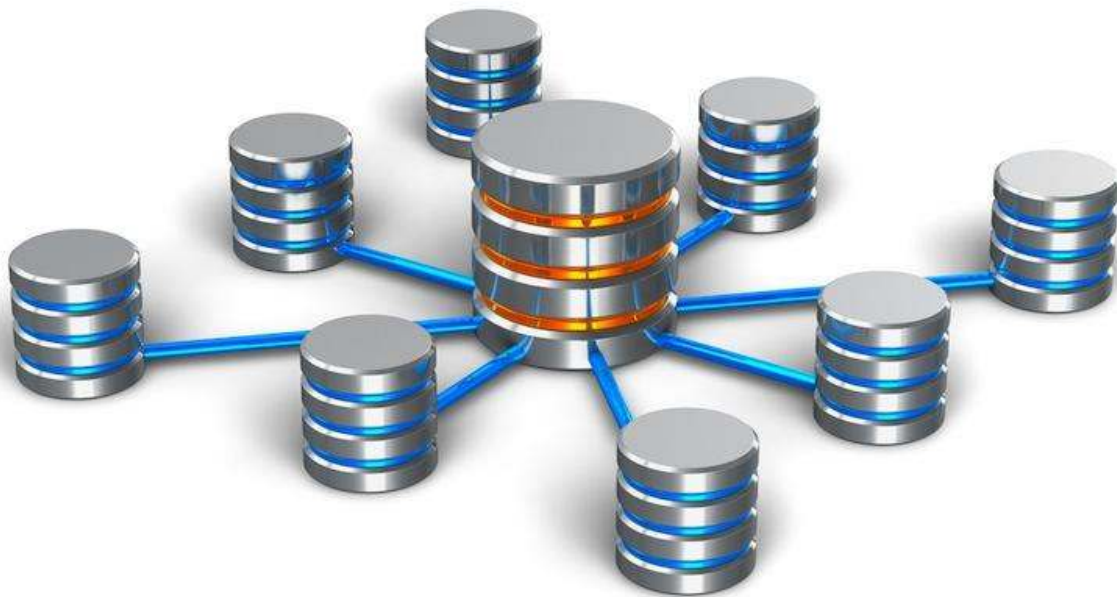




Gestión de Bases de Datos

Versión 1.0

**IES Luis Vélez de Guevara
Departamento de Informática**

21 de octubre de 2019

CONTENIDOS

1. ELEMENTOS DE LAS BASES DE DATOS	1
2. DISEÑO DE MODELOS LÓGICOS NORMALIZADOS	37
3. DISEÑO FÍSICO DE BASES DE DATOS. LENGUAJE DE DEFINICIÓN DE DATOS	95
4. CONSULTA DE BASES DE DATOS. LENGUAJE DE MANIPULACIÓN DE DATOS	147
5. MODIFICACIÓN DE BASES DE DATOS. LENGUAJE DE MANIPULACIÓN DE DATOS	201
6. SEGURIDAD DE LOS DATOS. LENGUAJE DE CONTROL DE DATOS	245

ELEMENTOS DE LAS BASES DE DATOS

1.1 INTRODUCCIÓN

En el entorno del mercado actual, la competitividad y la rapidez de maniobra de una empresa son imprescindibles para su éxito. Para conseguirlo existe cada vez una mayor demanda de datos y, por tanto, más necesidad de gestionarlos. Esta demanda siempre ha estado patente en empresas y sociedades, pero en estos años se ha disparado debido al acceso multitudinario a las redes integradas en Internet y a la aparición de los dispositivos móviles que también requieren esa información.

En informática se conoce como **dato** a cualquier **elemento informativo que tenga relevancia para un usuario**. Desde su nacimiento, la informática se ha encargado de proporcionar herramientas que faciliten la manipulación de los datos. Antes de la aparición de las aplicaciones informáticas, las empresas tenían como únicas herramientas de gestión de datos los ficheros con cajones, carpetas y fichas de cartón. En este proceso manual, el tiempo requerido para manipular estos datos era enorme. Pero la propia informática ha adaptado sus herramientas para que los elementos que el usuario utiliza en cuanto a manejo de datos se parezcan a los manuales. Por eso se sigue hablando de ficheros, formularios, carpetas, directorios,...

La clientela fundamental del profesional informático es la empresa. La empresa se puede entender como un sistema de información formado por diversos objetos: el capital, los recursos humanos, los inmuebles, los servicios que presta, etc.

Los sistemas de información actuales se basan en bases de datos (BD) y **sistemas de bases de datos (SGBD)** que se han convertido en elementos imprescindibles de la vida cotidiana de la sociedad moderna.

1.2 DEFINICIÓN DE BASE DE DATOS

1.2.1 Definición de Base de Datos

Cada día, la mayoría de nosotros nos encontramos con actividades que requieren algún tipo de interacción con una base de datos (ingreso en un banco, reserva de una entrada para el teatro, solicitud de una suscripción a una revista, compra de productos, ...). Estas interacciones son ejemplos de lo que se llama aplicaciones tradicionales de bases de datos (básicamente información numérica o de texto), aunque los avances tecnológicos han permitido que también

existan: bases de datos multimedia, sistemas de información geográfica (GIS), almacenes de datos, sistemas de proceso analítico on-line, ...

- Una **base de datos** se entenderá como una colección de datos relacionados entre sí y que tienen un significado implícito.
- Por **datos** queremos decir hechos conocidos que pueden registrarse y que tienen un significado implícito.

Ejemplo

Una agenda con los nombres y teléfonos de un conjunto de personas conocidas es una base de datos, puesto que es una colección de datos relacionados con un significado implícito.

La definición presentada anteriormente hace referencia a dos elementos para que un conjunto de datos constituya una Base de Datos:

1. **Relaciones entre datos**, tema que se tratará en las secciones siguientes.
2. **Significado implícito** de los datos que se atribuye dependiendo del contexto en que se utilizan los mismos. Por ejemplo, el dato fecha en una base de datos de VENTAS puede referirse a la fecha de emisión de las facturas, mientras que si la base de datos es de MÚSICA quizás corresponda a la fecha en que se grabó un tema musical. Es decir, el significado de un dato, depende de la BD que lo contenga.

Para manipular y gestionar las bases de datos surgieron herramientas software denominadas: sistemas gestores de bases de datos (SGBD en lo sucesivo) sobre los que se profundizará en las siguientes secciones.

1.3 EVOLUCIÓN HISTÓRICA Y TIPOS DE BD

1.3.1 Introducción

Los predecesores de los sistemas gestores de bases de datos fueron los sistemas gestores de ficheros o sistemas de archivos tradicionales.

1. **Archivos tradicionales**. Consiste en almacenar los datos en archivos individuales, exclusivos para cada aplicación particular. En este sistema los datos pueden ser redundantes (repetidos innecesariamente) y la actualización de los archivos es más lenta que en una base de datos.
2. **Base de datos**. Es un almacenamiento de datos formalmente definido, controlado centralmente para intentar servir a múltiples y diferentes aplicaciones. La base de datos es una fuente de datos que son compartidos por numerosos usuarios para diversas aplicaciones.

Así, en un Sistema de archivos tradicional la información está dispersa en varios ficheros de datos y existe un cierto número de programas que los recuperan y agrupan. Aunque los sistemas de ficheros o archivos supusieron un gran avance sobre los sistemas manuales, tienen inconvenientes bastante importantes que se solventaron, en gran medida, con la aparición de los sistemas de bases de datos.

1.3.2 Evolución y tipos de base de datos

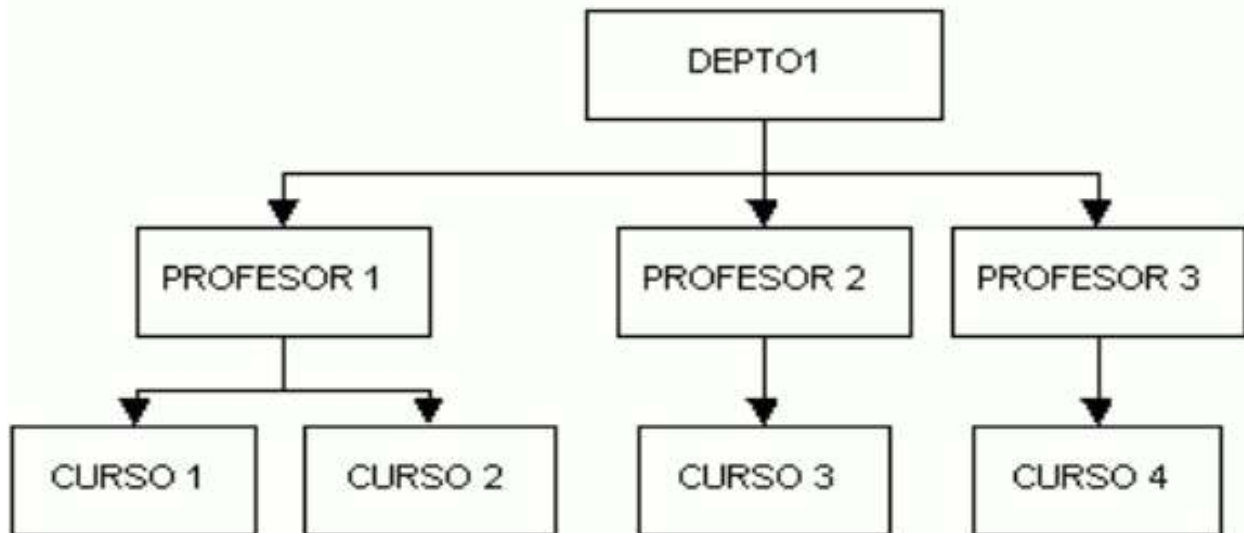
Coincidiendo con la evolución histórica de las bases de datos éstas han utilizado distintos modelos:

- Jerárquicos
- En red.
- Relacionales.
- Multidimensionales.

- De objetos.

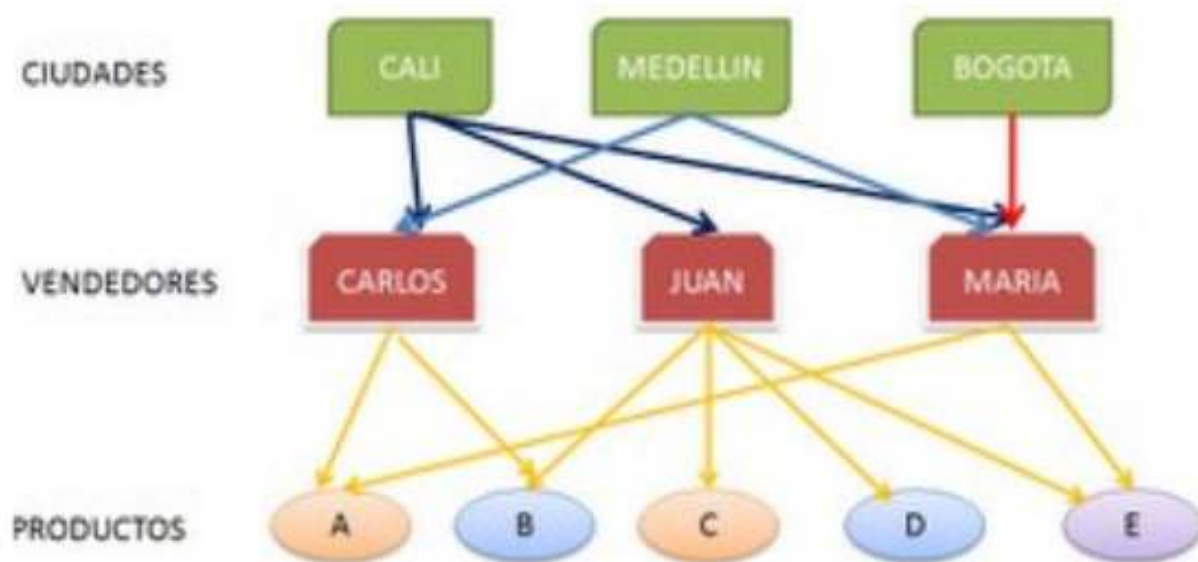
Bases de Datos con estructura jerárquica

La estructura jerárquica fue usada en las primeras BD. Las relaciones entre registros forman una estructura en árbol. Actualmente las bases de datos jerárquicas más utilizadas son IMS de IBM y el Registro de Windows de Microsoft.



Bases de Datos con estructura en red

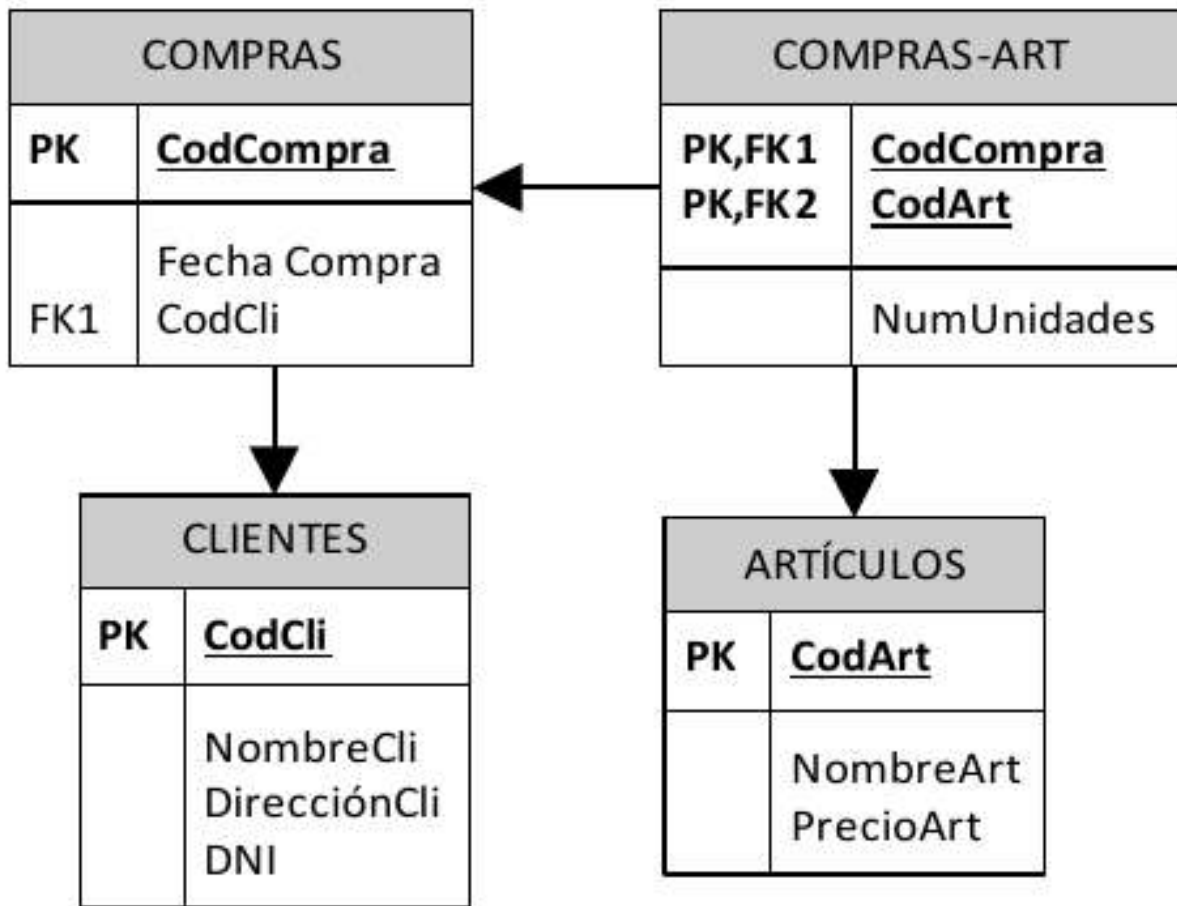
Esta estructura contiene relaciones más complejas que las jerárquicas. Admite relaciones de cada registro con varios que se pueden seguir por distintos caminos.



El inventor de este modelo fue Charles Bachman, y el estándar fue publicado en 1969 por CODASYL.

Bases de Datos con estructura relacional

La estructura relacional es la más extendida hoy en día. Almacena los datos en **filas o registros** (tuplas) y **columnas o campos** (atributos). Estas tablas pueden estar conectadas entre sí por claves comunes.



En este libro nos centramos en el estudio de bases de datos relacionales.

Bases de Datos con estructura multidimensional

La estructura multidimensional tiene parecidos a la del modelo relacional, pero en vez de las dos dimensiones filas-columnas, tiene N dimensiones. Esta estructura ofrece el aspecto de una hoja de cálculo.

	Abril		
	Mayo		
	Junio		
Producto1	212	534	254
Producto2	21	46	33
Producto3	310	321	200
Producto4	120	234	131
Producto5	43	78	55
Producto6	12	32	21
	Argentina	Brasil	Chile

Bases de Datos con estructura orientada a objetos

La estructura orientada a objetos está diseñada siguiendo el paradigma de los lenguajes orientados a objetos. De este modo soporta los tipos de datos gráficos, imágenes, voz y texto de manera natural. Esta estructura tiene gran difusión en aplicaciones web para aplicaciones multimedia.



1.3.3 Sistemas de ficheros tradicionales

En estos sistemas, cada programa almacenaba y utilizaba sus propios datos de forma un tanto caótica. La única ventaja que conlleva esto es que los procesos son independientes, por lo que la modificación de uno no afecta al resto.

Pero tiene grandes inconvenientes:

- **Datos redundantes.** Ya que se repiten continuamente.
- **Coste de almacenamiento elevado.** Al almacenarse varias veces el mismo dato en distintas aplicaciones, se requiere más espacio en los discos.
- **Tiempos de procesamiento elevados.** Al no poder optimizar el espacio de almacenamiento.
- **Probabilidad alta de inconsistencia en los datos.** Ya que un proceso cambia sus datos y no el resto. Por lo que el mismo dato puede tener valores distintos según qué aplicación acceda a él.
- **Difícil modificación en los datos.** Debido a la probabilidad de inconsistencia, cada modificación se debe repetir en todas las copias del dato (algo que normalmente es imposible).

En la siguiente figura se muestra un sistema de información basado en ficheros. En ella se ve que la información aparece inconexa y redundante.

JUGADORES		PORTEROS		BARCELONA		REAL_MADRID		JUG_LIGA_PASADA		JUG_LIGA_ACTUAL	
PK	<u>Codjugador</u>	PK	<u>Codjugador</u>	PK	<u>Codjugador</u>	PK	<u>Codjugador</u>	PK	<u>Codjugador</u>	PK	<u>Codjugador</u>
	Club jugador dorsal codpaís codDem		club jugador dorsal CodPaís NombDem		Club jugador dorsal codpaís codDem		Club jugador dorsal codpaís codDem		Club jugador dorsal codpaís codDem		Club jugador dorsal codpaís codDem
CLUBES		CLUB_JUG		ESTADIOS		EQUIPACIÓN		SOCIOS		ESPONSOR	
PK	<u>CodClub</u>	PK	<u>CodJugador</u>	PK	<u>CodEstadio</u>	PK	<u>CodEquip</u>	PK	<u>Numsocio</u>	PK	<u>CodEspon</u>
	Club Nombre Dirección Población Provincia CosPostal Tlfn Colores Himno Fax AñoFundación presupuesto Presidente Vicepresidente CodEquip Sponsor		CodClub		CodEquipo Estadio Dirección CodPostal Población Provincia Capacidad Sentados Inauguración Dimensiones		NombreEquip Encasa		CodClub Codpost Nombre Apellidos Dirección Provincia FechaAlta CuotaActual		Esponsor
		ENTRENADORES				PAÍSES					
		PK	<u>CodEntren</u>			PK	<u>CodPaís</u>				
			CodClub Entrenador FechaNcmto Población Provincia				Nombrepais Comunitario				
		ENTREN_CLUBES		CAMPEONATO		PALMARÉS		DEMARCACIÓN			
		PK	<u>codCluB</u>	PK	<u>CodTrofeo</u>	PK	<u>CodClub</u> <u>CodTrofeo</u>	PK	<u>CodDem</u>		
			CodEntrenador		NombreTrofeo		Año		Demarcación		

1.3.4 Sistemas de base de datos relacional

En este tipo de sistemas los datos se centralizan en una base de datos común a todas las aplicaciones. Estos serán los sistemas que estudiaremos en este curso.

