posibilidad de utilizar una cadena como campo clave. Esta cadena, cuando se propaga, ya contiene el texto que se quiere imprimir en papel o en pantalla, lo que minimiza las tareas en las aplicaciones para mostrar la información, ya que no es necesario buscar el texto correspondiente a un número. Otras veces, el campo clave es un número, pero no se trata de una secuencia, por el ejemplo, el DNI, el número de la Seguridad Social.

Generalmente, como el campo clave es el código que se usa para relacionar los registros, se trata más de un valor interno que de un valor propio de los registros. Por ello, además de memorizar estos datos, como el DNI, número de la Seguridad Social, código de identificación fiscal, etc., a cada uno de los registros de una tabla se le asigna un número secuencial atendiendo a la línea que ocupa en la tabla. Así, el valor que se propaga para las claves ajenas es este número, que, generalmente, tiene menos dígitos que los otros.

Estos campos autonuméricos permiten asignar esa secuencia automáticamente en cada inserción, sin necesidad de llevar ninguna cuenta del número de registros que hay en una tabla. Por lo tanto, en la mayoría de las tablas procedentes de tipos de entidad, se asigna un campo autonumérico para la identificación, exceptuando aquellos que derivan de los procesos de codificación de un campo, en los que se puede usarla propia cadena. De cualquier forma, depende de la base de datos y de su diseño físico, que hacen que se tomen diferentes decisiones para situaciones diferentes.

Otro factor importante que hay que tener en cuenta en las operaciones de inserción y que técnicos y programadores de las aplicaciones que van a acceder a la base de datos deben conocer es el orden de las columnas. Desde el punto de vista lógico, se trata de la misma tabla, pero, desde el punto de vista físico, son diferentes. Las dos tablas que se muestran abajo son análogas desde su vista lógica, pero son diferentes en su estructura física, ya que el orden de las columnas no es el mismo.

PRODUCTO						PRODUCTO					
P codProducto	NombreComercial	nombreInt	€ Tipo	prCoste	prVenta	P codProducto	prCoste	prVenta	€ Tipo	nombreInt	NombreComercial
1	Tarjeta Gráfica EX Th5+	TGEX5+	C	39,95	99	1	39,95	99	C	TGEX5+	Tarjeta Gráfica EX Th5+
2	MacBook 12N 32G 2TSSD	A12322	P	3195	3995	2	3195	3995	P	A12322	MacBook 12N 32G 2TSSD
3	HP Printer Inyección XJ01	phpxj	F	234	395	3	234	395	F	phpxj	HP Printer Inyección XJ01
4	Sobremesa JumpS2 32-16 2t	sj32162T	S	369	565	4	369	565	S	sj32162T	Sobremesa JumpS2 32-16 2t
5	Tarjeta red externa	NICExt	D	29	45	5	29	45	D	NICExt	Tarjeta red externa

Cada vez que se inserta un registro, es importante tener claro cuál es su campo clave y definirlo para que sea único para toda la vida de la base de datos. Cuando se inserta un registro en una tabla, se deben cumplir todos los requisitos declarados en ella, los campos obligatorios no admitirán nulos, los campos únicos así serán, los registros con claves ajenas deben existir en la tabla a la que se hace referencia, se deben cumplir las restricciones check, cumplir las condiciones asociadas a los TRIGGERS, etcétera.

Generalmente, son las aplicaciones las que hacen las inserciones en una base de datos, conectándose a ella, usando una cuenta de usuario y con suficientes privilegios para poder operar en ella. Esos comandos se embeben con el código del programa. Las inserciones, generalmente, se hacen desde aplicaciones que los programadores construyen y que requieren accesos a bases de datos. Estos comandos embebidos intercalan código en un lenguaje de programación y comandos SQL. En programación, por ejemplo, Java, se usan controladores JDBC u ODBC, que permiten la conexión y flujo de información desde y hacia la base de datos. Una vez que el usuario valida la información que introduce o importa, son las aplicaciones informáticas las que en su código embeben estas operaciones de inserción de datos. El SGBD controlará si la inserción es posible atendiendo a las restricciones inherentes y definidas en ellas.

INSERT INTO tabla[(columnas)] VALUES (valor1 [, valor2,...,valorn]);

La parte de *columnas* es opcional y permite indicar cuáles son las columnas las que hacen referencia los valores indicados tras la palabra VALUES. Si esta opción no se usa, será necesario indicar un valor para cada columna.

Supóngase que se pretende crear la tabla Producto y el orden de sus columnas son codProducto, NombreComercial, nombreInt, Tipo, prCoste y prVenta.

PRODUCTO					
P codProducto	NombreComercial	nombreInt	€ Tipo	prCoste	prVenta
1	Tarjeta Gráfica EX Th5+	TGEX5+	C	39,95	99
2	MacBook 12N 32G 2TSSD	A12322	P	3195	3995
3	HP Printer Inyección XJ01	phpxj	F	234	395
4	Sobremesa JumpS2 32-16 2t	sj32162T	S	369	565
5	Tarjeta red externa	NICExt	D	29	45

Para insertar el primer registro, se usaría el comando:

```
INSERT INTO Producto VALUES(1,'Tarjeta Gráfica EX Th5+', 'TGEX5+', 'C', 59.95, 99);
```

Se debe advertir que los datos que se insertan son acordes al orden en el que las columnas se crearon. Cuando se requiere insertar un registro en un orden diferente al que se ha creado, ya sea por requerimientos de la aplicación O porque no se conoce el orden real, se puede indicar cada una de las columnas en las que se inserta valor. Estas columnas se indican después del nombre de la tabla y es obligatorio que indiquen todas las columnas que sean NOT NULL. El siguiente comando es equivalente al anterior, pero los datos se insertan en otro orden:

```
INSERT INTO  
PRODUCTO(CodProducto,NombreInt,Tipo,prVenta,prCoste,NombreComercial)  
VALUES(1,'TGEX5+', 'C', 99, 59.95, 'Tarjeta Gráfica EX Th5+');
```

Incluso se puede indicar, aunque el orden sea el mismo. Esto se usa cuando hay muchos campos nulos y se quieren omitir.

```
INSERT INTO Producto VALUES(1,'Tarjeta Gráfica EX Th5+', 'TGEX5+', 'C', 59.95, 99);  
INSERT INTO Producto VALUES(2,'MacBook 12N 32G 2TSSD', 'A12322', 'P', 3195, 3995);  
INSERT INTO Producto VALUES(3,'HP Printer Inyección XJ01', 'phpxj', 'F', 234, 395);  
INSERT INTO Producto VALUES(4,'Sobremesa JumpS2 32-16 2t', 'sj32162T', 'S', 369, 565);  
INSERT INTO Producto VALUES(5,'Tarjeta red externa', 'NICExt', 'D', 29, 45);
```