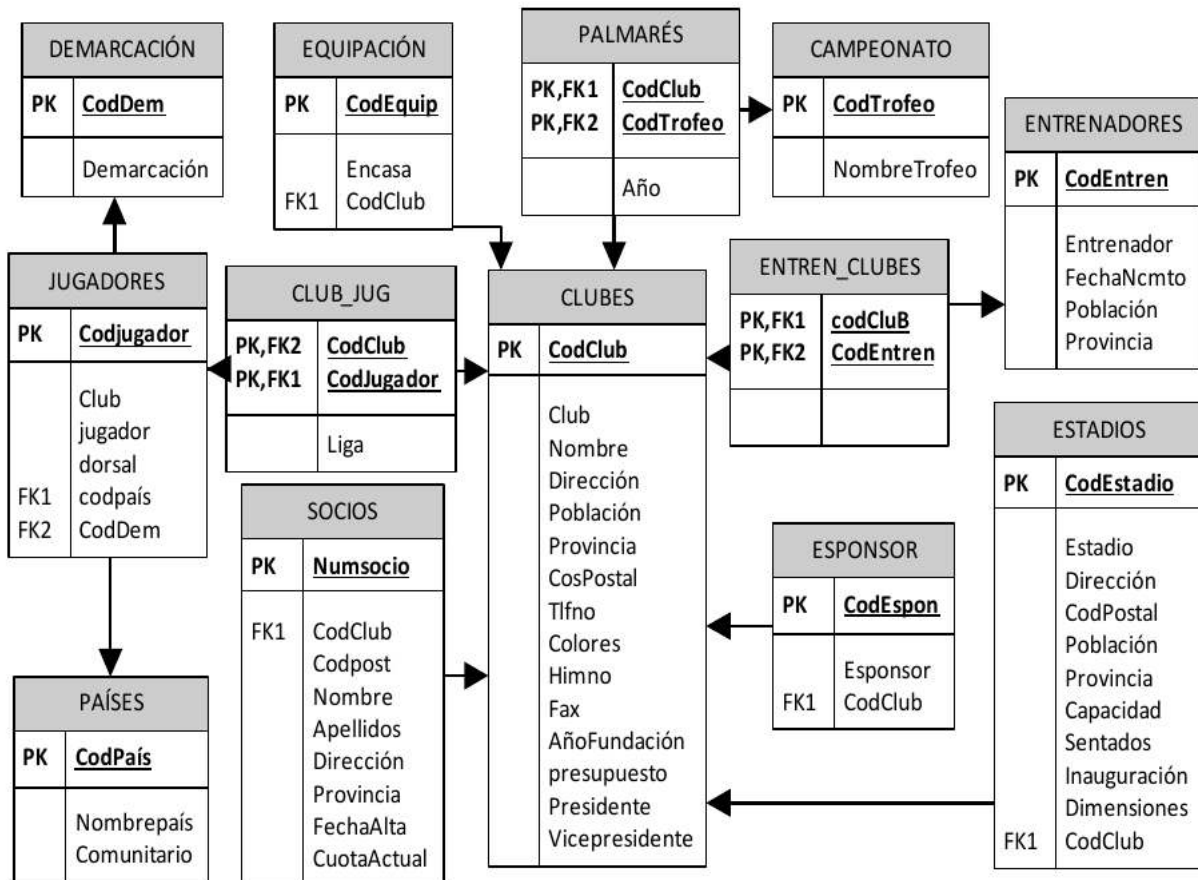
Sus **ventajas** son las siguientes:

- **Menor redundancia.** No hace falta tanta repetición de datos. Aunque, sólo los buenos diseños de datos tienen poca redundancia.
- **Menor espacio de almacenamiento.** Gracias a una mejor estructuración de los datos.
- **Acceso a los datos más eficiente.** La organización de los datos produce un resultado más óptimo en rendimiento.
- **Datos más documentados.** Gracias a los metadatos que permiten describir la información de la base de datos.
- **Independencia de los datos y los programas y procesos.** Esto permite modificar los datos sin modificar el código de las aplicaciones.
- **Integridad de los datos.** Mayor dificultad de perder los datos o de realizar incoherencias con ellos.
- **Mayor seguridad en los datos.** Al limitar el acceso a ciertos usuarios.

Como contrapartida encontramos los siguientes **inconvenientes**:

- **Instalación costosa.** El control y administración de bases de datos requiere de un software y hardware potente.
- **Requiere personal cualificado.** Debido a la dificultad de manejo de este tipo de sistemas.
- **Implantación larga y difícil.** Debido a los puntos anteriores. La adaptación del personal es mucho más complicada y lleva bastante tiempo.

En la siguiente figura se muestra un sistema de información basado en bases de datos. La información está relacionada y no es redundante.



1.3.5 Ejemplo de archivos tradicionales

Se cuenta con dos archivos: CLIENTES y FACTURAS.

El primer archivo tiene los datos básicos de los clientes, mientras que en el segundo se almacenan las ventas realizadas. Al emitir cada factura se ingresan nuevamente los datos num, nombre, domicilio.

Tabla 1: CLIENTES

Num	Nombre	Dirección	Teléfono	FechaNacimiento	oe-mail
1225	Juan García	Guaná 1202	985674863	13/08/1972	jgarcia@adinet.com
1226	Fernando Martí- nez	Rincón 876	984568643	23/02/1987	fmar@gmail.com
...	...	...	...	...	...

Tabla 2: FACTURAS

Num	Nombre	Dirección	Producto	Precio
1225	Joaquín García	Guaná 1202	Azulejos	1250
1226	Fernando Martínez	Rincón 876	Pintura	900
...	...	...	...	...

Desventajas:

- Se presentan **redundancias de datos** (datos repetidos innecesariamente: nombre, dirección). Se duplican esfuerzos.
- Se pueden producir **contradicciones entre los datos**, si por ejemplo se ingresan nombres diferentes para un mismo cliente (Juan por Joaquín).

1.4 CONCEPTOS BÁSICOS DE UNA BD

Resulta fundamental para un Técnico Superior en Informática que conozca los siguientes conceptos básicos:

1.4.1 Datos

Datos son hechos conocidos que pueden registrarse y que tienen un significado implícito. – Ramez Elmasri y Shamkant B. Navathe

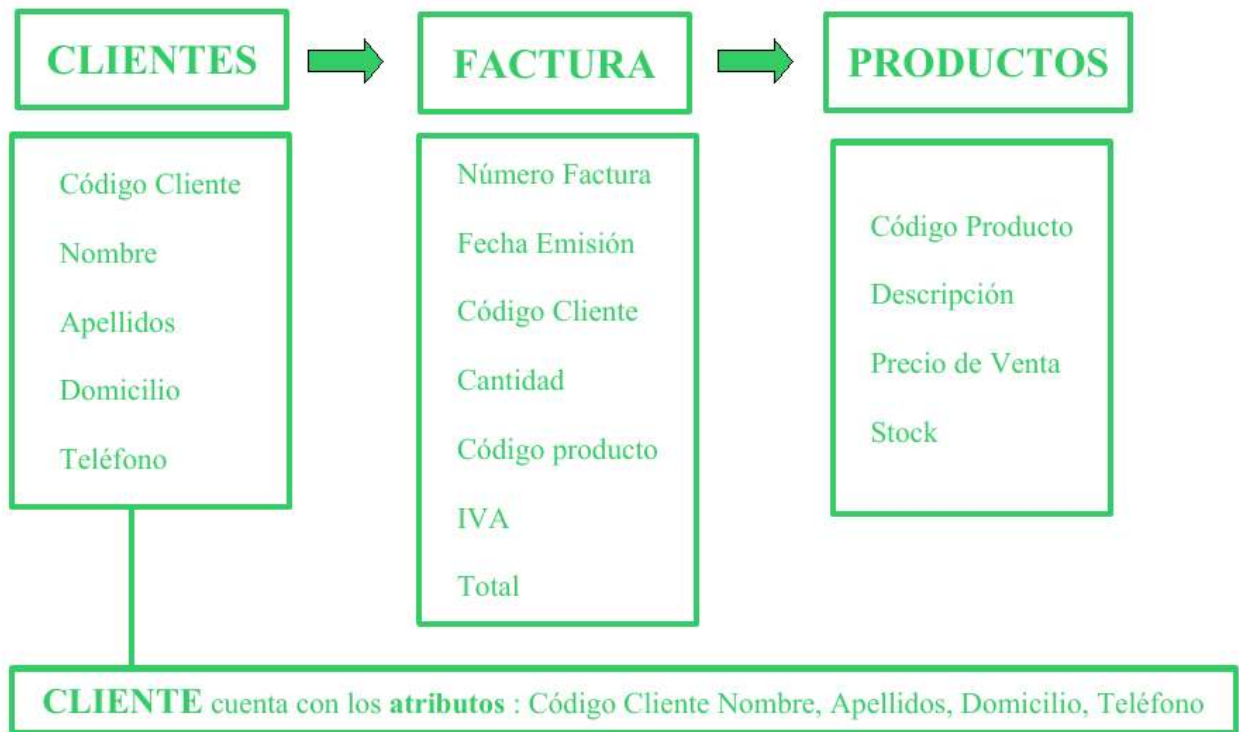
Ejemplo

Pueden constituir datos los nombres, números telefónicos y direcciones de personas que conocemos.



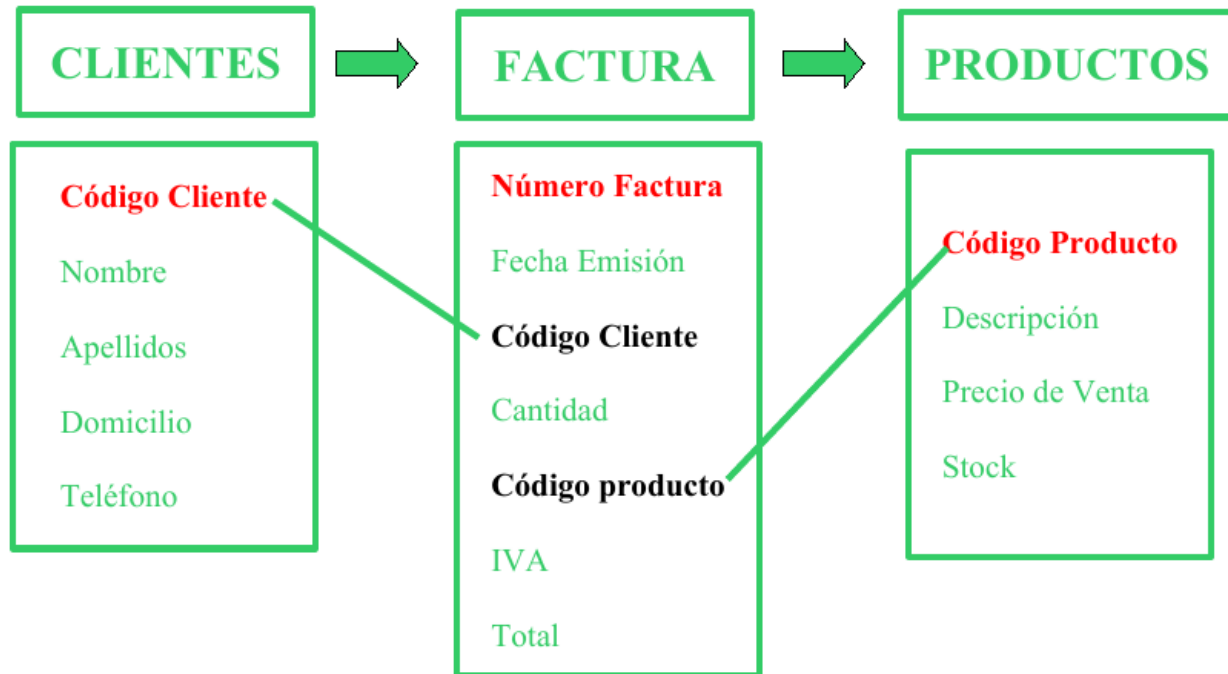
1.4.2 Entidades

Una entidad es todo aquello de lo cual interesa guardar datos, por ejemplo:



1.4.3 Claves primarias y claves foráneas. Relaciones

Cada entidad tiene una **clave primaria** o **campo clave** o **llave** que identifica unívocamente al conjunto de datos. Cuando en una entidad figura la clave primaria de otra entidad, ésta se denomina **clave foránea** o **clave ajena**. Las entidades se **relacionan** entre sí a través de las claves foráneas.



CLAVES PRIMARIAS

- **Código Cliente** es la clave primaria de **CLIENTES**. A cada cliente se le asocia un código y a cada código le corresponde un cliente.
- **Número Factura** es clave primaria de **FACTURAS**.
- **Código Producto** es clave primaria de **PRODUCTOS**.

CLAVES FORÁNEAS

- En **FACTURAS**, son claves foráneas **Código Cliente** y **Código Producto**. **CLIENTES** se relaciona con **FACTURAS** a través del Código Cliente que figura en ambas tablas y con **PRODUCTOS** mediante el Código Producto.

1.4.4 Restricciones de integridad referencial

- Código Cliente en Facturas debe cumplir que exista en Clientes y que sea clave primaria
- Código Producto en Facturas debe cumplir que exista en Productos y que sea clave primaria

Retomando la Definición de Base de Datos, la cual señala que ésta “... es un conjunto de datos relacionados entre sí y que tienen un significado implícito”, se observa en la imagen que los datos de las tablas se relacionan a través de las claves y que éstos tienen el significado implícito que se les atribuye en dicho contexto. Así, por ejemplo, el significado del dato Nombre se refiere al del CLIENTE, el de Fecha emisión a la de la FACTURAS y el de Descripción a la del PRODUCTO.

1.4.5 Metadatos

Metadatos son **datos acerca de los datos** presentes en la base de datos.

Por ejemplo:

- qué tipo de datos se van a almacenar (si son texto o números o fechas ...)

- qué nombre se le da a cada dato (nombre, apellidos, fecha, precio, edad,...)
- cómo están agrupados los datos
- cómo se relacionan,...

Ejemplo de Metadatos:

Dato	Tipo	Longitud
Num	Numérico	4
Nombre	Alfabético	20
.....		

El tipo de datos puede ser Numérico, alfabético, fecha, lógico (SÍ /NO).

La longitud indica la cantidad máxima de caracteres que puede asumir el dato.

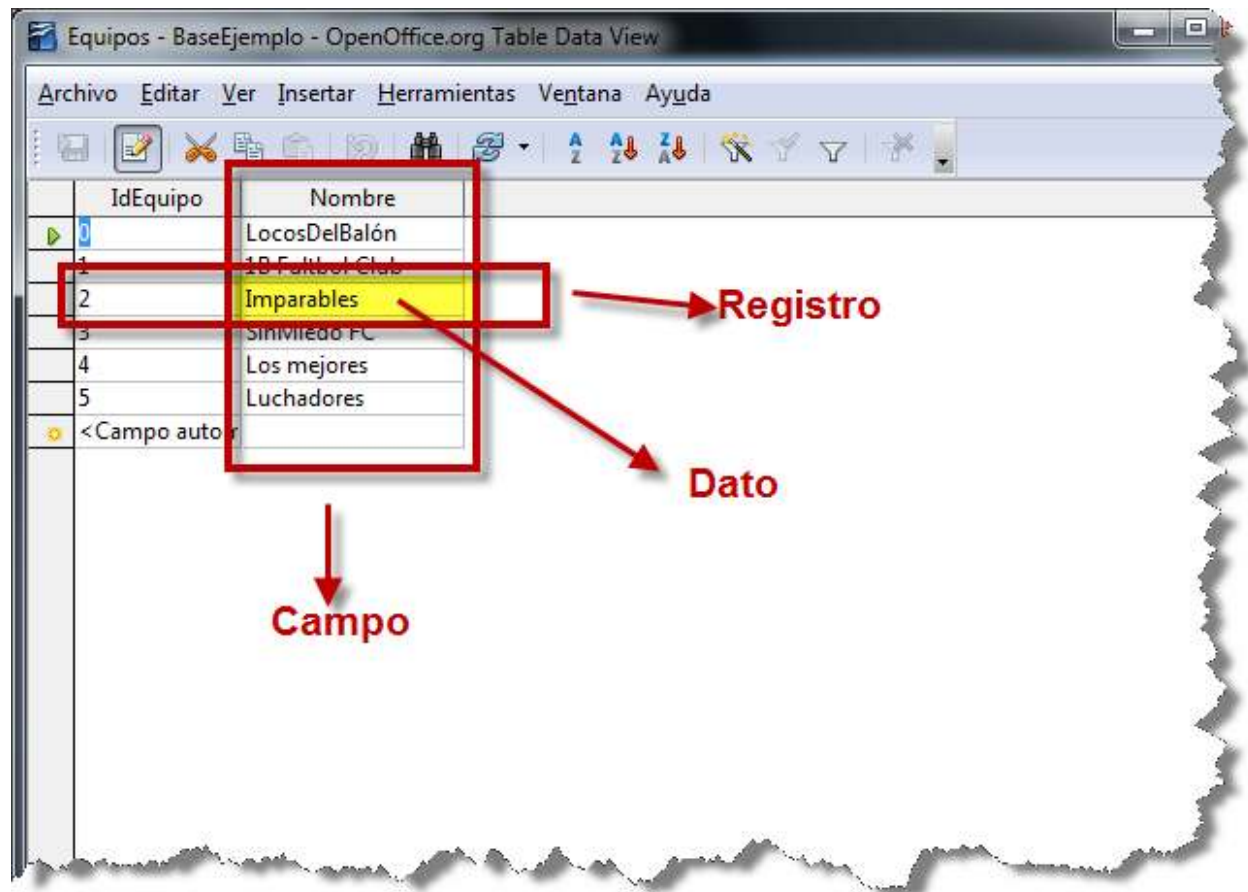
Ejemplo de Restricción de Dominio:

Num >0 y <9999

1.4.6 Otros conceptos sobre Bases de Datos

Además de los conceptos básicos anteriores, a modo de aclarar algunos de los componentes que se pueden encontrar en una base de datos, y que se verán en las próximas unidades, se definen los siguientes conceptos:

- **Tabla:** Es un conjunto de filas y columnas bajo un mismo nombre que representa el conjunto de valores almacenados para una serie de datos. Por ejemplo, la información de todos los clientes de una BD se almacenarán en una tabla llamada CLIENTES.
- **Campo:** Cada una de las **columnas** de una tabla. Identifica una familia de datos. Por ejemplo, el campo fecha-Nacimiento representa las fechas de nacimiento de todos los clientes que contiene una tabla CLIENTES.
- **Registro:** Corresponde a cada una de las **filas** de la tabla. También se llaman tuplas. Por ejemplo en la siguiente tabla CLIENTES, observamos dos registros, que corresponden a la información sobre los clientes Juan García y Fernando Martínez:



- **Tipo de Dato:** El tipo de dato indica la naturaleza del campo. Así, se puede tener datos numéricos, que son aquellos con los que se pueden realizar cálculos aritméticos (sumas, restas, multiplicaciones...), los datos alfa-numéricos, que son los que contienen caracteres alfabéticos y números...
- **Consulta:** Es una instrucción para hacer peticiones a una BD.
- **Índice:** Es una estructura que almacena los campos clave de una tabla, organizándolos para hacer más fácil encontrar y ordenar los registros.
- **Vista:** Se obtienen al guardar una consulta de una o varias tablas. De esta forma se obtiene una tabla virtual, es decir, no está almacenada en los dispositivos de almacenamiento del ordenador, aunque sí se almacena su definición.
- **Informe:** Es un listado ordenado de los campos y registros seleccionados en un formato fácil de leer. Por ejemplo, un informe de las facturas impagadas del mes de enero ordenadas por nombre de cliente.
- **Guiones o scripts:** Son un conjunto de instrucciones, que ejecutadas de forma ordenada, realizan operaciones avanzadas o mantenimiento de los datos almacenados en la BD.
- **Procedimientos:** Son un tipo especial de script que están almacenados en la BD y forman parte de su esquema.