Campos autonuméricos

En la mayoría de los casos, las tablas que proceden de tipos de entidad del diagrama conceptual pueden identificarse a través de un número secuencial a media que se hacen las inserciones. Las claves primarias no suelen estar sujetas a órdenes aleatorios de valores, a menos que sean datos como el DNI, número de la Seguridad Social, ISBN, Código de barra, etc. Es muy habitual que los campos claves estén sujetos a una secuencia autonumérica. Si se observa la tabla de la Figura 4.20, los valores de CodAsig cumplen una secuencia que comienza en 1 y se incrementa en cada inserción. Los SGBD disponen de mecanismos para automatizar esta cuenta y se usan en las inserciones de datos en las tablas. En Oracle, se hace necesario crear una secuencia. Para ello, se usa el comando CREATE SEQUENCE con la siguiente sintaxis:

```
CREATE SEQUENCE nombreSecuencia INCREMENT BY valor START WITH valor |  
MAXVALUE valor | MINVALUE valor | CYCLE | NOCYCLE | CACHE | NOCACHE | ORDER |  
NOORDER;
```

Las cláusulas más importantes son INCREMENT BY, para indicar el incremento de los valores; START WITH, el valor de inicio de la secuencia; MINVALUE, el valor mínimo cuando se comienza un nuevo ciclo, y MAXVALUE, para el valor máximo de la secuencia. La opción CYCLE permite que la secuencia comience de nuevo en caso de que llegue a su valor máximo. En caso contrario, se usará NOCYCLE.

El nombre de la secuencia debe ser único para cada tabla que se quiere automatizar la columna autonumérica. Por tanto, se crearán tantas secuencias como columnas se quieran automatizar. Para la tabla asignatura, se podría crear la secuencia siguiente:

```
CREATE SEQUENCE seqProducto START WITH 1 INCREMENT BY 1 MAXVALUE 99999 NOCYCLE;
```

Una vez que se ha creado una secuencia, se puede usar en los comandos INSERT. Para ello, se usará el método NEXTVAL que devuelve el valor actual de la secuencia y lo incrementa según el valor del incremento especificado con INCREMENT BY. Las inserciones podrían ser del siguiente modo:

```
INSERT INTO Producto VALUES(seqProducto.NEXTVAL,'Tarjeta Gráfica EX Th5+', 'TGEX5+', 'C', 59.95, 99);
INSERT INTO Producto VALUES(seqProducto.NEXTVAL,'MacBook 12N 32G 2TSSD', 'A12322', 'P', 3195, 3995);
INSERT INTO Producto VALUES(seqProducto.NEXTVAL,'HP Printer Inyección XJ01', 'phpxj', 'F', 234, 395);
INSERT INTO Producto VALUES(seqProducto.NEXTVAL,'Sobremesa JumpS2 32-16 2t', 'sj32162T', 'S', 369, 565);
INSERT INTO Producto VALUES(seqProducto.NEXTVAL,'Tarjeta red externa', 'NICExt', 'D', 29, 45);
```

Para comprobar el valor actual de la secuencia, se puede ejecutar la siguiente consulta:

```
SELECT seqProducto.CURRVAL FROM dual;
```

Con respecto a otros sistemas gestores de bases de datos, existen diferencias con respecto a las columnas autonuméricas. En SQL Server, se usa IDENTITY y, en MySQL, se usa AUTO_INCREMENT en la definición de la columna dentro del comando CREATE TABLE. El siguiente comando crea la tabla Proveedor usando MySQL:

```
CREATE TABLE Proveedor(
    codProveedor int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nombreProv VARCHAR(30),
    gerente VARCHAR(39)
);
```

GENERATED AS IDENTITY

En Oracle, se puede usar

GENERATED AS IDENTITY

En estos usos, en la operación INSERT, se deben indicar todas las columnas excepto la columna clave primaria.

Por ejemplo el comando Oracle equivalente al comando MySQL para crear la tabla proveedor podría ser:

```
CREATE TABLE Proveedor(
    codProveedor NUMBER(2) GENERATED ALWAYS AS IDENTITY,
    nombReProv VARCHAR(39),
    gerente VARCHAR(30)
);
```

Para una inserción en esta tabla, se deben indicar todas las columnas, como se observa en el comando siguiente:

```
INSERT INTO PROVEEDOR(nombreProv,gerente) VALUES('Components S/L', 'David Blanco');
```

Como se puede observar, se deben indicar todas las columnas de la tabla, excepto el campo clave. Al ser el primer registro de la tabla, Oracle inserta el código 1 a dicho registro. Veamos la sintaxis

GENERATED [ALWAYS | BY DEFAULT | [ON NULL]] AS IDENTITY [(OPCIONES)]

- **GENERATED ALWAYS:** Oracle siempre genera un valor para la columna, si intentamos insertar un valor en esa columna dará un error.
- **GENERATED BY DEFAULT:** Oracle genera un valor para la columna si nosotros no le damos valor. Si le damos un valor, Oracle lo insertará. En esta opción Oracle nos dará un error si insertamos un null.
- **GENERATED BY DEFAULT ON NULL:** Oracle genera un valor si intentamos insertar un null o no insertamos valor. Si le damos un valor lo insertará.

Opciones:

- **START WITH valor inicial** . Por defecto empieza en 1
- **INCREMENT BY valor.** Define el valor de incremento. Por defecto es 1
- **CACHE** define un número de valores que Oracle debería generar de antemano para mejorar el rendimiento. Se usa en caso en que la tabla tenga muchos inserts.

1. Ejemplo de generated always:

```
CREATE TABLE identity_demo (  
    id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
    description VARCHAR2(100) NOT NULL  
);
```

```
INSERT INTO identity_demo(description) VALUES('Oracle identity column demo');
```

2. Ejemplo de generated by default

```
CREATE TABLE identity_demo (  
    id NUMBER GENERATED BY DEFAULT AS IDENTITY,  
    description VARCHAR2(100) not null  
);
```

```
INSERT INTO identity_demo(description) VALUES('Oracle identity column demo');  
INSERT INTO identity_demo(id,description) VALUES(2, 'Oracle identity');
```

Pero nunca podré intentar insertar un null poniendo el campo y el valor NULL

3. Ejemplo de generated by default on null

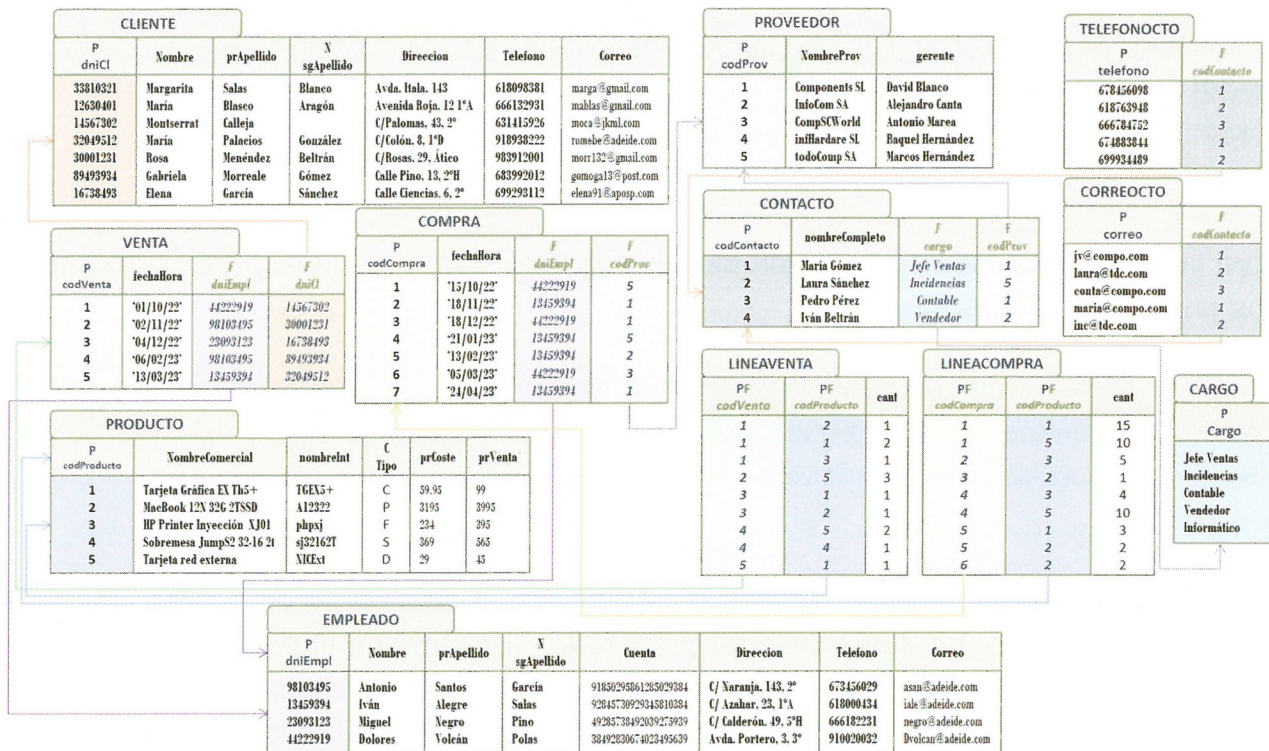
```
CREATE TABLE identity_demo (  
    id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,  
    description VARCHAR2(100) not null  
);
```

```
INSERT INTO identity_demo(description) VALUES('Oracle identity column demo');  
INSERT INTO identity_demo(id,description) VALUES(null, 'Oracle identity demo');
```

PRÁCTICA RESUELTA

INSERCIÓN EN LA "TIENDA DE INFORMÁTICA"

Para la base de datos de la tienda de informática cuyas tablas se ven abajo. Modifica el script de creación de la base de datos para crear las secuencias adecuadas y hacer los inserts de las tablas.



Update

Cuando se quieren actualizar los datos, se hace necesario analizar si esos cambios son buenos para la evolución de la base de datos, si deben estar permitidos y qué usuarios pueden modificarlos. Por ejemplo, cambiar el número de ejemplares que hay en una librería sobre sus productos puede ser un hecho fatal para la organización. Cuando los datos se transforman atendiendo a políticas establecidas, los valores antiguos pueden ser almacenados en otros espacios con el fin de disponer de la evolución de estos. Este proceso se conoce como historial de los datos. Por ejemplo, interesaría en gran medida almacenar el historial de los precios de coste y de ventas de los productos de una empresa. Además, imagínese, si un libro se ha vendido a 29 € el año pasado y ahora cuesta 25 €, las ventas que se hayan hecho no pueden estar relacionadas con el nuevo precio, sino que deben seguir con su precio de venta de 29 €.

Para modificar los valores almacenados en los registros de las tablas, se usa **UPDATE**

UPDATE tabla SET columna = valor_nuevo [,columna =valor_nuevo] [WHERE condicion];

Después de la cláusula SET, se pueden indicar más de una columna. La cláusula WHERE filtra aquellos registros a modificar. En caso de no usarla, el valor se actualiza para todos los registros de la tabla, por lo que se debe prestar atención en este tipo de operaciones.

Supóngase que se quieren hacer los siguientes cambios en la tabla Cliente de la base de datos del caso «Tienda de informática»: insertar el segundo apellido, Perez, del cliente con nombre Montserrat, Gabriela vive en el 2.º D, no en el 2.º H, y el teléfono de Elena termina en 3. Obviamente, nunca se insertan datos cuando ya existe una fila, sino que, en caso de sustituir algún valor NULL, lo que se debe hacer es modificar dicha columna y fila con UPDATE. Así, para sustituir el valor NULL de Montserrat, se usará UPDATE. Otra situación que resolver es como identificar el registro o registros que modificar. Lo óptimo es usar el campo clave de la fila que actualizar, ya que, por ejemplo, podría existir más de un cliente llamado Montserrat. En realidad, para concretar el registro que se quiere modificar, no basta con decir que se quiere modificar el apellido de Montserrat, sino, más bien, que se quiere actualizar el segundo apellido del cliente con DNI 14567302. El comando sería:

```
UPDATE Cliente SET sgApellido='Pérez' WHERE dni=14567392;
```

Para modificar la dirección de Gabriela, también se debe indicar su identificador, es decir, el cliente con DNI 89493934. También se debe tener en cuenta que la dirección está en uno campo y, por tanto, para sustituir la H por la D de la dirección de este cliente, se debe escribir toda la dirección completa nueva.

```
UPDATE Cliente SET direccion='Calle Pino,13, 2ºD' WHERE dni=89493934;
```

Para la tercera actualización ocurre lo mismo, se debe indicar el DNI e indicar el teléfono.

```
UPDATE Cliente SET telefono=699293113 WHERE dni=894939343;
```

Si se hubiera escrito el comando filtrando por el nombre Elena, se debe ser consciente de que se actualizarán todos los teléfonos de clientes cuyo nombre es Elena. Todos los clientes llamados Elena tendrían el mismo teléfono.


```
UPDATE Cliente SET telefono=699293113 WHERE Nombre='Elena';
```

También se debe tener en cuenta cómo se escribe un literal.