1.5 SISTEMAS DE GESTIÓN DE BASES DE DATOS: TIPOS

1.5.1 Sistema Gestor de Bases de Datos

Un sistema gestor de bases de datos (SGBD) es una aplicación que permite a los usuarios definir, crear y mantener una base de datos, y proporciona acceso controlado a la misma.

En general, un SGBD proporciona los siguientes servicios:

- Permite la **definición de la base de datos** mediante el lenguaje de definición de datos (**DDL – Data Description Language**). Este lenguaje permite especificar la estructura y el tipo de los datos, así como las restricciones sobre los datos. Todo esto se almacenará en la base de datos.
- Permite la **inserción, actualización, eliminación y consulta de datos** mediante el lenguaje de manejo o manipulación de datos (**DML - Data Manipulation Language**).
- Proporciona un acceso controlado a la base de datos mediante:
 - Un sistema de seguridad, de modo que los usuarios no autorizados no puedan acceder a la base de datos, mediante el lenguaje de control de datos (**DCL - Data Control Language**);
 - Un sistema de integridad que mantiene la integridad y la consistencia de los datos;
 - Un sistema de control de concurrencia que permite el acceso compartido a la base de datos;
 - Un sistema de control de recuperación que restablece la base de datos después de que se produzca un fallo del hardware o del software;
 - **Un diccionario de datos o catálogo** accesible por el usuario que contiene la descripción de los datos de la base de datos.

La principal herramienta de un SGBD es la interfaz de programación con el usuario. Esta interfaz consiste en un lenguaje muy sencillo mediante el cual el usuario interactúa con el servidor. Este lenguaje comúnmente se denomina **SQL, Structure Query Language**, está estandarizado por la ISO 1, es decir, todas las BD que soporten SQL deben tener la misma sintaxis a la hora de aplicar el lenguaje.

1.5.2 Tipos de SGBD

Los SGBD se pueden clasificar según las BD que gestionan (jerárquicas, relacionales, orientadas a objetos,...), pero como actualmente la mayoría de los SGBD integran múltiples filosofías, los clasificaremos según su capacidad y potencia del propio gestor, resultado los siguientes SGBD:

- **SGBD ofimáticos:** manipulan BD pequeñas orientadas a almacenar datos domésticos o de pequeñas empresas. Ejemplos típicos son Microsoft ACCESS y LibreOffice Base.
- **SGBD corporativos:** tienen la capacidad de gestionar BD enormes, de medianas o grandes empresas con una carga de datos y transacciones que requieren de un servidor de gran capacidad. Un ejemplo típico de BD corporativas es ORACLE, actualmente junto de DB2 el servidor de BD más potente del mercado (también el más caro). Nosotros para nuestro aprendizaje utilizamos una versión gratuita con fines educativos, que aunque bastante limitada, nos sirve para introducirnos en la filosofía de ORACLE.

1.6 ACTIVIDADES PROPUESTAS

1.6.1 Test

Para cada una de las siguientes cuestiones elige razonadamente cada una de las respuestas correctas.

¿Cuáles de los siguientes puntos representan inconvenientes de los Sistemas de Ficheros?

- a. Redundancia e Inconsistencia.
- b. Sistema de Gestión de Datos independiente de la máquina y del SO.
- c. Control de concurrencia.
- d. Dificil modificación de los datos.

Los sistemas orientados a BD presentan las siguientes ventajas. . .

- a. Integridad de los datos.
- b. Redundancia.
- c. Cada aplicación maneja sus propios datos.
- d. Independencia entre los datos y las aplicaciones que los usan.

Los datos son . . .

- a. . . todo aquello de lo cual interesa guardar información.
- b. . . hechos conocidos que pueden registrarse y que tienen un significado implícito.
- c. . . información acerca de los metadatos.
- d. . . las claves primarias y foráneas de cada entidad.

Un SGBD . . .

- a. . . esta formado por datos acerca de los datos presentes en la base de datos.
- b. . . es una aplicación que permite a los usuarios definir, crear y mantener una base de datos, y proporciona acceso controlado a la misma.
- c. . . permite a los usuarios tener acceso a la BD completa impidiendo restricciones.
- d. . . permite la inserción, actualización, eliminación y consulta de datos mediante el lenguaje de manejo o manipulación de datos.

1.6.2 Cuestiones

Contesta los siguientes apartados

1. Explica brevemente los antecedentes de las BD actuales.
2. Enumera y explica brevemente los inconvenientes que presentan los antiguos sistemas de Archivos o de Ficheros.

¿Qué significa que los datos de una BD tienen un significado implícito? Por tres ejemplos, diferentes de los expuestos en el tema, que ilustren tu explicación.

Definir que es un SGBD

Indica las principales ventajas de las BD frente a los antiguos sistemas de ficheros.

Nombre los distintos tipos de bases de datos que existen según el modelo que siguen

¿Qué son las vistas? ¿Para qué se utilizan?. Busca información en Internet para completar tu respuesta. ¿En qué se diferencia de una consulta?

Describe el significado de las siguientes siglas: DDL, DML y DCL. Explica la utilidad de cada una.

¿Qué es un script o guión?

Define los siguientes conceptos:

1. Dato
2. Tipo de Dato
3. Campo
4. Registro
5. Tabla
6. Relación
7. Consulta
8. Procedimiento

¿Qué es el diccionario de datos?

¿Qué quiere decir que una base de datos permita la concurrencia?

Utilizando el archivo “La biblia de Access” disponible en la moodle, contesta a las siguientes preguntas:

1. ¿Cuál es la extensión de un fichero que contiene la base de datos de ACCESS?
2. Describe dos formas de crear un formulario en ACCESS.
3. Indica los 10 tipos de datos básicos que existen en ACCESS.
4. Describe dos formas de crear una consulta en ACCESS.

1.6.3 Prácticas

PRÁCTICA 1

PLANTEAMIENTO

OBJETIVOS: Comparar un Sistema de Ficheros con un Sistema basado en BD.

ENUNCIADO: Se plantea un problema real y se muestra la solución dada al mismo utilizando un sistema de ficheros. Se analizará dicha solución y se detectarán en ella ejemplos de cada uno de los inconvenientes visto en teoría para los Sistemas de Ficheros. Es decir, buscaremos ejemplos en la solución propuesta que ilustren los siguientes inconvenientes:

- Coste de almacenamiento elevado.
 - Datos redundantes.
 - Probabilidad alta de inconsistencia en los datos.
 - Dificil modificación en los datos.
 - Tiempos de procesamiento elevados.
-

PROBLEMA REAL

Una empresa se encarga de dar publicidad a los inmuebles que ofrece en alquiler, tanto en prensa local como nacional, entrevista a los posibles inquilinos, organiza las visitas a los inmuebles y negocia los contratos de alquiler. Una vez firmado el alquiler, la empresa asume la responsabilidad del inmueble, realizando inspecciones periódicas para comprobar su correcto mantenimiento. A continuación se describen los datos que se manejan en las oficinas de la empresa para llevar a cabo el trabajo diario.

OFICINAS

La empresa tiene varias oficinas en todo el país. Cada oficina tiene un código de identificación que es único, tiene una dirección (calle, número y ciudad), un número de teléfono y un número de fax. Cada oficina tiene su propia plantilla.

PLANTILLA

Cada oficina tiene un director que se encarga de supervisar todas sus gestiones. La empresa sigue muy de cerca el trabajo de los directores y tiene registrada la fecha en que cada director empezó en el cargo en su oficina. Cada director tiene un pago anual por gastos de vehículo y una bonificación mensual que depende de los contratos de alquiler que haya realizado su oficina. En cada oficina hay varios supervisores. Cada uno es responsable del trabajo diario de un grupo de entre cinco y diez empleados que realizan las gestiones de los alquileres. El trabajo administrativo de cada grupo lo lleva un administrativo. Cada miembro de la plantilla tiene un código único que lo identifica en la empresa. De cada uno de ellos se quiere conocer el nombre, la dirección, el número de teléfono, la fecha de nacimiento, el número del DNI, su puesto en la empresa, el salario anual y la fecha en que entró en la empresa. De los administrativos se desea conocer también la velocidad con que escriben a máquina (en pulsaciones por minuto). Además, de cada empleado se debe guardar información sobre uno de sus parientes más próximos: nombre, relación con el empleado, dirección y número de teléfono.

INMUEBLES

Cada oficina de la empresa tiene una serie de inmuebles para alquilar. Estos inmuebles se identifican por un código que es único dentro de la empresa. Los datos que se guardan de cada inmueble son los siguientes: dirección completa (calle, número y ciudad), tipo de inmueble, número de habitaciones y precio del alquiler en euros (este precio es mensual). El precio del alquiler se revisa de forma anual. Cada inmueble se asigna a un empleado que es el responsable de su gestión. Cada miembro de la plantilla puede tener asignados hasta veinte inmuebles para alquilar.

PROPIETARIOS

Los propietarios de los inmuebles pueden ser particulares o empresas. A cada propietario se le asigna un código que es único en la empresa. De los particulares se guarda el nombre, la dirección y el número de teléfono. De las empresas se guarda el nombre comercial, tipo de empresa, la dirección, el número de teléfono y el nombre de la persona de contacto.

INQUILINOS (CLIENTES)

Cuando un cliente contacta con la empresa por primera vez, se toman sus datos: nombre, dirección, número de teléfono, tipo de inmueble que prefiere e importe máximo que está dispuesto a pagar al mes por el alquiler. Ya que es un posible inquilino, se le asigna un código que es único en toda la empresa. De la entrevista inicial que se realiza con cada cliente se guarda la fecha, el empleado que la realizó y unos comentarios generales sobre el posible inquilino.

VISITAS A LOS INMUEBLES

En la mayoría de los casos, los posibles inquilinos desean ver varios inmuebles antes de alquilar uno. De cada visita que se realiza se guarda la fecha y los comentarios realizados por el cliente respecto al inmueble.

ANUNCIOS

Cuando algún inmueble es difícil de alquilar, la empresa lo anuncia en la prensa local y nacional. De cada anuncio se guarda la fecha de publicación y el coste económico del anuncio. De los periódicos se guarda el nombre, la dirección, el número de teléfono, el número de fax y el nombre de la persona de contacto.

CONTRATOS DE ALQUILER

La empresa se encarga de redactar los términos de cada contrato de alquiler. Cada contrato tiene un número, un importe mensual, un método de pago, el importe del depósito, si se ha realizado el depósito, las fechas de inicio y finalización del contrato, la duración del contrato en meses y el miembro de la plantilla que lo formalizó. La duración mínima de un contrato es de tres meses y la duración máxima es de un año. Cada cliente puede tener alquilados uno o varios inmuebles al mismo tiempo.

INSPECCIONES

Como parte del servicio que presta la empresa, ésta se encarga de realizar inspecciones periódicas a los inmuebles para asegurarse de que se mantienen en buen estado. Cada inmueble se inspecciona al menos una vez cada seis meses. Se inspeccionan tanto los inmuebles alquilados, como los que están disponibles para alquilar. De cada inspección se anota la fecha y los comentarios sobre su estado que quiera incluir el empleado que la ha llevado a cabo.

ACTIVIDADES DE CADA OFICINA

En cada oficina se llevan a cabo las siguientes actividades para garantizar que cada empleado tenga acceso a la información necesaria para desempeñar su tarea de modo efectivo y eficiente. Cada actividad está relacionada con una función específica de la empresa. Cada una de estas funciones corresponde a uno o varios puestos de los que ocupan los empleados, por lo que éstos se indican entre paréntesis.

SOLUCIÓN PROPUESTA SEGÚN EL ANTIGUO SISTEMA DE FICHEROS

DEPARTAMENTO DE VENTAS

En esta inmobiliaria, el departamento de ventas se encarga de alquilar inmuebles. Por ejemplo, cuando un propietario pasa por el departamento de ventas para ofrecer en alquiler su piso, se rellena un formulario en donde se recogen los datos del piso, como la dirección y el número de habitaciones, y los datos del propietario. El departamento de ventas también se encarga de atender a los clientes que desean alquilar un inmueble. Cuando un cliente (posible inquilino) pasa por este departamento se rellena un formulario con sus datos y sus preferencias: si quiere un piso o una casa, el importe mensual que está dispuesto a pagar por el alquiler, etc. Para gestionar toda esta información, el departamento de ventas posee un sistema de información. El sistema del departamento de ventas tiene tres ficheros: fichero de inmuebles, fichero de propietarios y fichero de inquilinos.

INMUEBLE

Inum	Calle	Área	Población	Tipo	Hab.	Alquiler	Pnum
IA14	En medio, 128	Centro	Castellón	Casa	6	600	P46
IL94	Riu Ebre, 24	Ronda Sur	Castellón	Piso	4	350	P87
IG4	Sorell, 5	Grao	Castellón	Piso	3	300	P40
IG36	Alicante, 1		Segorbe	Piso	3	325	P93
IG21	San Francisco, 10		Vinaroz	Casa	5	550	P87
IG16	Capuchinos, 19	Rafalafena	Castellón	Piso	4	400	P93

PROPIETARIO

Pnum	Nombre	Apellido	Dirección	Teléfono
P46	Amparo	Felipe	Asensio 24, Castellón	964 230 680
P87	Manuel	Alejandro	Av.Libertad 15, Vinaroz	964 450 760
P40	Alberto	Estrada	Av.del Puerto 52, Castellón	964 200 740
P93	Yolanda	Robles	Purísima 4, Segorbe	964 710 430

INQUILINO

Qnum	Nombre	Apellido	Dirección	Teléfono	Tipo	Alquiler
Q76	Juan	Felip	Barceló 47, Castellón	964 282 540	Piso	375
Q56	Ana	Grangel	San Rafael 45, Almazora	964 551 110	Piso	300
Q74	Elena	Abaso	Navarra 76, Castellón	964 205 560	Casa	700
Q62	Alicia	Mori	Alloza 45, Castellón	964 229 580	Piso	550

DEPARTAMENTO DE CONTRATOS

El departamento de contratos se ocupa de gestionar los contratos de alquiler de los inmuebles. Cuando un cliente desea formalizar un contrato, un empleado de la empresa rellena un formulario con los datos del inquilino y los datos del inmueble. Este formulario se pasa al departamento de contratos, que asigna un número al contrato y completa la información sobre el pago y el período del contrato. Para gestionar esta información, el departamento de contratos posee un sistema de información con tres ficheros: el fichero de los contratos, el fichero de los inmuebles alquilados y el fichero de los inquilinos que tienen en vigor un contrato de alquiler.

CONTRATO

Cnum	Inum	Qnum	Importe	Pago	Depósito	Pagado?	Inicio	Fin	Meses
10024	IA14	Q62	600	Visa	1200	S	1/6/99	31/5/00	12
10075	IL94	Q76	350	Efectivo	700	N	1/1/00	30/6/00	6
10012	IG21	Q74	550	Cheque	1100	S	1/7/99	30/6/00	12

INMUEBLE

Inum	Calle	Área	Población	Alquiler
IA14	Enmedio, 128	Centro	Castellón	600
IL94	Riu Ebre, 24	Ronda Sur	Castellón	350
IG21	San Francisco, 10		Vinaroz	550

INQUILINO

Qnum	Nombre	Apellido	Dirección	Población	Teléfono
Q76	Juan	Felip	Barceló, 47	Castellón	964 282 540
Q74	Elena	Abaso	Navarra, 76	Castellón	964 205 560
Q62	Alicia	Mori	Alloza, 45	Castellón	964 229 580

Importante: Al tratarse de una solución basada en Sistemas de ficheros, cada departamento accede a sus propios ficheros mediante una serie de programas de aplicación escritos especialmente para ellos. Estos programas son totalmente independientes entre un departamento y otro, y se utilizan para introducir datos, mantener los ficheros y generar los informes que cada departamento necesita. Es importante destacar que la estructura física de los ficheros de datos y de sus registros está definida dentro de los programas de aplicación.

La situación es muy similar en el resto de departamentos:

DEPARTAMENTO DE NÓMINAS

En el departamento de nóminas tiene un fichero con los datos de los salarios de los empleados. Los registros de este fichero tienen los siguientes campos: número de empleado, nombre, apellido, dirección, fecha de nacimiento, salario, DNI y número de la oficina en la que trabaja.

DEPARTAMENTO DE PERSONAL

El departamento de personal tiene un fichero con los datos de los empleados. Sus registros tienen los siguientes campos: número de empleado, nombre, apellidos, dirección, teléfono, puesto, fecha de nacimiento, salario, DNI y número de la oficina en la que trabaja.

CUESTIONES

1. Completa una tabla buscando ejemplos en el enunciado de cada uno de los inconvenientes estudiados en los sistemas de ficheros. Deberás completar la respuesta con una breve explicación que ilustre el porqué dicho ejemplo presenta el inconveniente en cuestión.

Tabla 3: INCONVENIENTES

INCONVENIENTE	EJEMPLO	EXPLICACIÓN
Coste de almacenamiento elevado		
Datos redundantes		
Probabilidad alta de inconsistencia de los datos		
Difícil modificación de los datos		
Tiempo de procesamiento elevado		

Nota: La tabla la deberás entregar en formato apaisado para que puedas dar las explicaciones sin límite de espacio.

2. Supongamos ahora que quisiéramos resolver el problema real usando el enfoque actual de BD. Detecta en el enunciado ejemplos de los siguientes elementos fundamentales de una BD:

- Datos
- Entidades
- Claves principales
- Claves foráneas
- Integridad referencial
- Metadatos

Nota: Los tres últimos elementos pueden ser más difíciles de encontrar. Si es así, no os preocupéis porque estamos empezando.

3. Utiliza los ejemplos sobre datos detectados en el enunciado para explicar el concepto de “significado implícito” de los mismos.

PRÁCTICA 2

PLANTEAMIENTO

OBJETIVO: Adentrarse en el diseño e implementación de BD a través de un ejemplo guiado. La realización autónoma de este tipo de prácticas será el objetivo esencial de este módulo.