Al filtrar por 'Elena', si alguien se llamase ELENA, este registro no se seleccionaría. Una solución puede ser convertir el literal en mayúsculas y compararlo con ELENA, así, independientemente de cómo esté escrito, se filtraría todos los registros con nombre Elena. Para obtener el literal de una columna escrito en mayúsculas, se usa la función UPPER. El problema que tiene la función UPPER es que no funciona con los caracteres que tienen tildes ya que estos son caracteres especiales y están ubicados en lugares diferentes en la tabla ASCII. Esto se estudiará más adelante en profundidad.

```
UPDATE Cliente SET telefono=699293113 WHERE UPPER(Nombre)='ELENA';
```

También se puede actualizar toda una columna y usar operaciones aritméticas. Por ejemplo, supóngase que se quieren rebajar todos los precios de venta un diez por ciento.

El comando para ello podría ser:

```
UPDATE Producto SET prVenta=prVenta-prVenta/10;
```

Recuérdese que se aconseja no cambiar los precios, pues no se deben perder los históricos de estos, sobre todo, cuando están relacionados en otras ventas o compras.

Delete

Eliminar los datos de una tabla siempre conlleva procesos de reflexión sobre su extracción y eliminación definitiva. En la mayoría de los casos, estos datos son exportados a tablas historiales. Además, casi seguro que los registros que se quieren eliminar están relacionados con otros y supone pérdida de información del pasado. Por ejemplo, al intentar eliminar un registro de la tabla producto que se sabe que no se está usando en la actualidad. Si se eliminase, ¿qué pasaría, por ejemplo, con la contabilidad de ejercicios económicos de dichos años? ¿Pueden ser eliminados? ¿Se cumple con normativas sobre su almacenamiento fuera de la base de datos en relación con la protección de datos, uso de los soportes y datos históricos? Estas preguntas, entre otras muchas, influyen en gran medida en el proceso de eliminación de registros.

El comando del que se dispone para eliminar registros es DELETE y su sintaxis es:

```
DELETE [FROM] tabla [WHERE condicion];
```

Supóngase que se quiere eliminar de la tabla Cargo el registro Informático.

```
DELETE Cargo WHERE codCargo='Informático';
```

Supóngase que se quieren eliminar los teléfonos del contacto con código 1.

```
DELETE TelefonoCto WHERE codContacto=1;
```

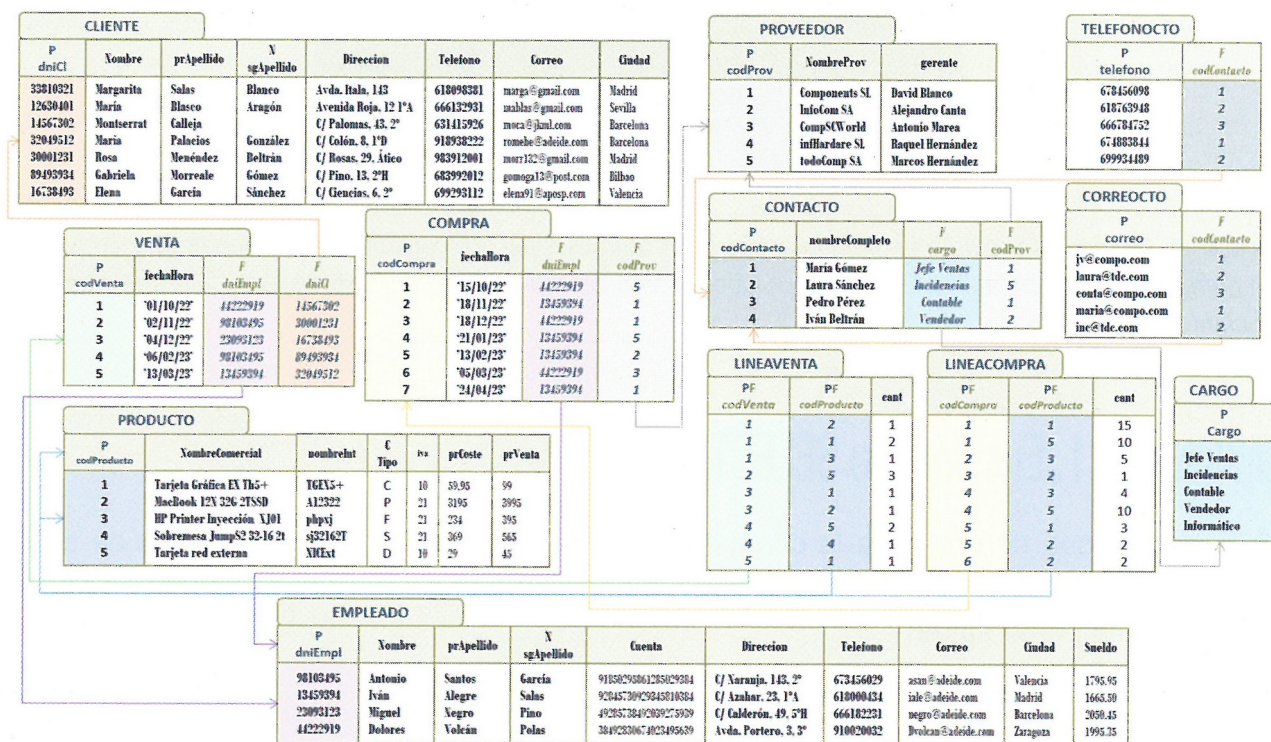
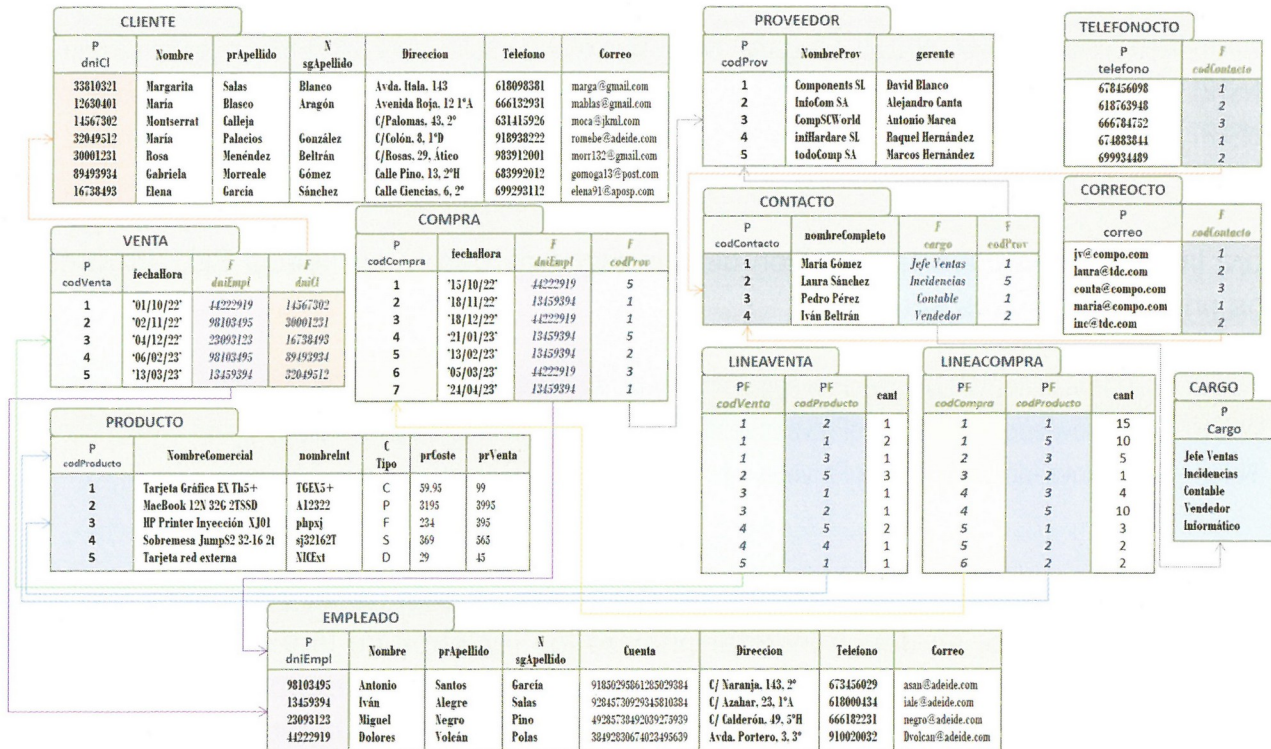
Si no se usa la cláusula WHERE en el comando DELETE, se eliminarán todos los registros de la tabla. Así, el comando siguiente dejaría vacía la tabla Producto:

```
DELETE Producto;
```