ENUNCIADO: Se plantea el enunciado de una base de datos y los esquemas conceptuales y lógicos que resultan y que ¡pronto sabrás realizar por ti mismo!. A partir de ellos, se explicará paso a paso como crear las tablas y las relaciones entre las mismas. Cómo introducir los datos y cómo realizar algunas consultas sencillas sobre ellos. Cuando finalicemos este módulo sabrás hacer todo esto solo y además en lenguaje SQL!!

PROBLEMA: Se quiere realizar una BD para una empresa dedicada a la comercialización de cocinas.

Ejercicio

- a. Detecta en el enunciado anterior todas las ENTIDADES y DATOS. Indica a qué tipo de Base de Datos de entre los vistos en el tema, corresponde la solución planteada.

Al igual que antes de construir un edificio, se deben pasar por una serie de fases previas:

- recogerse con detalle las características que debe reunir,
- realizar los planos necesarios y una especificación de calidades,... antes de realizar una base de datos realizaremos también un diseño previo.

El Diseño de base de datos, se verá con detalle a partir de la próxima unidad. No obstante, y a modo de introducción, nos acercamos al mismo en el siguiente ejemplo guiado. Para entenderlo, hacemos una breve introducción de las distintas fases por las que pasa el análisis y el diseño de una BD:

Fase de análisis: Especificación de Requisitos Software o E.R.S.

Los informáticos se reúnen con los futuros usuarios del sistema para recopilar la información que necesitan para saber que desean dichos usuarios.

Supongamos que después de unas entrevistas previas, obtenemos que la empresa lo que desea es lo siguiente:

Especificación de requisitos

La empresa desea realizar un control de sus ventas y montajes, para lo cual se tiene en cuenta:

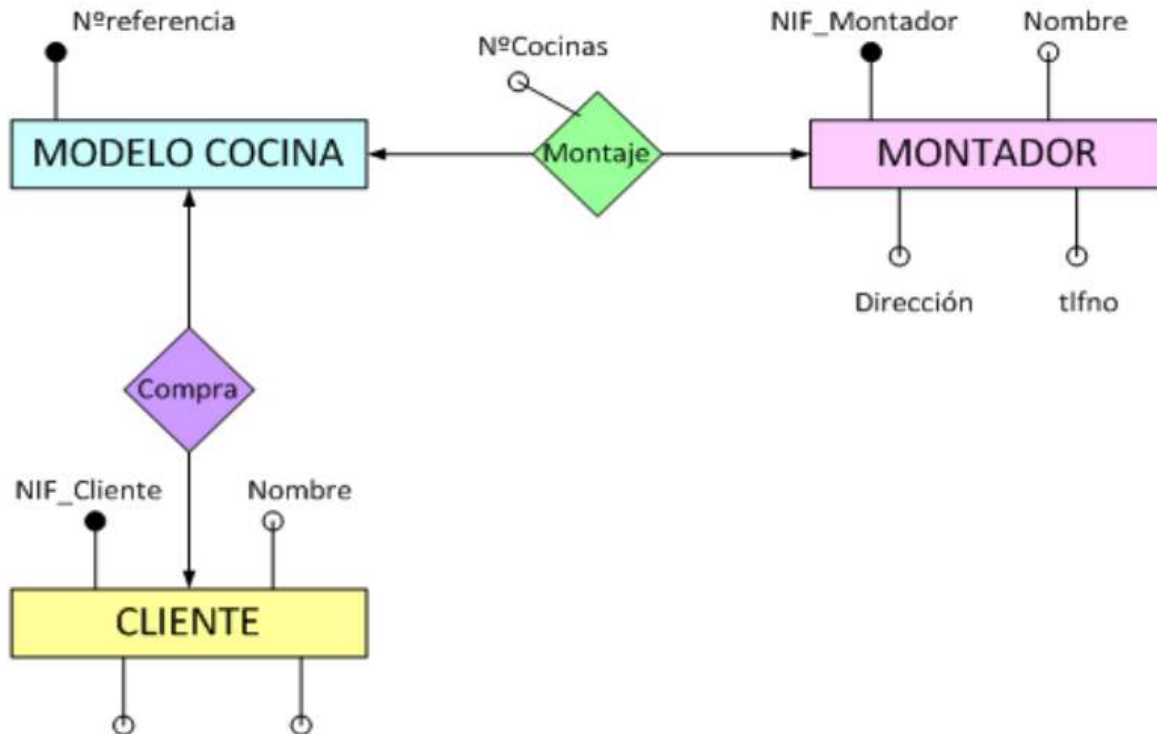
- De cada modelo de cocina nos interesa el número de referencia del modelo.
- De un montador nos interesa su NIF, nombre, dirección, teléfono de contacto y el número de cocinas que ha montado de cada modelo.
- Cada modelo cocina lo debe montar al menos un montador, y el mismo montador puede montar varios modelos, porque no se especializan en ninguno en concreto.
- De un cliente nos interesa su NIF, nombre, dirección y teléfono. Cada modelo de cocina pueden comprarlo uno o varios clientes, y el mismo cliente puede comprar varias modelos de cocinas.

Fase 1 del Diseño: Diseño Conceptual.

A partir de la E.R.S., se diseñará un modelo que tienen un gran poder expresivo para poder comunicarse con el usuario que no experto en informática. El modelo que utilizaremos en este módulo y que explicaremos en la siguiente unidad es el modelo Entidad/relación.

Diseño Conceptual

A partir de la E.R.S, que supone una descripción del mundo real sobre el que queremos diseñar nuestra base de datos, el primer paso será diseñar el esquema conceptual que lo describe.



Es algo parecido al paso previo que realizan los arquitectos al crear el plano de un edificio antes de construirlo. Tiene sus propios símbolos que deben conocer todos los arquitectos para entender el plano. Al igual que ellos, vosotros tendréis que aprender a conocer los símbolos que utilizaréis e interpretaréis para poder diseñar una BD. Estos símbolos se aprenderán en el Tema siguiente, pero en este y a modo de introducción, se presenta como quedaría la interpretación del mundo real de nuestro problema mediante el esquema conceptual llamado entidad/relación.

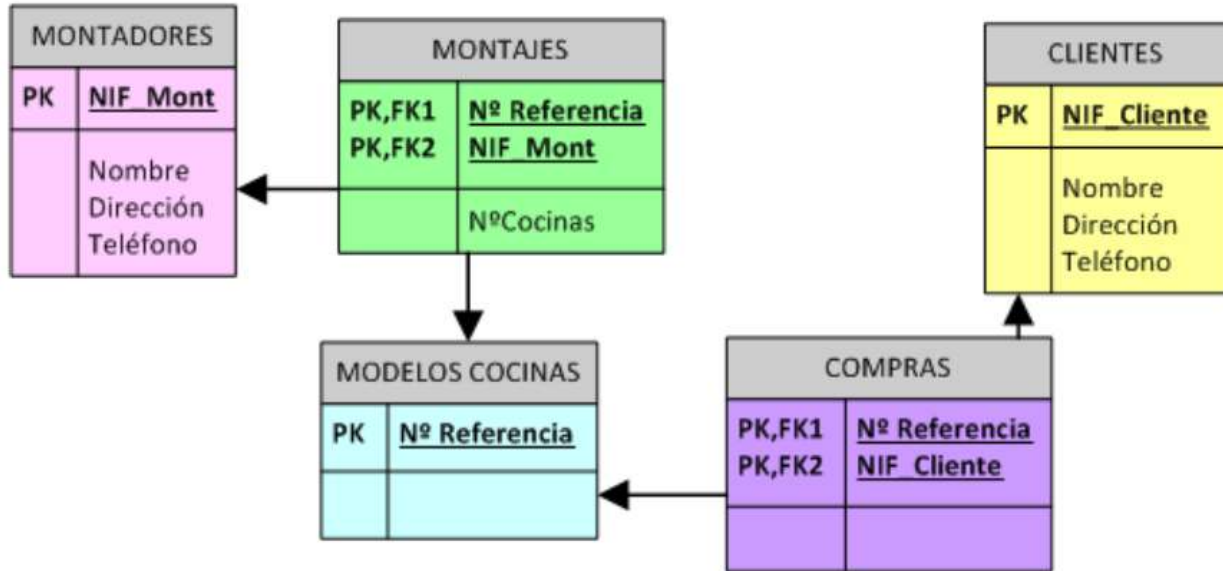
- b) ¿Cómo crees que hemos representado las entidades en el esquema anterior?
- c) ¿Cómo crees que hemos representado los datos en el esquema anterior?

Fase 2 del diseño: Diseño Lógico.

A partir del modelo entidad/relación se creará un modelo que suele ser más difícil de entender para el usuario final y que generalmente tiene una traducción directa al modelo físico en que entiende el SGBD. El modelo lógico elegido dependerá de la BD, pues no es lo mismo modelizar una BD orientada a objetos que una BD relacional. El modelo que utilizaremos en este módulo es el modelo relacional.

Diseño Lógico

A partir del esquema conceptual, aprenderemos a obtener el esquema lógico, el cual va a depender del SGBD que utilicemos. En nuestro caso nos basaremos en el modelo relacional que es el más extendido. De nuevo, y a modo de ejemplos de ¡lo que seréis capaces de hacer en breve! os presento como quedaría el esquema relacional del ejemplo anterior.



Cada una de las “cajas” representadas en el esquema anterior recibirá el nombre de relación (por eso Modelo Relacional). Una relación no es otra cosa que una tabla y en ella se transformará en el diseño físico (Creación de la Bd en un SGBD)

- d) ¿Detectas algún ejemplo en el modelo que ilustre el concepto de «significado implícito» de los datos de una BD?

Fase 3 del diseño: Diseño físico.

Es el resultado de aplicar el modelo lógico a un SGBD concreto. Generalmente está expresado en un lenguaje de programación de BBDD tipo SQL. Aunque en este primer ejemplo introductorio utilizaremos como SGBD Access, que se basa en herramientas gráficas para implementar la BD.

Diseño Físico

A partir del esquema lógico, aprenderemos a crear físicamente nuestra BD en el SGBD. Para interpretar el esquema sólo es necesario saber que:

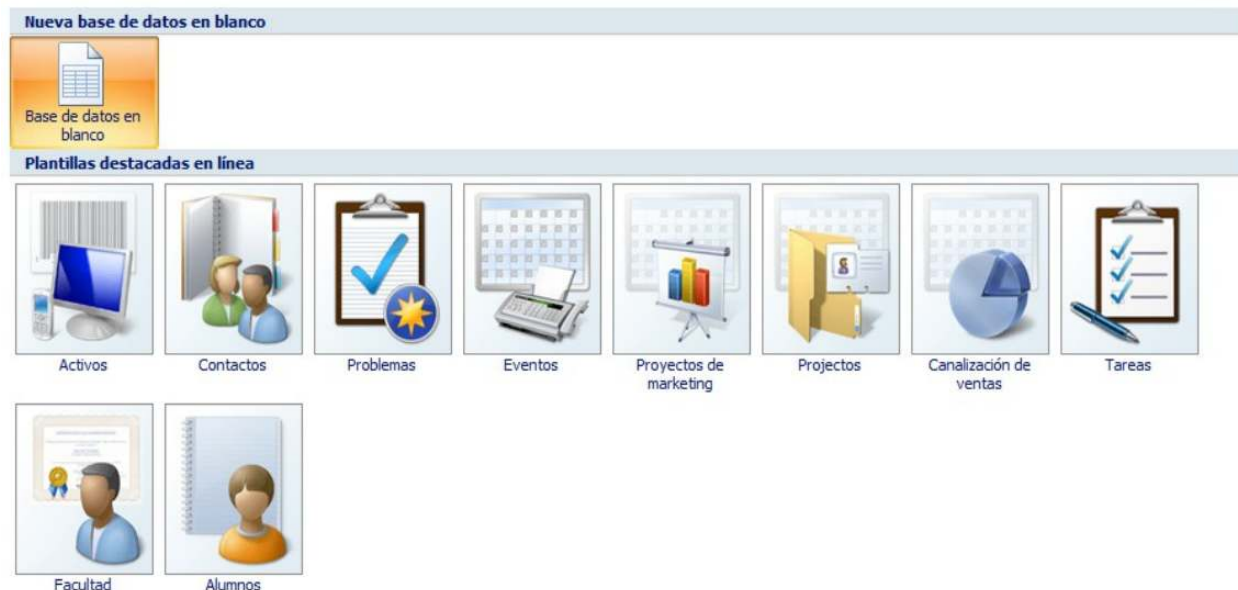
- Cada relación o caja será una tabla en nuestro SGBD.
- Los campos precedidos de “PK” serán las claves principales en nuestras tablas.
- Los campos precedidos de “FK” serán las claves foráneas en nuestras tablas.

Nota: Para resolver la última parte de esta práctica utilizaremos como SGBD Microsoft ACCESS. Aunque es una aplicación muy extendida no es una herramienta verdaderamente potente para un Técnico informático. Es por eso que sólo la usaremos en las secciones introductorias y cuando nos adentremos en el módulo, pasaremos al uso de ORACLE. En concreto vamos a usar el manual de ACCESS 2007 y podréis consultarlo en todo momento ya que estará a vuestra disposición en la MOODLE.

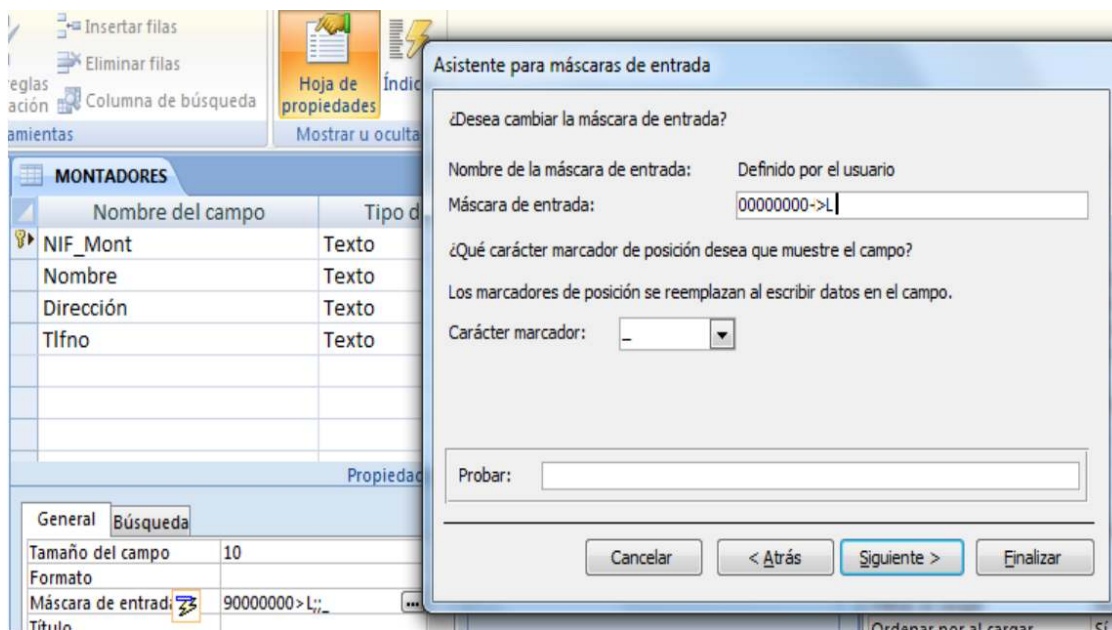
Paso a Tablas usando Access

1. Entraremos en ACCESS y crearemos una nueva base de datos en blanco a la que pondremos por nombre Práctica
2. A continuación iremos a Crear → Diseño de tabla y procederemos a crear una a una las siguientes tablas:

Introducción a Microsoft Office Access



Utiliza las capturas de pantalla y el manual de ACCESS para crear las máscaras de entrada correspondientes. Se crearán máscaras para el NIF y el tlfno, del tipo 59567840-T y 676 987 659, tanto en las tablas MONTADORES como CLIENTES.



CLIENTES		
Nombre del campo	Tipo de datos	Descripción
NIF_Clientes	Texto	
Nombre	Texto	
Dirección	Texto	
Tlfno	Texto	

Propiedades del campo

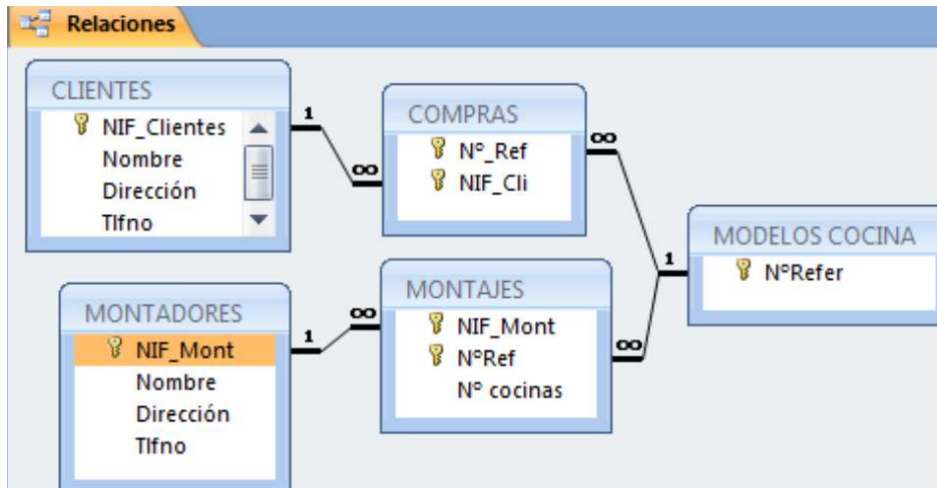
General	Búsqueda
Tamaño del campo	10
Formato	
Máscara de entrada	00000000\->L;0;_

MODELOS COCINA	
Nombre del campo	Tipo de datos
NºRefer	Autonumérico

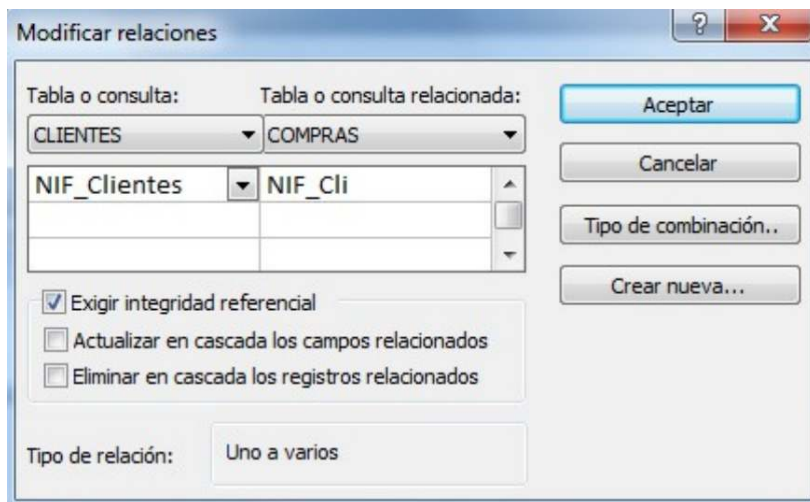
MONTAJES		
Nombre del campo	Tipo de datos	
NIF_Mont	Texto	
NºRef	Texto	
Nº cocinas	Texto	

COMPRAS	
Nombre del campo	Tipo de datos
Nº_Ref	Texto
NIF_Cli	Texto

- e) A partir de las imágenes anteriores ¿qué identificarías como metadatos?
- f) Crea las tablas anteriores en ACCESS.
2. A continuación estableceremos las relaciones entre las tablas según el siguiente esquema antes de poder introducir los datos.