PRÁCTICA PROPUESTA

DML EN LA "TIENDA DE INFORMÁTICA"

Para la base de datos de la tienda de informática cuyas tablas se ven abajo INDICA TODOS LOS COMANDOS ALTER, insert, update y delete necesarios para convertirla en la que se ve más abajo



PRÁCTICA PROPUESTA

DISPOSITIVOS MÓVILES

Dados los siguientes requisitos de información, obtén el script DDL para la creación de sus tablas.

R1. Se desea almacenar información sobre las compras y ventas de dispositivos móviles en un determinado país. Se registrará información sobre los fabricantes, almacenando su código, su nombre, su dirección web, un teléfono de contacto y la ciudad en la que tiene sede.

R2. Los fabricantes publican la marca y el modelo de cada móvil que saca al mercado. Se hace necesario almacenar dicha información, así como el precio de venta recomendado.

R3. Los clientes, de los que se hace necesario almacenar sus datos personales, así como la ciudad en la que viven y un teléfono de contacto, compran productos en diferentes tiendas, ya sea en la misma ciudad en la que viven o en otra. Interesa registrar cada uno de los productos que los clientes compran y a qué tienda.

R4. Consideramos que, en cada venta, se almacena qué producto se vende, qué cantidad, el precio final del producto y el color del dispositivo (una venta por cada dispositivo).

R5. Sobre los diferentes puntos de venta, se almacenarán sus datos más representativos, como su CIF, nombre comercial, su dirección web, un teléfono de contacto y la ciudad en la que tiene su domicilio fiscal.

PRÁCTICA PROPUESTA

DISPOSITIVOS MÓVILES II

Plantea los comandos DML atendiendo a estos cambios:

Eliminar el precio final de los móviles, puesto que se cobra siempre el precio recomendado.

Codificar el campo Marca usando la tabla Marca(P-**CodMarca**, U-Marca).

Codificar el campo color con la tabla Color(P-**CodColor**, U-Color).

Codificar el campo Ciudad con la tabla Ciudad(P-**CodCiudad**, U-Ciudad).

Se pueden comprar más de un móvil diferente en una misma venta, creando las tablas Venta y LineaVenta.

Las tablas que se deben obtener son Fabricante, Marca, Movil, Ciudad, Color, Cliente, LineaVenta, Venta y Tienda.

Crea las secuencias necesarias y realiza varias inserciones utilizando las secuencias anteriores.

Borra las secuencias y crea columnas IDENTITY inserta varias filas.

Solucionario

Plataforma de videojuegos online

Primero, se crea el usuario JuegosOnline. Para ello, se puede ejecutar en sqlplus estos comandos:

```
ALTER SESSION SET CONTAINER=XEPDB1;  
CREATE USER JUEGOSONLINE IDENTIFIED BY JUEGOSONLINE DEFAULT TABLESPACE  
USERS TEMPORARY TABLESPACE TEMP;  
GRANT CONNECT, RESOURCE TO JUEGOSONLINE;
```

Una vez creado el usuario, se usa para autenticarse y crear las tablas SitioWeb, Socio, Juego, Tematica, TemaJuego, Autor, Autoría y Sesión.

Para la tabla SitioWeb con descripción Sitioweb(P-codSitioWeb, DireccionURL), el comando que la crea es el siguiente:

```
CREATE TABLE SitioWeb(  
    codSitioWeb NUMBER(2),  
    DireccionURL VARCHAR2(40) UNIQUE,  
    CONSTRAINT pk_Sitioweb PRIMARY KEY(CodSitioWeb)  
);
```

Para la tabla Socio con descripción SociolP-dniSocio,Nombre, prApellido, N-sgApellido), el comando es el siguiente:

```
CREATE TABLE Socio(  
    dniSocio NUMBER(8),  
    Nombre VARCHAR2(29) NOT NULL,  
    PrApellido VARCHAR2(29) NOT NULL,  
    SgApellido VARCHAR2(29) NULL,  
    CONSTRAINT pk_Socio PRIMARY KEY(dniSocio)  
);
```

Para la tabla Juego con descripción Juego(P-codJuego, Titulo, N-NumNiveles), el comando es el siguiente:

```
CREATE TABLE Juego(  
    codJuego NUMBER(4),  
    Título VARCHAR2(30) NOT NULL,  
    NumNiveles NUMBER(3) NULL,  
    CONSTRAINT pk_Juego PRIMARY KEY(codJuego)  
);
```