- g) Crea las relaciones en ACCESS: Herramientas de Bases de Datos-> Relaciones-> Agregar las tablas. Para establecer las relaciones deberás pinchar con el ratón sobre la clave principal y, sin soltar, ponerte sobre la clave foránea. Una vez sobre ella soltarás y marcarás la casilla “Exigir integridad referencial”



3. Ya estamos listos para introducir datos. Para hacerlo pincharemos sobre las tablas y accederemos a ellas en “Vista hoja de datos”. Se introducirán los datos siguientes:

CLIENTES

NIF_Cliente ▾	Nombre_C ▾	Dirección_C ▾	Teléfono_C ▾
52567898-S	Agustín Moreno Carvajal	Ronda, 13	957234516
54678963-L	Lucía Rosique Fernández	Alfonso XII, 25	957897654
67545679-V	Martín Romero Wic	Jalón, 2	957472658
78654332-S	Rafael López Pérez	Cervantes, 14	957456432
87654322-A	Patricia Mena Vicente	Sevilla, 23	957654321
89765432-R	Antonio Fernández Martínez	Sastres, 7	957675432

MONTADORES

NIF_M ▾	Nombre_M ▾	Dirección_M ▾	Teléfono_I ▾
67589799-G	Luis Angulo Boza	San Martín, 2	957654532
78654568-Z	Mario Fernández Cacho	Almodóvar, 34	956781111
87654329-F	Javier González Pérez	Avenida de Cádiz, 45	957897676
87698762-S	Román Boza Núñez	Marruecos, 56	957676767

MODELOS COCINAS

Num-Ref ▾
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

MONTAJES

Num_Ref ▾	NIF_M ▾	Número de ▾
1	67589799-G	1
2	87654329-F	2
19	67589799-G	1
3	87698762-S	1
3	78654568-Z	1
5	78654568-Z	1
6	87654329-F	2
6	87698762-S	1
7	67589799-G	1

COMPRAS

Num_Ref ▾	NIF_C ▾
1	52567898-S
2	54678963-L
2	89765432-R
3	78654332-S
3	87654322-A
5	87654322-A
6	52567898-S
6	54678963-L

4. El verdadero sentido de tener nuestros datos almacenados en una BD es poder consultarlos en caso de necesidad. A continuación vamos a ver ejemplos guiados de consultas de datos. Vamos a ver ahora como se realizan algunas consultas sencillas. Iremos a las opciones Crear-> Diseño de consultas

- h) Muestra los nombres y teléfonos de todos los montadores. Sólo habrá que mostrar la tabla MONTADORES



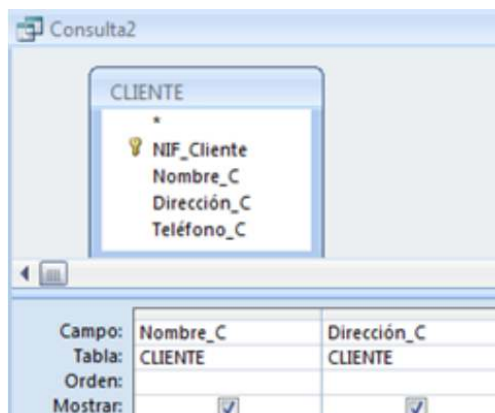
Y de ella elegir los campos: Nombres y Teléfonos.

Campo:	Nombre_M	Teléfono_M
Tabla:	MONTADOR	MONTADOR
Orden:		
Mostrar:	☒	☒
Criterios:		
O:		

Una vez realizada tal selección se grabará la consulta con el nombre de Consulta1 Y para mostrarla elegiremos la pestaña "Consultas" de la lista desplegable de "Tablas". Pinchando sobre "Consulta 1" se deberá abrir el resultado de la misma.

Nombre_M	Teléfono_M
Luis Angulo Boza	957654532
Mario Fernández Cacho	956781111
Javier González Pérez	957897676
Román Boza Núñez	957676767

- i) Muestra los nombres y direcciones de los clientes y graba dicha consulta con el nombre de "Consulta2".



Nombre_C	Dirección_C
Agustín Moreno Carvajal	Ronda, 13
Lucía Rosique Fernández	Alfonso XII, 25
Martín Romero Wic	Jalón, 2
Rafael López Pérez	Cervantes, 14
Patricia Mena Vicente	Sevilla, 23
Antonio Fernández Martínez	Sastres, 7

j) Muestra todas las cocinas montadas por el montador de NIF 87654329-F.

Consulta3

C-M

*
 Num_Ref
 NIF_M
 Número de cocinas

Campo: Num_Ref NIF_M
 Tabla: C-M C-M
 Orden: Ascendente Ascendente
 Mostrar: ☒ ☐
 Criterios: "87654329-F"

Num_Ref	
2	6
*	0

k) Muestra el NIF de los clientes que han adquirido la cocina de modelo 2 y la cocina de modelo 6.

Consulta4

C-Cli

*
 Num_Ref
 NIF_C

Campo: NIF_C Num_Ref
 Tabla: C-Cli C-Cli
 Orden: Ascendente
 Mostrar: ☒ ☒
 Criterios: 2 O 6

NIF_C	Num_Ref
89765432-R	2
54678963-L	2
52567898-S	6
89765432-R	6
54678963-L	6

l) Muestra los nombres de los clientes que han adquirido una cocina del modelo 2 o del modelo 6

Consulta5

CLIENTE

*
 NIF_Cliente
 Nombre_C
 Dirección_C

1

C-Cli

*
 Num_Ref
 NIF_C

Campo: Nombre_C Num_Ref
 Tabla: CLIENTE C-Cli
 Orden: Ascendente
 Mostrar: ☒ ☒
 Criterios: 2 O 6

Nombre_C	Num_Ref
Antonio Fernández Martínez	2
Lucía Rosique Fernández	2
Agustín Moreno Carvajal	6
Antonio Fernández Martínez	6
Lucía Rosique Fernández	6

PRÁCTICA 3

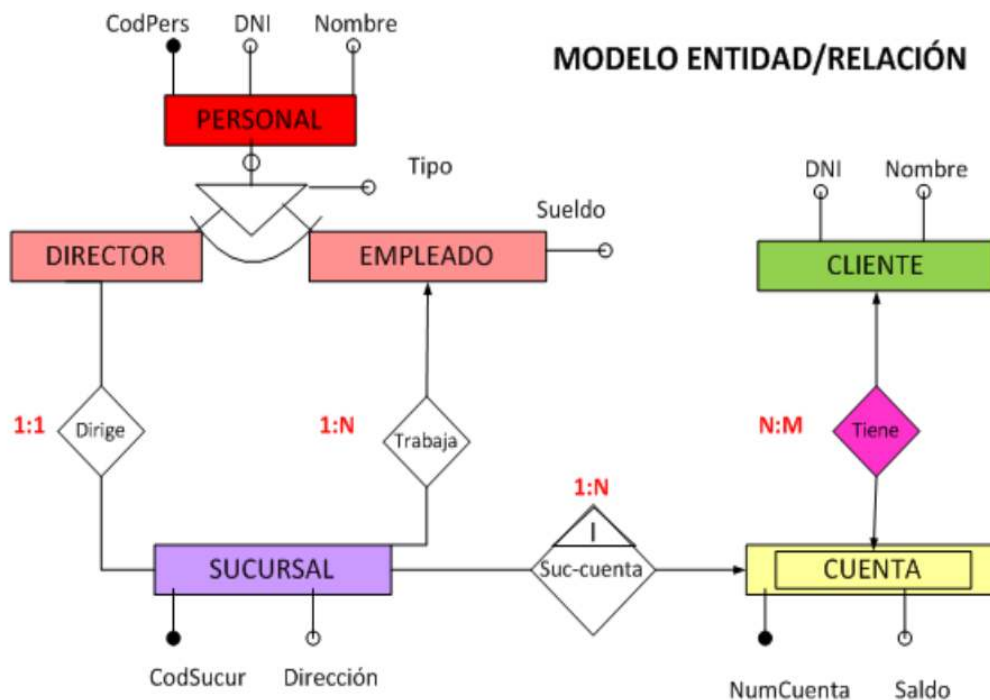
PLANTEAMIENTO

OBJETIVO: Adentrarse en el diseño e implementación de BD a través de un ejemplo no guiado.

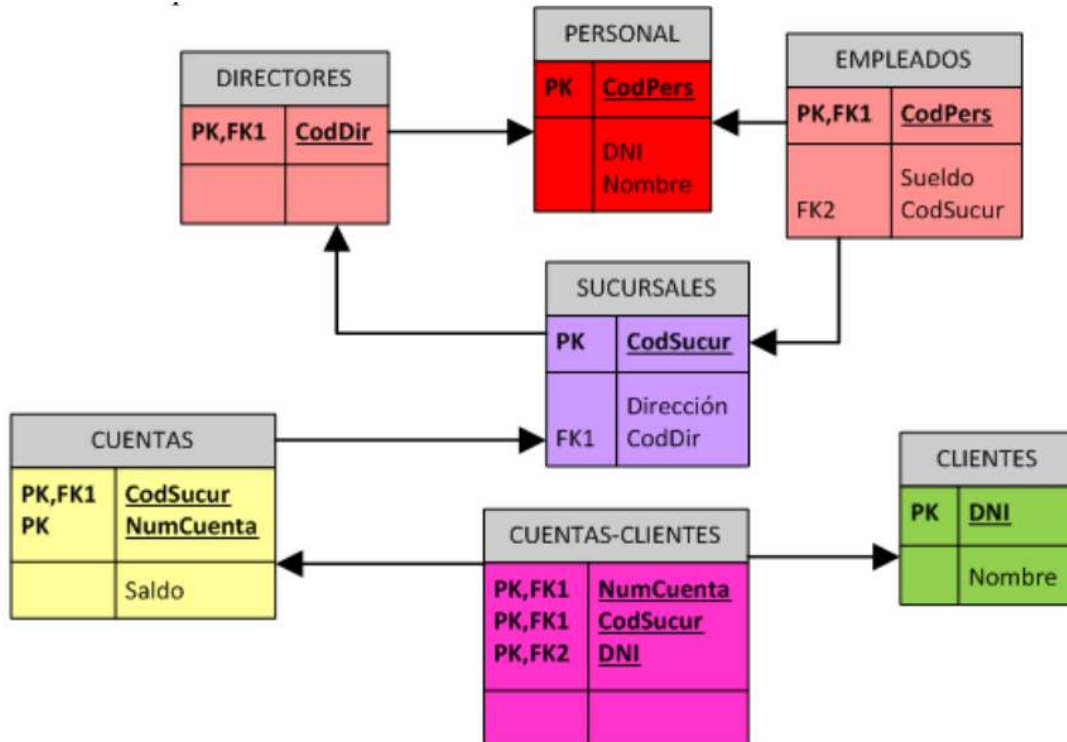
ENUNCIADO: Se plantea el enunciado de una base de datos y los esquemas conceptuales y lógicos que resultan y a partir de los cuales se deberán crear las tablas relacionadas en ACCESS. Un banco desea informatizar el seguimiento de las cuentas de ahorro que sus clientes tienen en sus sucursales así como la gestión del personal. Realiza la BD que represente este sistema y que cumpla las siguientes restricciones: Cada Cuenta de Ahorro tiene un número e interesa conocer el saldo de las mismas. Cada cliente puede tener 1 o varias cuentas en una o varias sucursales. El cliente se identifica por su DNI y guardamos también su nombre y dirección. De una cuenta de ahorro pueden ser titulares varios clientes. Cada sucursal se conoce por su número de sucursal y su dirección. El personal del banco se divide en: empleados de sucursal y directores de sucursal. De todos interesa conocer su código de empleado, DNI y su nombre.

Los empleados se encuentran destinados de forma exclusiva a una sucursal. De ellos interesa conocer además su sueldo. Cada sucursal tiene un único director de sucursal.

- Identifica las entidades y los datos que detectas en el enunciado anterior.
- Observa el siguiente **modelo Entidad/Relación** que se obtendría a partir del enunciado anterior. Detecta en él cuáles serían las entidades y los datos.



- Observa el siguiente **modelo Relacional** que se obtendrá a partir del modelo Entidad/Relación anterior. Indica en él:



- Cuáles son las tablas que deberás crear en ACCESS.
 - Cuáles son las claves principales.
 - Cuáles son las claves foráneas.
 - A qué tipo de Base de Datos y Sistema Gestor de Bases de Datos corresponde el diseño planteado.
- d. Utiliza el modelo Relacional para crear la BD correspondiente en ACCESS. Deberás tener en cuenta que los campos cumplirán las siguientes condiciones:
- Los códigos del personal empezarán por la letra P e irán seguidos de un número correlativo. Ejemplo: P1, P2, P3...
 - Los números de cuentas bancarios actuales están formados por 20 dígitos que tienen la estructura siguiente:
 - El código del banco al que pertenece la cuenta (4 dígitos).
 - El código de la sucursal en el que se abrió la cuenta (4 dígitos).
 - Un número de control, llamado dígito de control, que impide errores de teclado (2 dígitos).
 - Y por último, el número de cuenta (10 dígitos).
 - Como en nuestra BD todas las cuentas pertenecen a la misma entidad bancaria, el código de sucursal estará formado por 4 dígitos.
 - Los DNI incluirán la letra. Tendrá un tamaño de 9 caracteres.
 - El Saldo y el Sueldo serán campos numéricos. El sueldo siempre será un número mayor que 0, pero el saldo puede ser negativo (números rojos).
 - Los campos «Nombre» y «dirección» deben tener el tamaño adecuado para incluir los datos. Ponemos tamaño 50.

PRÁCTICA 4

PLANTEAMIENTO

OBJETIVO: Se trata de una práctica de ampliación. Con ella se pretende que el alumno utilice el manual “Biblia de ACCESS 2007”. Para ello se proponen una serie de prácticas enumeradas del 1 al 12 para cuya realización se deberá buscar la información del manual. Es deseable que el alumno adquiera autonomía en el manejo de documentación y por eso habrá que intentar consultar al profesor sólo cuando sea estrictamente necesario. Todo lo que se requiere para solucionar la práctica está en el manual que podéis consultar en la plataforma Moodle del curso GBD.

ENUNCIADO: Ejercicio de Microsoft ACCESS.

I. Creación de una Base de Datos y diseño de varias tablas.

1. Crear una nueva Base de Datos. Lllamarla CURSACC01.MDB
2. Crear una tabla nueva para registrar la información de fichas de Clientes. Lllamarla CLIENTES. Estará compuesta por los siguientes campos:

<i>Nombre del campo</i>	<i>Tipo de datos</i>	<i>Tamaño</i>	<i>Propiedades</i>
CODCLIENTE	Numérico	Entero largo	Título: CÓDIGO CLIENTE
NOMBRECLI	Texto	25	Título: NOMBRE CLIENTE
DIRECCION	Texto	50	
CODPOSTAL	Texto	5	Poner una Máscara de entrada Título: CÓDIGO POSTAL
POBLACION	Texto	25	Valor predeterminado: Barcelona
TELEFONO	Texto	11	
FAX	Texto	11	
DESCUENTO	Numérico	Simple	Formato porcentual con 2 decimales Regla validación: <0,25
ZONAVENTAS	Numérico	Byte	Título ZONA DE VENTAS Requerido

3. Asignar como Clave Principal el campo CODCLIENTE.
4. Crear una tabla nueva para registrar la información de fichas de Artículos. Lllamarla ARTICULOS. Compuesta por los siguientes campos:

<i>Nombre del campo</i>	<i>Tipo de datos</i>	<i>Tamaño</i>	<i>Propiedades</i>
CODARTIC	Numérico	Entero largo	Título: CÓDIGO ARTÍCULO
DESCRIPCION	Texto	30	
PVP	Numérico	Simple	Formato Estándar con 2 decimales

5. Asignar como Clave Principal el campo CODARTIC.
6. Crear una Tabla nueva para registrar la información de Pedidos. Lllamarla PEDIDOS. Compuesta por los siguientes campos:

<i>Nombre del campo</i>	<i>Tipo de datos</i>	<i>Tamaño</i>	<i>Propiedades</i>
NUMPEDIDO	Autonumérico	Entero largo	Título NUMERO PEDIDO
CODCLIENTE	Númérico	Entero largo	
CODARTIC	Númérico	Entero largo	
UNIDADES	Númérico	Simple	Formato Estándar con 0 decimales
FECHAPED	Fecha		Formato Fecha Corta

- Asignar como Clave principal el campo NUMPEDIDO.
- Crear una Tabla nueva para registrar la información de las zonas de Ventas. Llamarla ZONAS. Compuesta por los siguientes campos:

<i>Nombre del campo</i>	<i>Tipo de datos</i>	<i>Tamaño</i>	<i>Propiedades</i>
ZONA	Númérico	Byte	
DESCRIPCION	Texto	25	Título NOMBRE DE ZONA

- Asignar como Clave principal el campo ZONA.

II. Adquirir práctica en el uso de las hojas de datos.

- Abra la tabla ARTICULOS, y cumplimente 6 ó 7 registros. Para los precios indique diversas cantidades entre 100 y 500 -esto será útil para algunos de los ejercicios posteriores-.

Consejo: Invéntese los datos, pero utilice números consecutivos para el campo CODARTIC, para facilitar más adelante la introducción de datos en PEDIDOS.

- Abrir la tabla ZONAS, y cumplimentar 4 registros.

Consejo: Zonas: Norte, Sur, Este y Oeste

- Abra la tabla CLIENTES y cumplimente al menos 10 registros. No es preciso cumplimentar todos los campos, pero necesariamente debe rellenar CODCLIENTE, NOMBRECLI, CODPOSTAL, POBLACION, DESCUENTO y ZONAVENTAS, pues utilizaremos estos datos más adelante. En el campo ZONAVENTAS utilice exclusivamente datos que haya insertado en el campo ZONA de la tabla ZONAS. Varios clientes deberán ser de Barcelona y Madrid.

Consejo: Invéntese los datos, pero utilice números consecutivos para el campo CODCLIENTE, para facilitar más adelante la introducción de datos en PEDIDOS.

Observe como las propiedades de campos que hemos definido, determinan el comportamiento de Access al introducir códigos postales (máscara) y descuentos (regla de validación); observe también que ocurre cuando intenta omitir ZONAVENTAS en algún registro (requerido). Observe como los nombres de los campos difieren de los de las columnas en aquellos campos para los cuales se ha definido la propiedad título.

- Con la tabla CLIENTES practique los siguientes puntos:
 - Redimensione el tamaño de las columnas a las necesidades de su contenido.
 - Ordene todos sus registros en base a los datos de la columna NOMBRECLI.

- Observe el efecto de la ordenación en las otras columnas. Pruebe con otras columnas.
- 5. Pida que Access busque un dato cualquiera dentro de su tabla.
- 6. Cree y aplique un filtro cada vez, capaz de:
 - Mostrar solo clientes de la Zona de Ventas 1
 - Mostrar solo clientes de la Población de Barcelona
 - Mostrar solo clientes de Barcelona y con un Descuento superior al 5 %
- 7. Mueva la columna TELEFONO a la derecha de la columna NOMBRECLI. Pruebe otros movimientos.
- 8. Oculte las columnas DESCUENTO y ZONAVENTAS. Vuelva a mostrarlas. Pruebe otras.
- 9. Cree un nuevo registro con la particularidad que el contenido del campo CODCLIENTE ya exista en otro de sus registros. Observe la reacción del sistema (bloqueo frente la violación de la clave principal.)

III. Establecer relaciones entre tablas y experimentar con la integridad referencial.

- 1) Abrir la ventana de Relaciones.
- 2) Agregar las tablas CLIENTES, ARTICULOS, PEDIDOS y zonas para crear las relaciones entre las mismas.
- 3) Crear las siguientes relaciones entre las tablas correspondientes:
 - Todas las relaciones se crearán exigiendo integridad referencial, con actualización y eliminación en cascada.
 - Guardar el diseño de la relación.
- 4) Abrir la tabla PEDIDOS y llenar entre 15 y 20 registros. Recordar que debido a la relación establecida con integridad referencial en los campos CODCLIENTE y CODARTIC solo se admitirán aquellos códigos existentes de la tabla CLIENTES y ARTICULOS respectivamente.
- 5) Probar de entrar algún CODCLIENTE o CODARTIC inexistente en la tabla PEDIDOS y observar el resultado.
- 6) Observar 2 ó mas registros en PEDIDOS con el mismo código de cliente (si no tiene registros que cumplan esta característica, créelos); estos dos registros se modificarán y se borrarán en el próximo ejercicio.
- 7) Cerrar la tabla PEDIDOS.
- 8) Abrir la tabla CLIENTES, localizar el registro correspondiente al cliente que hemos observado en el punto 6 y modificar el CODCLIENTE por otro Código no existente.
- 9) Cerrar la tabla CLIENTES.
- 10) Abrir la tabla PEDIDOS y comprobar como los dos pedidos introducidos en el ejercicio 6 han modificado su Código de Cliente. De manera análoga estos cambios podrían haber sido realizados en la tabla ARTICULOS.
- 11) Cerrar la tabla PEDIDOS.
- 12) Abrir la tabla CLIENTES, localizar el registro del cual se ha cambiado el Código y borrarlo.
- 13) Cerrar la tabla CLIENTES.
- 14) Abrir la tabla PEDIDOS y comprobar que los registros relacionados de esta tabla con el mismo código de cliente eliminado, se han borrado también de la tabla.
- 15) Cerrar la tabla PEDIDOS.

IV. Adquirir práctica en el diseño de consultas de selección

- 1) Diseñar una consulta que sea capaz de devolver todos los clientes que pertenezcan a la ZONA DE VENTAS número 1.
- 2) Además de el criterio anterior, esta consulta solamente deberá mostrarnos los campos CODCLIENTE y NOMBRECLIENTE, sin mostrar el campo de ZONA DE VENTAS.
- 3) Modificar la consulta para que nos muestre también los registros de la ZONA DE VENTAS número 3.
- 4) Guardar la consulta con el nombre SELECCIÓN ZONA DE VENTAS.
- 5) Modificar la consulta para hacer que cada vez que ejecutemos la consulta nos solicite la ZONA DE VENTAS que deseamos ver. (Parámetros).
- 6) Probar su funcionamiento con diferentes Zonas de ventas.
- 7) Modificar la consulta para que en vez de solicitar el código de la zona de ventas, nos solicite el nombre de la zona (campo DESCRIPCION).
- 8) Modificar la consulta para que aparezcan solamente aquellos registros de la tabla CLIENTES con las condiciones actuales de la consulta pero además solamente deberán salir aquellos que hayan realizado alguna venta.

Consejo: Para comprobarlo asegúrese de crear algún cliente nuevo en una zona. Este cliente no debería aparecer en el resultado de la consulta, pues no tiene pedidos.

- 9) Guardar la consulta.
- 10) Crear una nueva consulta basada en la tabla CLIENTES en la cual aparezcan los campos: NOMBRECLIENTE, CODPOSTAL y POBLACIÓN, debiendo aparecer solamente los registros que pertenezcan a la POBLACIÓN de Barcelona.
- 11) Guardar la consulta con el nombre CLIENTES DE BARCELONA.
- 12) Ejecutar la consulta.
- 13) Guardar la consulta con el nombre AÑADIR REGISTROS.
- 14) Abrir la tabla NUEVA TABLA PEDIDOS y comprobar que se han agregado los registros.
- 15) Cerrar la tabla NUEVA TABLA PEDIDOS .
- 16) Ejecutar otra vez la consulta y observar el mensaje que nos presenta Access.

U. Adquirir práctica en el diseño de Formularios combinados con consultas

- 1) Crea una nueva consulta de selección en la que aparezcan los campos: NOMBRECLIENTE, ARTICULO.DESCRIPCION, UNIDADES, PVP, DESCUENTO.
 - Añadir un campo calculado llamado NETO, que será el resultado de: $\text{Unidades} * \text{PVP} * (1 - \text{DESCUENTO})$. Aplicar a este campo calculado el formato Euro con 2 decimales.
 - Grabar la consulta con el nombre DESCUENTO.
- 2) Crea un formulario para la consulta que hemos creado en el punto anterior.
 - El formulario deberá ser de Tipo Tabular y con todos los campos de la consulta.
 - Grabar el formulario con el nombre DESCUENTO.
- 3) Crea un informe para la consulta DESCUENTO.

- El informe será de tipo tabular con todos los campos de la consulta y deberá estar ordenado por NOMBRECLI.
 - Grabar el informe con el nombre DESCUENTO.
- 4) Crea una consulta de selección en la que aparezcan los siguientes campos: NOMBRECLI, COD-POSTAL, POBLACION, DESCUENTO.
- Esta consulta deberá preguntarme el nombre del cliente que quiero visualizar cada vez que la ejecute (parámetros).
 - Grabar esta consulta con el nombre PARÁMETROS DESCUENTO.

Consejo: Usar en el criterio el operador “Como” para que se puedan utilizar comodines al introducir el nombre.

VI. Adquirir práctica en el diseño de Formularios

- 1) Crea una consulta de selección que nos presente de la tabla Artículos todos sus campos de aquellos que el artículo sea el 00001.
- 2) Guarda la consulta con el nombre IDENTIFICACIÓN DEL ARTICULO.
- 3) Crea un formulario de tipo simple para la consulta anterior.
- 4) Modifica el aspecto del título del formulario añadiendo colores, bordes y cambiando el tipo de letra.
- 5) Añade 2 registros a la tabla Artículos a través del formulario.
- 6) Guarda el formulario con el nombre IDENTIFICACIÓN DEL ARTICULO.
- 7) Comprueba que los registros que has añadido se encuentran en la tabla.

VII. Adquirir práctica en el diseño de consultas

- 1) Abre la base de datos, en la cual están contenidas las tablas de CLIENTES, ARTÍCULOS y PEDIDOS.
- 2) Crea una consulta de selección en que aparezcan: CODCLIENTE, NOMBRE, UNIDADES.
- 3) Modifica la consulta anterior para que aparezca también el campo PVP , y me muestre solamente aquellos que las unidades son mayores a 70.
- 4) Graba esta consulta con el nombre CONSULTA_SEL_1.
- 5) Crea una consulta de CREACIÓN DE NUEVA TABLA en la que se creen los campos, CODARTIC, CODCLIENTE y DESCRIPCION, pero solamente aquellos que la descripción este entre las Letras A y F.
- 6) Llamar a la nueva tabla DESCRIPCION y guardar la consulta con el nombre DESCRIP_NUEVA.
- 7) Ver el contenido de la nueva tabla creada.
- 8) Crea una consulta del tipo DATOS AÑADIDOS, en la que aparezcan los campos CODARTIC, CODCLIENTE y DESCRIPCION y me agregue a la tabla con nombre DESCRIPCION aquellos registros que el PVP este entre 70 y 100.
- 9) Guardarla con el nombre AÑADIR.
- 10) Comprobar en la tabla DESCRIPCION el resultado de la consulta.

DISEÑO DE MODELOS LÓGICOS NORMALIZADOS

2.1 INTRODUCCIÓN

En este tema veremos como hacer el diseño conceptual y lógico de una base de datos. Empezaremos elaborando el modelo conceptual usando diagramas Entidad-Relación y Entidad-Relación extendidos. Este diseño es de más alto nivel, más próximo al usuario y más alejado del diseño físico de la BD. A continuación, a partir del modelo Entidad-Relación, procederemos a generar el modelo relacional, el cual ya se halla muy próximo al modelo físico de BD. Veremos las reglas de transformación que hemos de seguir para ello. Por último deberemos normalizar las tablas obtenidas para evitar redundancias. Resumiendo, los 2 modelos lógicos, de mayor a menor nivel de abstracción, que veremos en este tema son:

- **Modelo Entidad-Relación (extendido)**
- **Modelo Relacional**

En el siguiente tema, realizaremos el diseño físico de la BD a partir del modelo relacional.

2.2 DISEÑO DE BD

El diseño de una base de datos consiste en extraer todos los datos relevantes de un problema, por ejemplo, saber que datos están implicados en el proceso de facturación de una empresa que vende artículos de informática, o, que datos son necesarios para llevar el control de pruebas diagnósticas en un centro de radiológico. Para extraer estos datos, se debe realizar un análisis en profundidad del problema, para averiguar qué datos son esenciales para la base de datos y descartar los que no sean necesarios. Una vez extraídos los datos esenciales comenzamos a construir los modelos adecuados. Es decir, construimos, mediante una herramienta de diseño de base de datos, un esquema que exprese con total exactitud todos los datos que el problema requiere almacenar. Ya dijimos en el tema anterior, que es algo equivalente al dibujo de un plano previo a la construcción de un edificio. También introducimos en el tema 1, las distintas fases por las que atraviesa el proceso de diseño de una Base de Datos. Además, previo al diseño es necesario realizar una primera fase denominada de análisis.

2.2.1 Fase de Análisis: Especificación de requisitos Software (E.R.S.)

Antes de pasar a diseñar una BD hay que tener claro que es lo que queremos hacer. Para ello, típicamente los informáticos se reúnen con los futuros usuarios del sistema para recopilar la información que necesitan para saber que desean dichos usuarios. Normalmente se hace una reunión inicial a y partir de ella se elabora una batería de preguntas para entrevistar a los usuarios finales en una segunda reunión y obtener de ella una información detallada de lo que se espera de nuestra BD. De estas entrevistas, se extrae el documento más importante del análisis, el documento de Especificación de Requisitos Software o E.R.S. A partir de dicha E.R.S. Se extrae toda la información necesaria para la modelización de datos.

2.2.2 Fase 1 del diseño. Diseño Conceptual: Modelo Entidad/Relación (E/R)

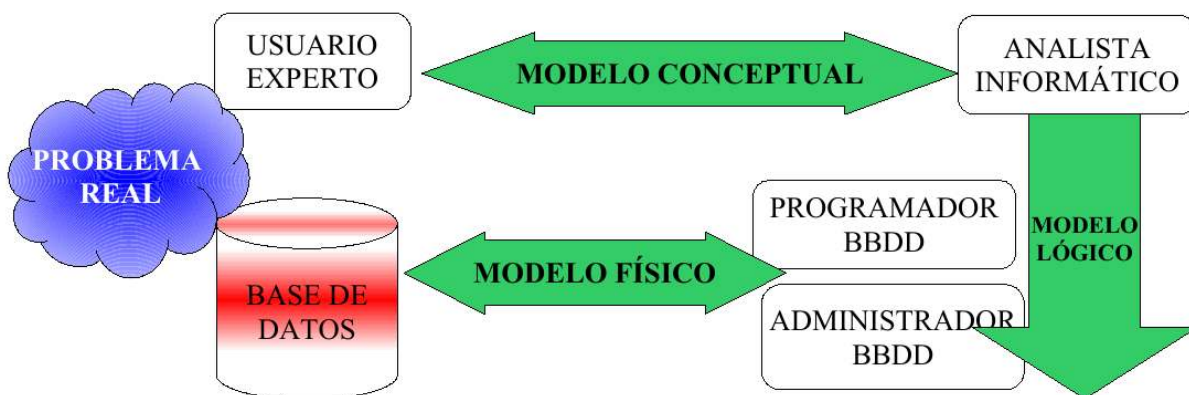
Habitualmente quien realiza la modelización es un analista informático que no tiene porqué ser un experto en el problema que pretende resolver (Contabilidad, Gestión de Reservas hoteleras, medicina, economía, etc.). Es por esto que es imprescindible contar con la experiencia de un futuro usuario de la BD que conozca a fondo todos los entresijos del negocio, y que, a su vez, no tienen porqué tener ningún conocimiento de informática. El objetivo de esta fase del diseño consiste en representar la información obtenida del usuario final y concretada en el E.R.S. mediante estándares para que el resto de la comunidad informática pueda entender y comprender el modelo realizado. El modelo que se utiliza en esta primera fase del diseño tiene un gran poder expresivo para poder comunicarse con el usuario que no es experto en informática y se denomina Modelo Conceptual. El modelo conceptual que utilizaremos es el Modelo Entidad/Relación e iremos profundizando en él a lo largo de esta unidad.

2.2.3 Fase 2 del diseño. Diseño Lógico: Modelo Relacional

Este modelo es más técnico que el anterior porque está orientado al personal informático y generalmente tiene traducción directa al modelo físico que entiende el SGBD. Se obtienen a partir del modelo conceptual y dependerá de la implementación de la BD. Así, no es lo mismo implementar una base de datos jerárquica u orientada a objetos que una BD relacional. El modelo que se usará en este módulo es el Modelo Relacional.

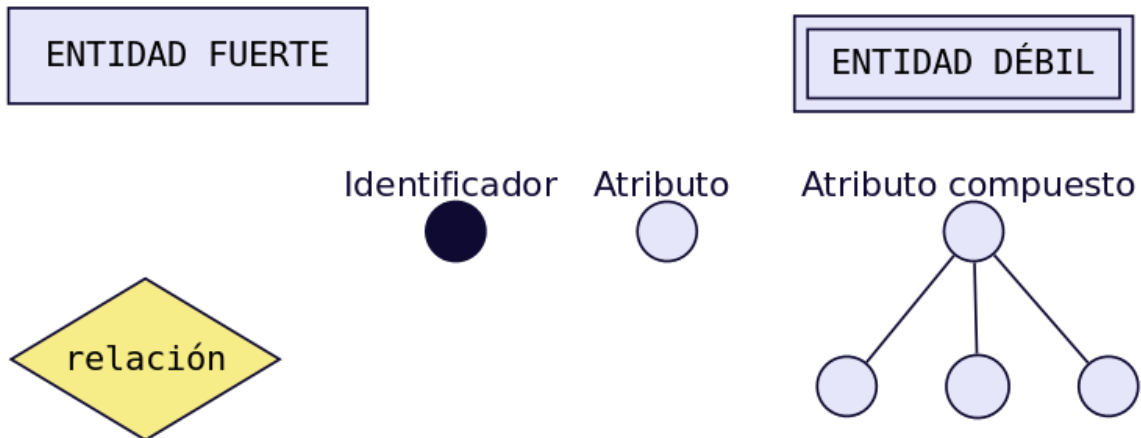
2.2.4 Fase 3 del diseño. Diseño Físico: Modelo Físico

Es el resultado de aplicar el modelo lógico a un SGBD concreto. Generalmente está expresado en un lenguaje de programación de BBDD tipo SQL. En este módulo, transformaremos el Modelo Relacional en el modelo físico mediante el sublenguaje DDL de SQL. Esto se estudiará en el próximo tema.



2.3 MODELO ENTIDAD/RELACIÓN

El modelo Entidad-Relación es el modelo más utilizado para el diseño conceptual de bases de datos. Fue introducido por Peter Chen en 1976 y se basa en la existencia de objetos a los que se les da el nombre de entidades, y asociaciones entre ellos, llamadas relaciones. Sus símbolos principales se representan en el cuadro siguiente.



A continuación se detallan los elementos fundamentales de este modelo.

2.3.1 Entidades

Una entidad es cualquier objeto o elemento acerca del cual se pueda almacenar información en la BD. Las entidades pueden ser concretas como una persona o abstractas como una fecha. Las entidades se representan gráficamente mediante rectángulos y su nombre aparece en el interior. Un nombre de entidad sólo puede aparecer una vez en el esquema conceptual.

Tipos de entidades

Hay dos tipos de entidades: fuertes y débiles. Una entidad débil es una entidad cuya existencia depende de la existencia de otra entidad. Una entidad fuerte es una entidad que no es débil.



■



■

■



■

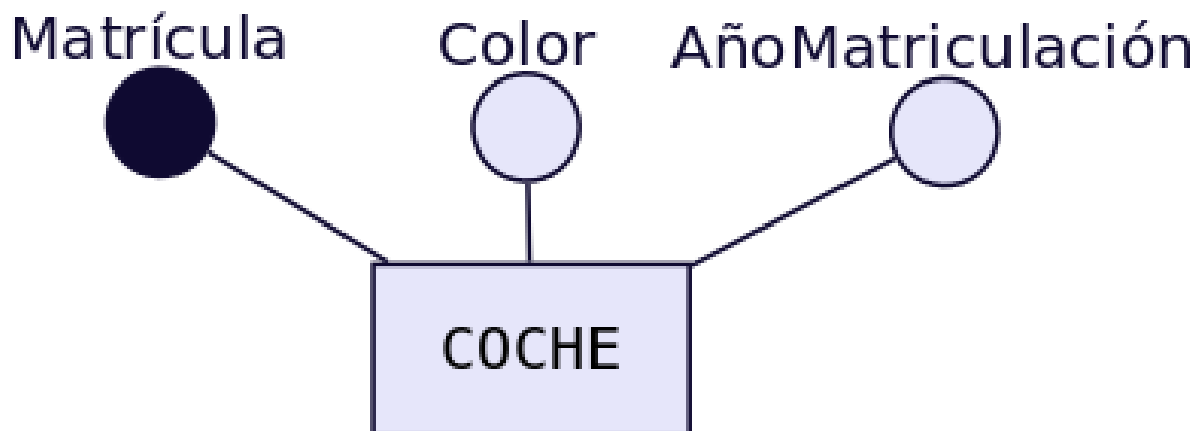
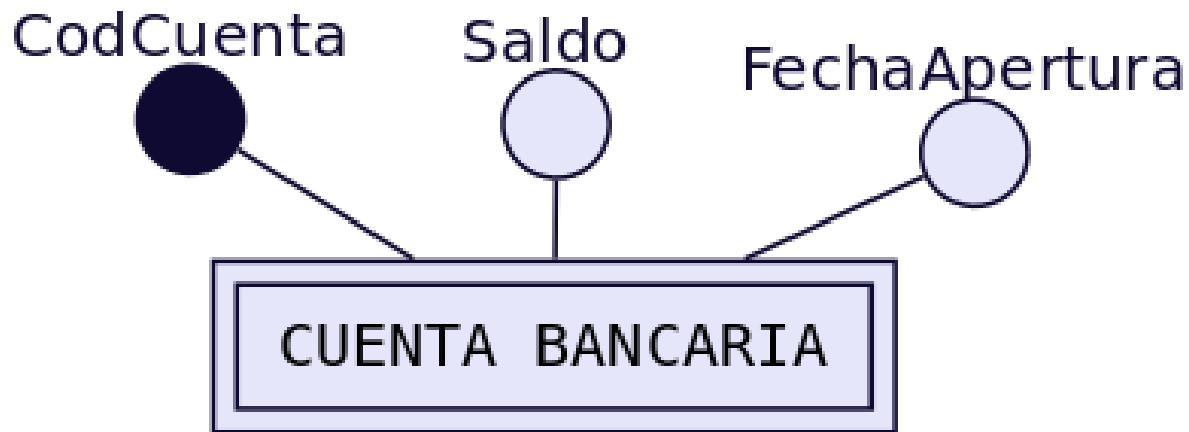
2.3.2 Atributos

Una entidad se caracteriza y distingue de otra por los atributos, en ocasiones llamadas propiedades o campos, que representan las características de una entidad. Los atributos de una entidad pueden tomar un conjunto de valores permitidos al que se le conoce como dominio del atributo. Dando valores a estos atributos, se obtienen las diferentes ocurrencias de una entidad.