Para las tablas Tematica con la descripción Tematica(P-Tematica) y TemaJuego con la descripción TemaJuego(F-P-codJuego->Juego, P-F-Tematica->Tematica) tenemos:

```
CREATE TABLE Tematica(  
    Tematica VARCHAR2(20),  
    CONSTRAINT pk_tematica PRIMARY KEY(Tematica)  
);
```



```

CREATE TABLE TemaJuego(
    codJuego NUMBER(4),
    Tematica VARCHAR2(20),
    CONSTRAINT pk_TemaJuego PRIMARY KEY(codJuego,Tematica),
    CONSTRAINT fk_TemaJuego_Juego FOREIGN KEY(codJuego) REFERENCES
    Juego(codJuego),
    CONSTRAINT fk_TemaJuego_Tematica FOREIGN KEY(Tematica) REFERENCES
    Tematica(Tematica)
);

```

Para la tabla Autor(P-dni, Nombre, prApellido, N-sgApellido, Empresa) y Autoria(P-F-codJuego> Juego, P-F-dni> Autor), los comandos son:

```

CREATE TABLE Autor(
    dni NUMBER(8),
    Nombre VARCHAR2(29) NOT NULL,
    PrApellido VARCHAR2(29) NOT NULL
    SgApellido VARCHAR2(29) NULL,
    Empresa VARCHAR2(20) NOT NULL,
    CONSTRAINT pk_Autor PRIMARY KEY(dni)
);

```

```

CREATE TABLE Autoria(
    codJuego NUMBER(4),
    dni NUMBER(8),
    CONSTRAINT pk_Autoria PRIMARY KEY(codJuego,dni),
    CONSTRAINT fk_Autoria_Juego FOREIGN KEY(codJuego) REFERENCES
    Juego(codJuego),
    CONSTRAINT fk_Autoria_Autor FOREIGN KEY(dni) REFERENCES Autor(dni)
);

```

Para la tabla Sesion(P-codSesion, Fecha, F-codSitioWeb->SitioWeb, F-dniSocio->Socio, F-codJuego->Juego) el comando sería:

```

CREATE TABLE Sesion(
    CodSesion NUMBER(7),
    Fecha DATE,
    codSitioWeb NUMBER(2) NOT NULL,
    dniSocio NUMBER(8) NOT NULL,
    codJuego NUMBER(4) NOT NULL,
    CONSTRAINT pk_sesion PRIMARY KEY(codSesion),
    CONSTRAINT fk_SesionSitioWeb FOREIGN KEY(CodSitioWeb) REFERENCES
    SitioWeb(codSitioWeb),
    CONSTRAINT fk_Sesion_Socio FOREIGN KEY(dniSocio) REFERENCES
    Socio(dniSocio),
    CONSTRAINT fk_Sesion_Juego FOREIGN KEY(codJuego) REFERENCES
    Juego(codJuego)
);

```


Tienda de informática

Primero, se crea el usuario Tienda informática.

```
ALTER SESSION SET CONTAINER=XEPDB1;  
CREATE USER TIENDAINFORMATICA IDENTIFIED BY tienda DEFAULT TABLESPACE USERS  
TEMPORARY TABLESPACE TEMP;  
GRANT CONNECT, RESOURCE, TO TIENDAINFORMATICA;
```

Tras su autenticación, se crean las tablas

Las restricciones que tiene la tabla

Empleado(P-**dniEmpl**, Nombre, prApellido, N-SgApellido, Cuenta, Domicilio, Telefono, Correo) son PRIMARY KEY para la columna dniEmpl y NULL para la columna sgApellido, es decir, todas son NOT NULL exceptuando sgApellido. El comando para crear esta tabla es el siguiente:

```
CREATE TABLE Empleado(  
    dniEmpl NUMBER(8),  
    Nombre VARCHAR2(29) NOT NULL,  
    PrApellido VARCHAR2(29) NOT NULL,  
    SgApellido VARCHAR2(29),  
    Cuenta NUMBER(20) NOT NULL,  
    Domicilio VARCHAR2(69) NOT NULL,  
    Telefono NUMBER(9) NOT NULL,  
    Correo VARCHAR2(128) NOT NULL,  
    CONSTRAINT pk_emp1eado PRIMARY KEY(dniEmpl));
```

Se podrian añadir, a modo de ejemplos, algunas restricciones CHECK. Supóngase que se quiere indicar que el nombre debe tener como minimo tres letras: Se puede usar la función LENGTH para conocer el número de caracteres que tiene una cadena. Asi, se podria cerrar que el campo cuenta tenga 20 digitos y el teléfono, 9. Con estos cambios, el comando quedaria como sigue:

```
CONSTRAINT ck_empleado CHECK(LENGTH(Nombre)>=3 AND LENGTH(prApellido)>=4 AND  
LENGTH(cuenta)=29 AND LENGTH(Telefono)=9)
```

Para la tabla Cliente(P-dniCI, Nombre, prApellido, N-sgApellido, Domicilio, Telefono, Correo), el comando resulta muy análogo al anterior:

```
CREATE TABLE Cliente  
    dniCI Number(8),  
    Nombre VARCHAR2(20) NOT NULL,  
    PrApellido VARCHAR2(20) NOT NULL  
    SgApellido VARCHAR2(20),  
    Domicilio VARCHAR2(60) NOT NULL,  
    Telefono NUMBER(9) NOT NULL,  
    Correo VARCHAR2(128) NOT NULL,  
    CONSTRAINT pk_cliente PRIMARY KEY(dniCI),  
    CONSTRAINT ck_cliente CHECK(LENGTH(Nombre)>=3 AND LENGTH(prApellido)>=4  
AND LENGTH(Telefono)=9));
```

Las restricciones de la tabla

Venta(P-**CodVenta**, fechaHora, F-dniEmpl -> Empleado, F-dniCl -> Cliente) son PRIMARY KEY para la columna CodVenta, FOREIGN KEY para dniEmpl y dniCl. En esta tabla, la columna dniEmpl almacena el DNI del empleado que realiza la venta y la columna dniCl, el cliente que hace la compra. Ambos campos son obligatorios. Las tablas a la que apuntan las claves ajenas deben estar creadas con antelación. El campo clave es de tipo NUMBER(7) porque se considera que se van a insertar 9 999 999 ventas como máximo.

Este valor depende de la previsión que hace la organización sobre sus ventas. Para las claves ajenas, se podrían añadir las restricciones de integridad referencial, por ejemplo, ON DELETE CASCADE. El comando para crear esta tabla es el siguiente:

```
CREATE TABLE Venta(  
    codVenta NUMBER(7),  
    fechaHora DATE NOT NULL,  
    dniEmpl NUMBER(8) NOT NULL,  
    dniCL NUMBER(8) NOT NULL,  
    CONSTRAINT pk_Venta PRIMARY KEY(CodVenta),  
    CONSTRAINT fk_venta_empl FOREIGN KEY(dniEmpl) REFERENCES Empleado(dniEmpl) ON  
DELETE CASCADE,  
    CONSTRAINT fk_venta_cliente FOREIGN KEY(dniCL) REFERENCES Cliente(dniCL) ON DELETE  
CASCADE);
```

Para la tabla

Producto(P-**CodProducto**, nombreComercial, nombreInterno, C-Tipo, precioCoste, precioVenta)

Se crean las restricciones PRIMARY KEY para CodProducto y CHECK para Tipo. En esta fase, es el momento de aclarar y definir los valores de un campo codificado. Consiste en decidir si los valores se pueden definir con una condición en el CHECK o, sin embargo, se hace necesario crear una tabla nueva e insertar en ella los posibles valores. Esto se hará cuando se prevé que los valores pueden cambiar o crecer o también cuando el número de valores es grande para definirlo en un check. Cuando se crea la tabla, existen dos posibilidades: usar el propio valor como identificador o añadir un campo clave asociado a una secuencia. Esto último se hará cuando los valores ocupen mucho espacio, como cadenas de larga longitud, y estos campos no tengan una tasa alta de consultas. Esto depende de la eficiencia espacial o temporal que se busque, ya que, al añadir un código numérico a los valores, se propaga un dato más pequeño, ahorrando espacio, pero se hace necesario cruzar las tablas, disminuyendo la eficiencia temporal. Estas decisiones de diseño se pueden volver a evaluar y transformar en caso de detección de deficiencia. Para este caso, los valores del campo Tipo son componente de un ordenador, sobremesa, portátil, dispositivo de comunicación y periférico. Se usará un check codificándolo con las letras C, S, P, D, F, por lo que el campo Tipo será de un carácter, es decir, CHAR(1). El CHECK resultante sería CHECK(Tipo IN ('C' , 'S' , 'P' , 'D' , ' F')). Para el campo clave CodProducto, se decide de tipo NUMBER(4), por lo que se considera que existirán como máximo 10000 productos codificados desde el 0 hasta el 9999. Para las columnas PrecioCoste y PrecioVenta, se consideran que pueden tomar valores que van desde 0,01 hasta 9999,99. Para ello, se definen del tipo NUMBER(6,2), es decir, un número de seis dígitos de los cuales dos son para la parte decimal.

Se pueden añadir algunas restricciones más. Por ejemplo, los precios no pueden ser negativos, el nombre comercial e interno deben tener como mínimo cinco caracteres y el precio de coste debe ser menor que el precio de venta. Considerando estas restricciones de chequeo, el comando para crear la tabla quedaría como sigue:

```
Create table Producto(  
    codProducto NUMBER(4),  
    nombreComercial VARCHAR2(30) NOT NULL,  
    nombreInterno VARCHAR2(30) NOT NULL,  
    Tipo CHAR(1) NOT NULL,  
    PrecioCoste NUMBER(6,2) NOT NULL,  
    PrecioVenta NUMBER(6,2) not null,  
    CONSTRAINT pk_producto PRIMARY KEY(codProducto),  
    CONSTRAINT ck_producto CHECK(Tipo IN ('C','S','P','D','F') AND  
    LENGTH(nombreComercial)>=5 AND LENGTH(nombreInterno)>=5 AND  
    precioCoste>=0 AND precioVenta>=0.01 AND precioCoste<precioVenta)  
);
```

Se debe advertir que cada condición del check se une con el operador AND, ya que se deben cumplir todas para que un registro sea admitido en dicha tabla. Cuando se escriben por separado, también se deben cumplir todas las restricciones de una tabla. El comando equivalente es el siguiente:

```
CONSTRAINT ck_prod_tipo CHECK(Tipo IN ('C','S','P','D','F')),  
CONSTRAINT ck_prod_nCom CHECK(LENGTH(nombreComercial)>=5),  
CONSTRAINT ck_prod_nInt CHECK(LENGTH(nombreInterno)>=5),  
CONSTRAINT ck_prod_precio CHECK(precioCoste>=0 AND precioVenta>=0.01 AND  
precioCoste<precioVenta)
```

La tabla que une los productos vendidos en cada venta es

LineaVenta(P-F-CodVenta -> Venta, P-F-CodProducto -> Producto, cantidad)

Las tablas a las que apuntan las claves ajenas deben estar creadas con antelación. CodVenta es clave primaria y clave ajena que apunta a Venta. Este campo tiene que ser del mismo tipo que el campo clave de venta, es decir, NUMBER(7). Igual ocurre con CodProducto que apunta a la tabla Producto, por lo que su tipo es NUMBER(4). El campo clave de esta tabla se encuentra compuesto por las columnas CodVenta y codProducto. El comando para crear esta tabla es:

```
CREATE TABLE LineaVenta(  
    codVenta NUMBER(7),  
    codProducto NUMBER(4),  
    cantidad NUMBER(2) NOT NULL,  
    CONSTRAINT pk_lineaVenta PRIMARY KEY(codVenta,codProducto),  
    CONSTRAINT fk_lineaVenta_Venta FOREIGN KEY(codVenta) REFERENCES  
venta(CodVenta),  
    CONSTRAINT fk_lineaVenta_Prod FOREIGN KEY(CodProducto) REFERENCES  
Producto(CodProducto));
```

Para la tabla Proveedor(P-CodProv,NombreProv, gerente),
se usa el comando siguiente:

```
CREATE TABLE Proveedor(  
    codProv NUMBER(8),  
    NombreProv VARCHAR2(20) NOT NULL,  
    gerente VARCHAR2(20) NOT NULL,  
    CONSTRAINT pk_Proveedor PRIMARY KEY(codProv)  
);
```

Para la gestión de los teléfonos y correos de los contactos, así como del cargo de estos, se tenían las tablas con descripciones siguientes:

Cargo(P-cargo)

Contacto(P-codContacto, nombreCompleto, F-cargo->cargo)

CorreoCto(P-Correo, F-codContacto -> Contacto)

TelefonoCto(P-Telefono, F-codContacto -> Contacto)

La secuencia de comandos CREATE TABLE para estas tablas es la siguiente:

```
CREATE TABLE Cargo(  
    cargo VARCHAR2(20),  
    CONSTRAINT pk_cargo PRIMARY KEY(cargo));  
  
CREATE TABLE Contacto(  
    codContacto NUMBER(4),  
    NombreCompleto VARCHAR2(30) NOT NULL,  
    Cargo VARCHAR2(20) NOT NULL,  
    CONSTRAINT pk_contacto PRIMARY KEY(codContacto),  
    CONSTRAINT Fk_contacto_cargo FOREIGN KEY(cargo) REFERENCES Cargo(cargo));  
  
CREATE TABLE CorreoCto(  
    Correo VARCHAR2(100),  
    CodContacto NUMBER(4) NOT NULL,  
    CONSTRAINT pk_Correocto PRIMARY KEY(Correo),  
    CONSTRAINT fk_correocto_contacto FOREIGN KEY(codContacto) REFERENCES  
Contacto(codContacto));  
  
CREATE TABLE TelefonoCto(  
    Telefono NUMBER(9),  
    CodContacto NUMBER(4) NOT NULL,  
    CONSTRAINT pk_teleFonocto PRIMARY KEY(telefono),  
    CONSTRAINT fk_telefonocto_contacto FOREIGN KEY(codContacto) REFERENCES  
Contacto(codContacto));
```