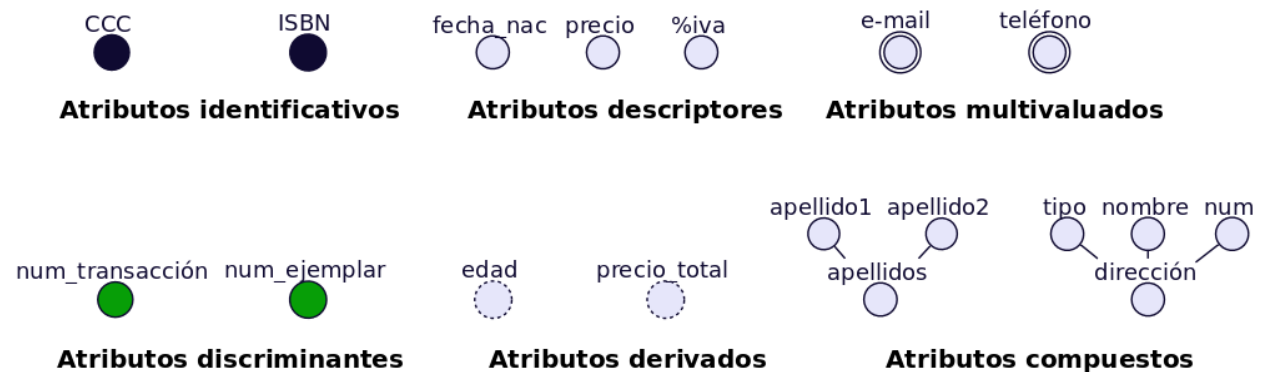
En esencia, existen dos tipos de atributos: - Identificadores de entidad (también llamados clave primaria o clave principal): son atributos que identifican de manera unívoca cada ocurrencia de una entidad. - Descriptores de entidad: son atributos que muestran unas características de la entidad.

Siempre debe existir, al menos, un atributo identificativo.

Ejemplos de atributos:



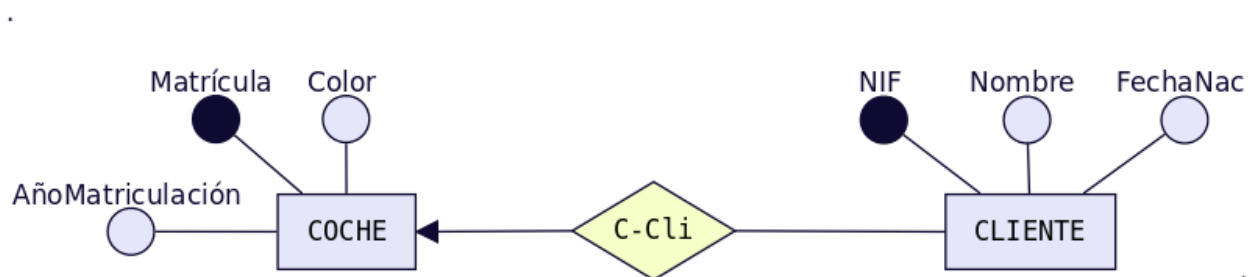
Tipos de atributos



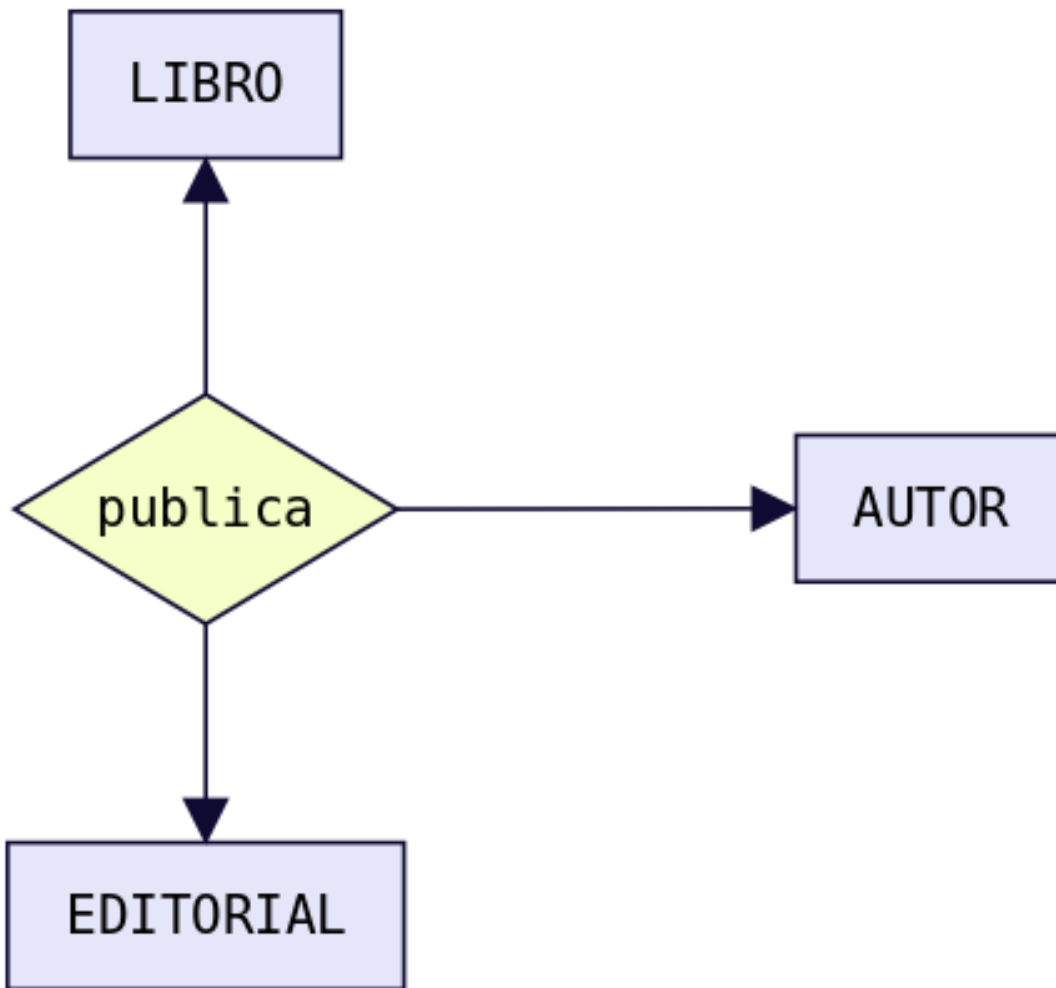
- **Atributos identificadores o identificativos:** Son atributos cuyos valores no se repiten dentro de una misma entidad o relación. Sirven para identificar de forma unívoca cada ocurrencia. Actúan como clave principal o primaria. Por ejemplo CCC (Código Cuenta Corriente) que identifica cada cuenta bancaria. O ISBN (International Standard Book Number) que identifica cada libro que se publica. Un atributo identificativo puede ser un atributo compuesto. Por ejemplo CCC podría descomponerse en 3 atributos: num_banco, num_sucursal y num_cuenta.
- **Atributos discriminadores o discriminantes:** Son atributos que discriminan distintas ocurrencias de una entidad débil en identificación dentro de la entidad fuerte de la que dependen. Lo representaremos con un círculo relleno de un color distinto a los atributos identificadores y descriptivos. Por ejemplo num_transacción dentro de una CCC o num_ejemplar dentro de un ISBN.
- **Atributos descriptores o descriptivos:** Son los atributos que describen diversas propiedades de una entidad o relación (¡la relaciones también pueden tener atributos!). Son los más frecuentes.
- **Atributos derivados:** Son atributos cuyos valores se calculan a partir de los valores de otros atributos. Por ejemplo podemos disponer de un atributo fecha_nac que sería un atributo descriptivo normal y calcular el valor del atributo edad a partir de él. El precio_total también podría calcularse a partir del precio + %iva.
- **Atributos multivaluados:** Son atributos descriptores que poseen varios valores de un mismo dominio. Por ejemplo, si necesitamos almacenar varios e-mail de una misma persona entonces deberemos utilizar un atributo multivaluado. Igual sucede con el teléfono. Si sólo necesitamos almacenar un sólo valor utilizaremos un atributo descriptivo normal.
- **Atributos compuestos:** Muchas veces se confunden con los anteriores, aunque no tienen nada que ver con ellos. Un atributo compuesto es un atributo que puede ser descompuesto en otros atributos pertenecientes a distintos dominios.

2.3.3 Relaciones

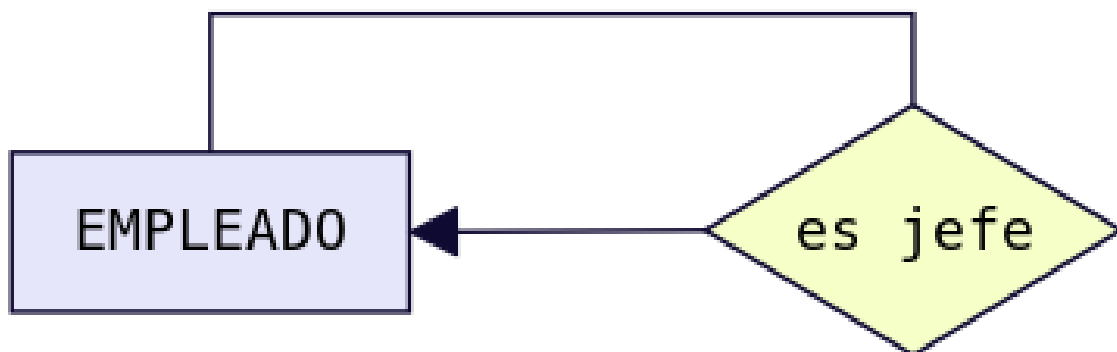
Una relación es la asociación que existe entre dos a más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior. Normalmente le pondremos de nombre la primera o primeras letras de las entidades que relaciona. Las entidades que están involucradas en una determinada relación se denominan entidades participantes. El número de participantes en una relación es lo que se denomina grado de la relación. Por ejemplo la relación CLIENTE-COCHE es de grado 2 o binaria, ya que intervienen dos entidades.



Observa que el nombre que ponemos a la relación usa las primeras letras de cada entidad. En este caso como ambas empiezan por «C» se añade algunas letras más para hacer referencia a CLIENTES. También podríamos haber puesto como nombre de la relación uno más descriptivo de la misma, por ejemplo «Compra» (CLIENTE compra COCHE), pero esta nomenclatura puede conducir a confusión a la hora de determinar la cardinalidad de la relación cuando estamos aprendiendo. La relación PUBLICAR, es de grado 3, ya que involucra las entidades LIBRO, EDITORIAL y AUTOR.



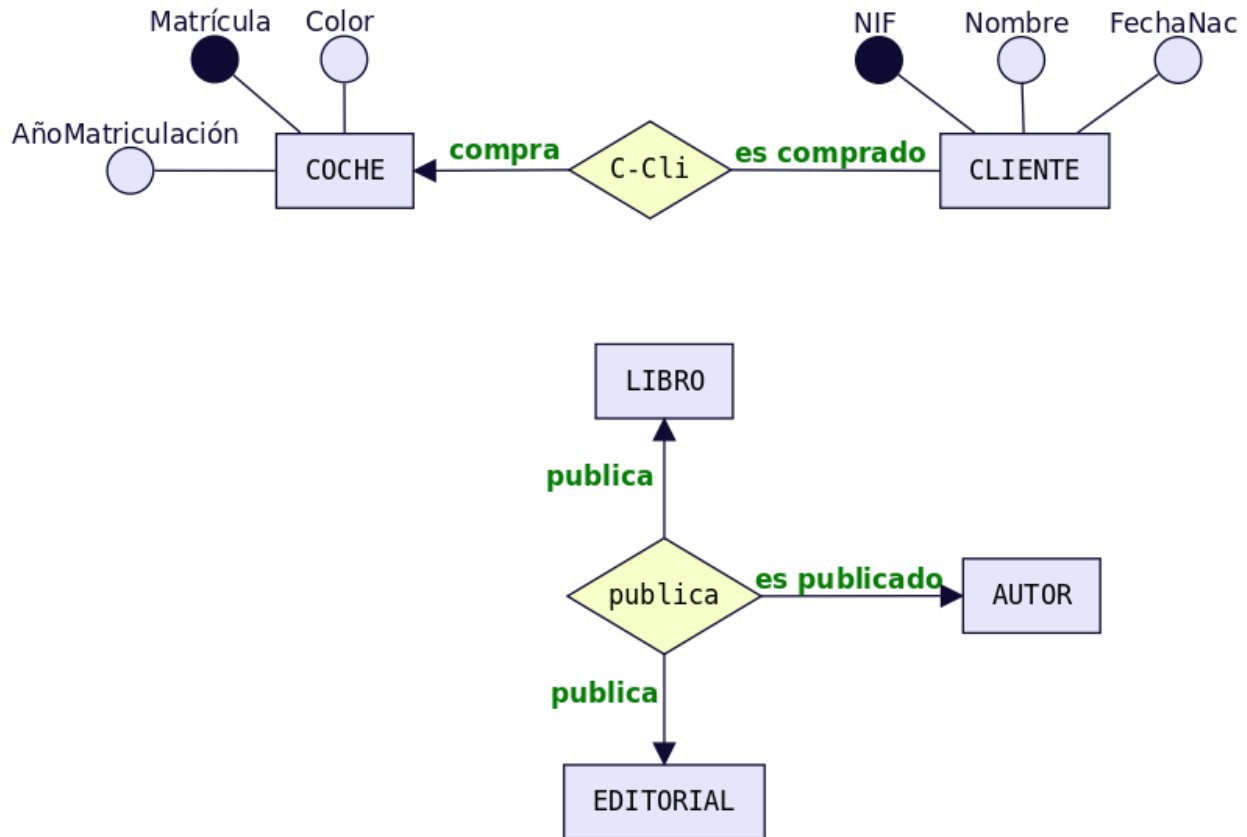
Cuando una entidad está relacionada consigo misma, hablamos de relación reflexiva.



Aunque el modelo E-R permite relaciones de cualquier grado, la mayoría de las aplicaciones del modelo sólo consideran relaciones del grado 2.

El Papel o Rol de una entidad en una relación

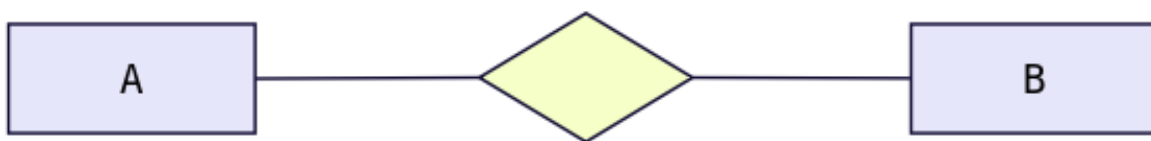
Es la función que tiene en una relación. Se especifican los papeles o roles cuando se quiera aclarar el significado de una entidad en una relación. A continuación mostramos los mismos ejemplos del punto anterior pero incluyendo el papel o rol de cada entidad en las relaciones:



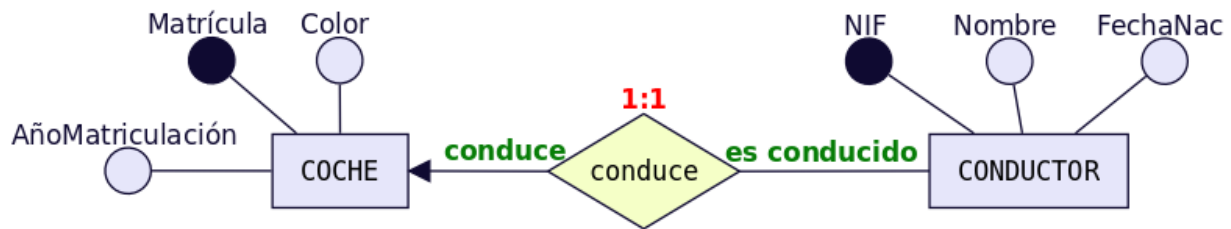
La Cardinalidad de una relación

Cuando la relación es binaria, cosa que ocurre en la mayoría de los casos, la cardinalidad es el número de ocurrencias de una entidad asociadas a una ocurrencia de la otra entidad. Existen principalmente tres tipos de cardinalidades binarias:

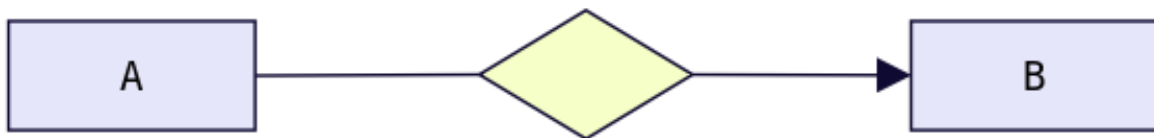
Relación uno a uno 1:1 A cada elemento de la primera entidad le corresponde no más de un elemento de la segunda entidad, y a la inversa. Es representado gráficamente de la siguiente manera:



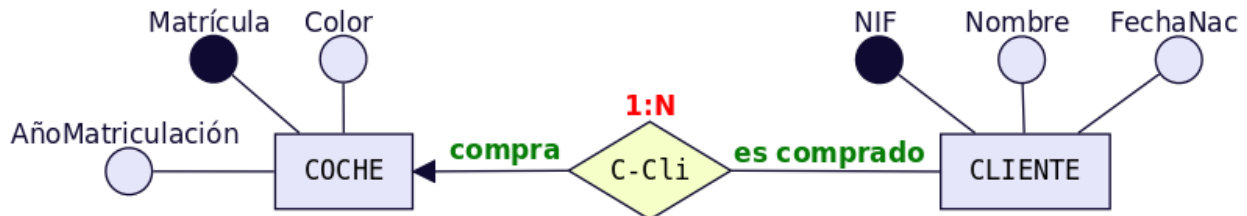
Ejemplo cardinalidad 1:1



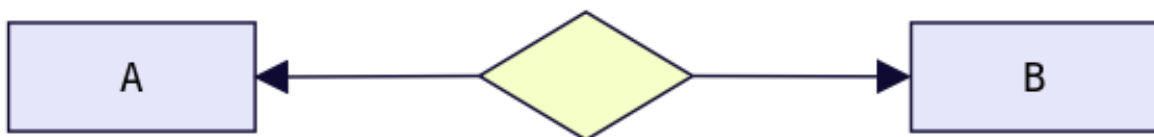
Relación uno a muchos 1:N Significa que cada elemento de una entidad del tipo A puede relacionarse con cualquier cantidad de elementos de una entidad del tipo B, y un elemento de una entidad del tipo B solo puede estar relacionado con un elemento de una entidad del tipo A. Su representación gráfica es la siguiente: Nótese en este caso que el extremo punteado de la flecha de la relación de A y B, indica un elemento de A conectado a muchos de B.