Por último, para la gestión de las compras, se usan dos tablas. Compra con la descripción

Compra(P-CodCompra, fechaHora, F-dniEmpl ->Empleado, FcodProv -> Proveedor)

para almacenar los datos generales de la compra, la fecha, en el DNI del empleado que la hace y el código del proveedor. Para registrar todos los productos que se compran, se

requiere una tabla que relacione una compra con los diferentes productos que intervienen en ella, cuya descripción es:

LineaCompra(PF-CodCompra -> Compra, PF-CodProducto->Producto, cantidad)

Los comandos para crear estas dos tablas son:

```
CREATE TABLE Compra(  
    codCompra NUMBER(7),  
    fechaHora DATE NOT NULL,  
    dniEmpl NUMBER(8) NOT NULL,  
    codProv NUMBER(3) NOT NULL,  
    CONSTRAINT pk_Compra PRIMARY KEY(codCompra),  
    CONSTRAINT fk_compra_empl FOREIGN KEY(dniEmpl) REFERENCES  
Empleado(dnfEmpl),  
    CONSTRAINT fk_compra_prov FOREIGN KEY(codProv) REFERENCES  
Proveedor(codProv));
```

```
CREATE TABLE LineaCompra(  
    codcompra NUMBER(7),  
    codProducto NUMBER(4),  
    Cantidad NUMBER(2) NOT NULL,  
    CONSTRAINT pk_lineaCompra PRIMARY KEY(codCompra,codProducto),  
    CONSTRAINT fk_lc_compr FOREIGN KEY(codCompra) REFERENCES  
Compra(codCompra),  
    CONSTRAINT fk_lc_pr FOREIGN KEY(codProducto) REFERENCES  
Producto(codProducto));
```


Inserción en la Tienda de informática

La empresa ya ha comprado ciertos artículos para ponerlos a la venta. Estos productos se observan a través de su código en la tabla LineaCompra y la cantidad de cada artículo comprado. Ya se han dado de alta a algunos clientes en la tabla Cliente, los cuales han realizado sendas compras. Esto se observa en la tabla LineaVenta. Los datos de los proveedores con los que ya se han establecido relaciones se encuentran en la tabla Proveedor, Contacto para registrar las personas con las que se trabaja, TelefonoCto y CorreoCto para registrar sus teléfonos y correos y la tabla Cargo resultante de codificar cada uno de los posibles cargos de los trabajadores de las empresas proveedoras.

En las tablas Venta y Compra, se registra la información principal, fecha, empleado que la hace y cliente para venta y proveedor para compra.

Las tablas a las que se les puede crear una secuencia para su campo identificador son Cliente, Proveedor, Venta, Compra, Contacto, Producto y Empleado. Para ello, se crea una secuencia para cada tabla. Tanto para la tabla Cliente como para la tabla Empleado, se ha considerado como identificador el DNI, por lo que no es posible usar una secuencia para ellas.

Las secuencias que crear serían:

```
CREATE SEQUENCE seqProveedor;  
CREATE SEQUENCE seqVenta;  
CREATE SEQUENCE seqCompra;  
CREATE SEQUENCE seqContacto;  
CREATE SEQUENCE seqProducto;
```

En la inserción de los registros, se usarán dichas secuencias usando NEXTVAL con el formato Secuencia.NEXTVAL. Para la tabla Cliente, las primeras inserciones serían:

```
INSERT INTO Cliente VALUES(33810321,'Margarita','Salas','Blanco','Avda. Italia, 143518098381,  
'marga@gmail.com');  
INSERT INTO Cliente VALUES(12630401,'María','Blasco','Aragon','Avenida Roja, 12 1ºA', 666132931,  
'mablas@gmail.com');  
INSERT INTO Cliente VALUES(4567302,'Ivontserrat','Calleja',NULL,'C/Palomas,43, 2º', 631415926,  
'moca@jkm.com');
```

Para la tabla Proveedor, la inserción de sus registros usa la secuencia SEQ_Proveedor.

Los primeros registros son:

```
INSERT INTO Proveedor VALUES(seqProveedor.NEXTVAL, 'Components SL','David Blanco');  
INSERT INTO Proveedor VALUES(seqProveedor.NEXTVAL, 'InfoCom SA','Alejandro Canta');  
INSERT INTO Proveedor VALUES(seqProveedor.NEXTVAL, 'CompSCWorld','Antonio Marea');
```

Para la tabla TelefonoCto, la inserción de sus registros debe tener en cuenta que exista el contacto. Por ello, se deben insertar primero los registros de la tabla Contacto. Para la inserción de los contactos, la tabla Cargo debe estar creada y existir cualquier registro que se vaya a relacionar. Así, las inserciones de los registros en la tabla Cargo se harían con los siguientes comandos:

```
INSERT INTO Cargo VALUES('Jefe Ventas');  
INSERT INTO Cargo VALUES('Incidencias');  
INSERT INTO Cargo VALUES('Contable');  
INSERT INTO Cargo VALUES('Vendedor');  
INSERT INTO Cargo VALUES('Informático');
```

Una vez que ya existen los registros en cargo, se pueden insertar los registros de contacto:

```
INSERT INTO Contacto VALUES(seqContacto.NEXTVAL, 'María Gómez','Jefe Ventas',1);
INSERT INTO Contacto VALUES(seqContacto.NEXTVAL, 'Laura Sánchez','Incidencias',5);
INSERT INTO Contacto VALUES(seqContacto.NEXTVAL, 'Pedro Pérez','Contable',1);
INSERT INTO Contacto VALUES(seqContacto.NEXTVAL, 'Iván Beltrán','Vendedor',2);
```

Ahora, ya se pueden insertar los registros en la tabla TelefonoCto:

```
INSERT INTO TelefonoCto VALUES(678456098,1);
INSERT INTO TelefonoCto VALUES(618763948,2);
```

Así como los registros en la tabla CorreoCto:

```
INSERT INTO CorreoCto VALUES('jv@compo.com',1);
INSERT INTO CorreoCto VALUES('laura@tde.com',2);
```

El proceso continuaría de la misma forma, prestando atención sobre los registros que deben estar en la tabla antes de aquellos que usen como clave ajena algún código de estos.