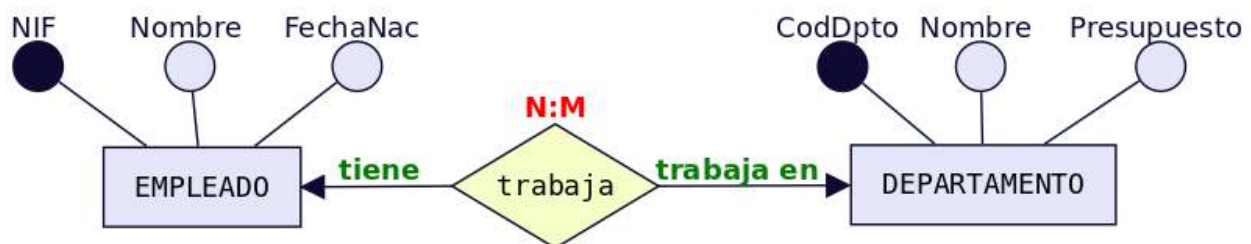
Ejemplo cardinalidad 1:N



Muchos a muchos N:M Establece que cualquier cantidad de elementos de una entidad del tipo A pueden estar relacionados con cualquier cantidad de elementos de una entidad del tipo B. El extremo de la flecha que se encuentra punteada indica el «varios» de la relación.



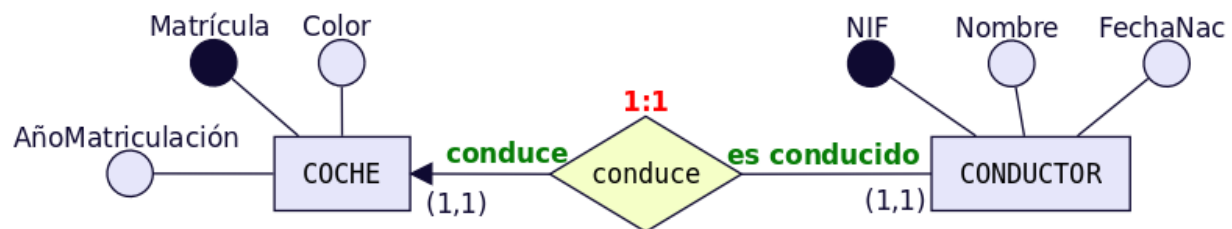
Ejemplo cardinalidad N:M



La Participación de una entidad

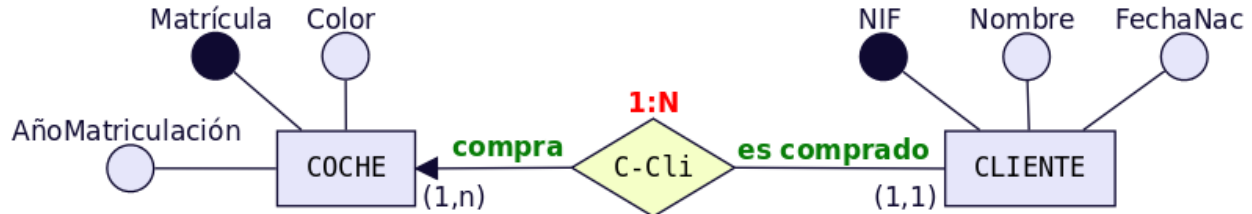
La participación de una entidad también se conoce como cardinalidad de la entidad dentro de una relación. Una misma entidad puede tener distinta cardinalidad dentro de distintas relaciones. Para obtener la participación, se debe fijar una ocurrencia concreta de una entidad y averiguar cuántas ocurrencias de la otra entidad le corresponden como mínimo y como máximo. Después realizar lo mismo en el otro sentido. Estas ocurrencias mínimas y máximas (llamadas también participación de una entidad) se representarán entre paréntesis y con letras minúsculas en el lado de la relación opuesto a la entidad cuyas ocurrencias se fijan. Para determinar la cardinalidad nos quedamos con las participaciones máximas de ambas y se representan con letras mayúsculas separadas por dos puntos junto al símbolo de la relación. Veamos algunos ejemplos:

Ejemplo 1



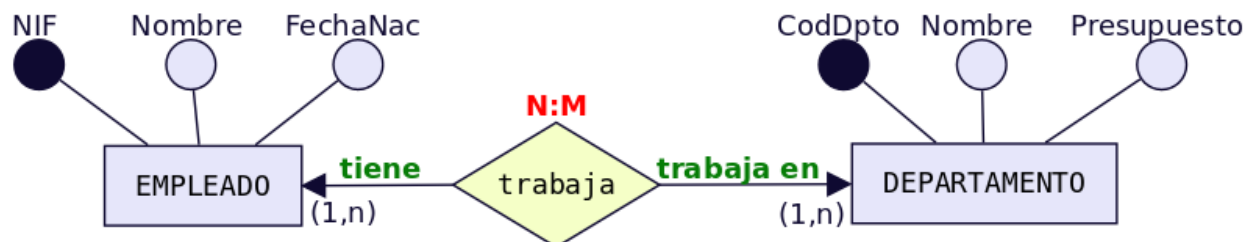
Un conductor «conduce» como mínimo 1 coche y como máximo 1 coche → Participación (1,1) y se pone en el lado opuesto a CONDUCTOR, es decir, junto a COCHE. Un coche «es conducido» como mínimo por 1 conductor y como máximo por 1 conductor → Participación (1,1) y se pone en el lado opuesto a COCHE, es decir, junto a CONDUCTOR. Para determinar la cardinalidad nos quedamos con las dos participaciones máximas. Es decir → 1:1.

Ejemplo 2



Un cliente «compra» como mínimo 1 coche y como máximo puede comprar más de un coche, es decir, varios coches. Ese varios se representa con la letra «n» → Participación (1,n) y se pone en el lado opuesto a CLIENTE, es decir, junto a COCHE. Un coche «es comprado» como mínimo por 1 cliente y como máximo por 1 cliente → Participación (1,1) y se pone en el lado opuesto a COCHE, es decir, junto a CLIENTE. Para determinar la cardinalidad nos quedamos con las dos participaciones máximas y la «n» se pone en mayúsculas «N». Es decir → 1:N.

Ejemplo 3



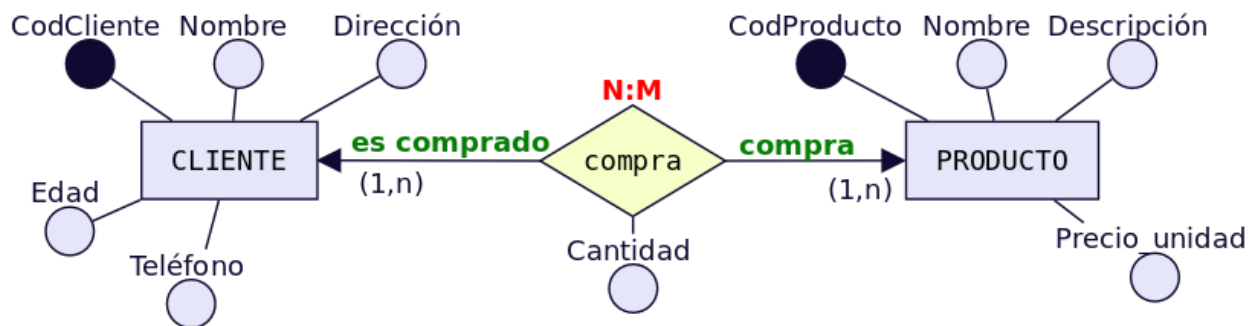
Un empleado «trabaja» como mínimo 1 departamento y como máximo puede trabajar en varios. Ese varios se representa con la letra «n» → Participación(1,n) y se pone en el lado opuesto a EMPLEADO, es decir, junto a DEPARTAMENTO. Un departamento «tiene» como mínimo por 1 empleado y como máximo puede tener varios → Participación

(1,n) y se pone en el lado opuesto a DEPARTAMENTO, es decir, junto a EMPLEADO. Para determinar la cardinalidad nos quedamos con las dos participaciones máximas y la «n» se pone en mayúsculas «N» y para diferenciar el otro «varios» en lugar de «N» ponemos «M» (Igual que cuando en matemáticas había dos variables no se ponía x e x sino x e y). Es decir $\rightarrow N:M$.

Atributos propios de una relación

Las relaciones también pueden tener atributos, se les denominan atributos propios. Son aquellos atributos cuyo valor sólo se puede obtener en la relación, puesto que dependen de todas las entidades que participan en la relación. Veamos un ejemplo.

Ejemplo:



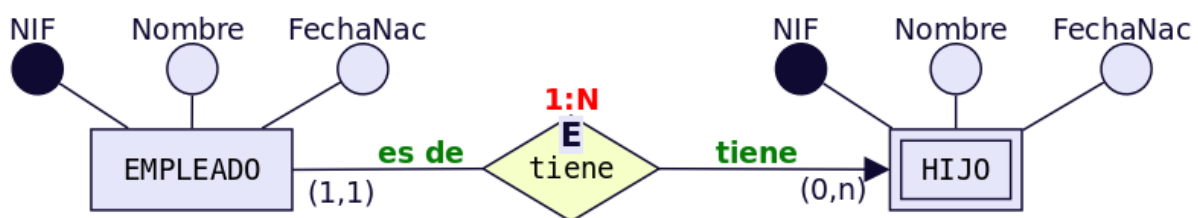
Tenemos la relación «Compra» entre cliente y producto. Así un cliente puede comprar uno o varios productos, y un producto puede ser comprado por uno o varios clientes. Encontramos una serie de atributos propios de cada una de las entidades [CLIENTE (Cod_Cliente, Nombre, Dirección, edad, teléfono) y PRODUCTO (Cod_Producto, Nombre, Descripción, Precio_Unidad)], pero también podemos observar como el atributo «Cantidad» es un atributo de la relación. ¿Por qué? Pues porque un mismo cliente puede comprar distintas cantidades de distintos productos y un mismo producto puede ser comprado en distintas cantidades por distintos clientes. Es decir el atributo cantidad depende del cliente y del producto de que se traten.

Relaciones de dependencia: Entidades Fuertes y Entidades Débiles

Al definir las entidades hablamos de dos tipos de ellas: fuertes y débiles. Una entidad débil está unida a una entidad fuerte a través de una relación de dependencia. Hay dos tipos de relaciones de dependencia:

Dependencia en existencia Se produce cuando una entidad débil necesita de la presencia de una fuerte para existir. Si desaparece la existencia de la entidad fuerte, la de la débil carece de sentido. Se representa con una barra atravesando el rombo y la letra E en su interior. Son relaciones poco frecuentes.

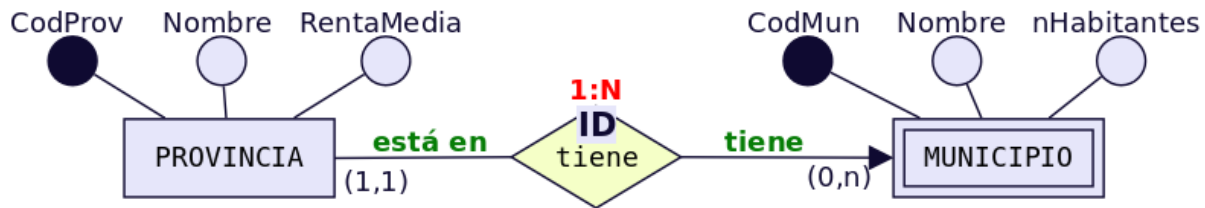
Ejemplo:



En la figura se muestra el caso de que un empleado puede tener ninguno, uno o varios hijos, por lo que los datos de los hijos deben sacarse en una entidad aparte, aunque siguen siendo datos propios de un empleado. Si se eliminase un registro de un empleado, no tendría sentido seguir manteniendo en la base datos la información sobre sus hijos.

Dependencia en identificación Se produce cuando una entidad débil necesita de la fuerte para identificarse. Por sí sola la débil no es capaz de identificar de manera unívoca sus ocurrencias. La clave de la entidad débil se forma al unir la clave de la entidad fuerte con los atributos identificadores de la entidad débil.

Ejemplo:

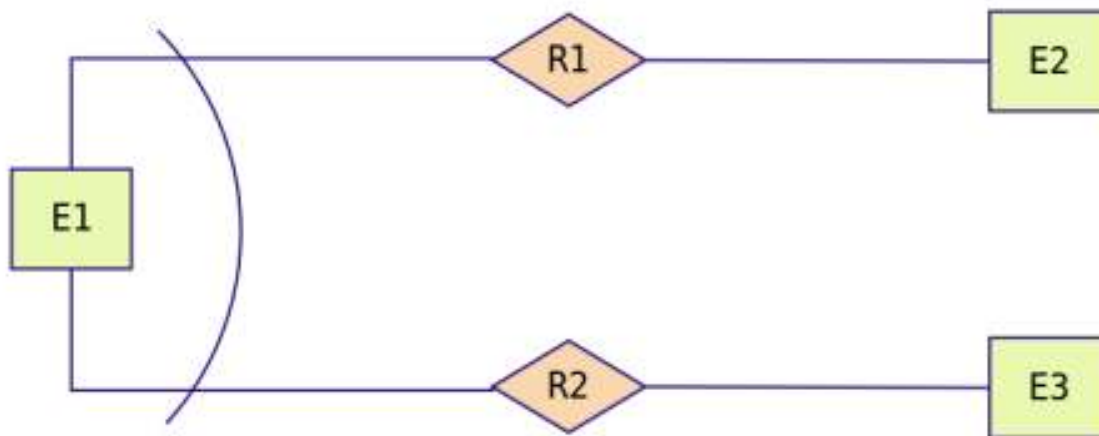


En la figura se observa que la provincia tiene uno o varios municipios y que un municipio pertenece a una sola provincia. Ahora bien si lo que identifica a los municipios es el código que aparece en el código postal, se tiene que las dos primeras cifras corresponden al código de la provincia y las tres últimas al del municipio. Por ejemplo, el C.P de Écija es 41400, dónde 41 es el código de la provincia y 400 el del municipio. De esta forma, habrá distintos municipios con código 400 en distintas provincias. Uno de estos municipios se distinguirá del resto al anteponerle las dos primeras cifras correspondientes al código de la provincial.

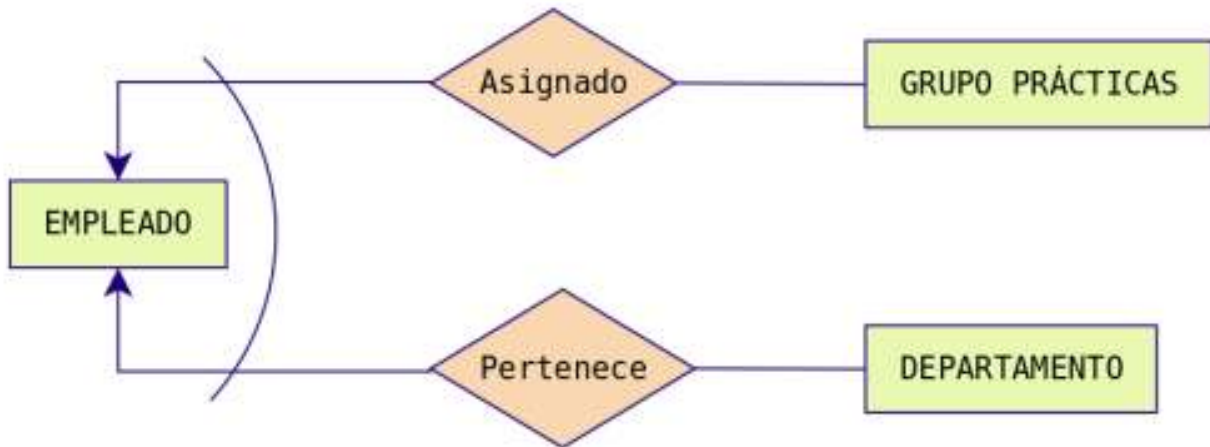
Símbolos de exclusividad o inclusividad entre relaciones

Otros símbolos usados en el modelo E/R son los siguientes:

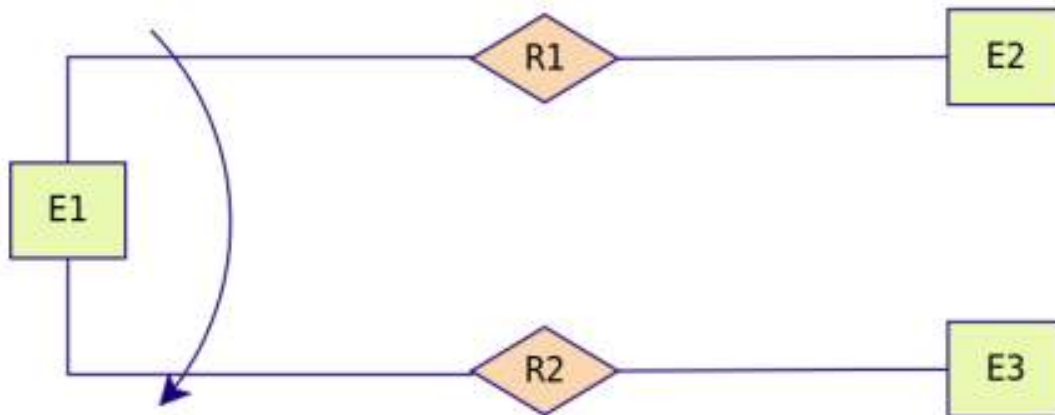
Restricción de exclusividad entre dos tipos de relaciones R1 y R2 respecto a la entidad E1. Significa que E1 está relacionada, o bien con E2 o bien con E3, pero no pueden darse ambas relaciones simultáneamente.



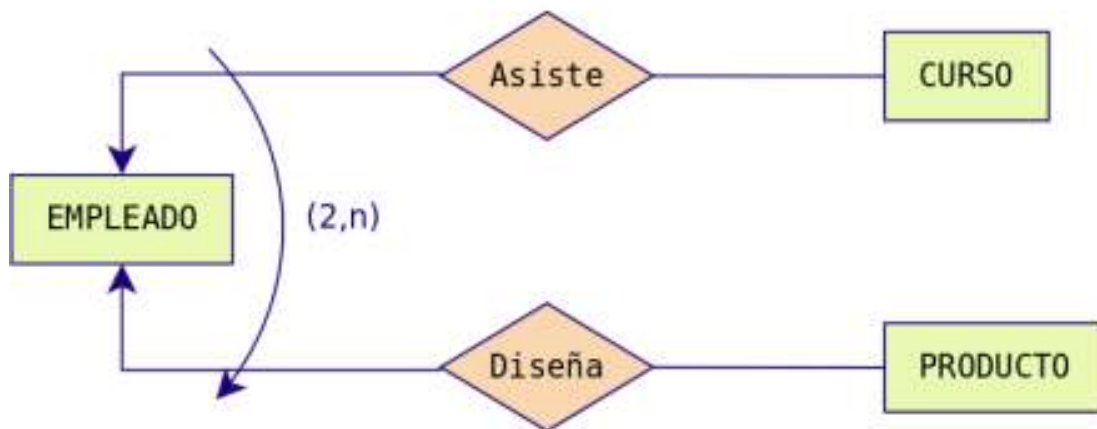
Ejemplo: Un empleado puede estar en una empresa, o bien realizando prácticas, en cuyo caso está asignado a un grupo de prácticas y no pertenece a ningún departamento en concreto. O bien puede ser empleado en plantilla y en este caso pertenece a un departamento.



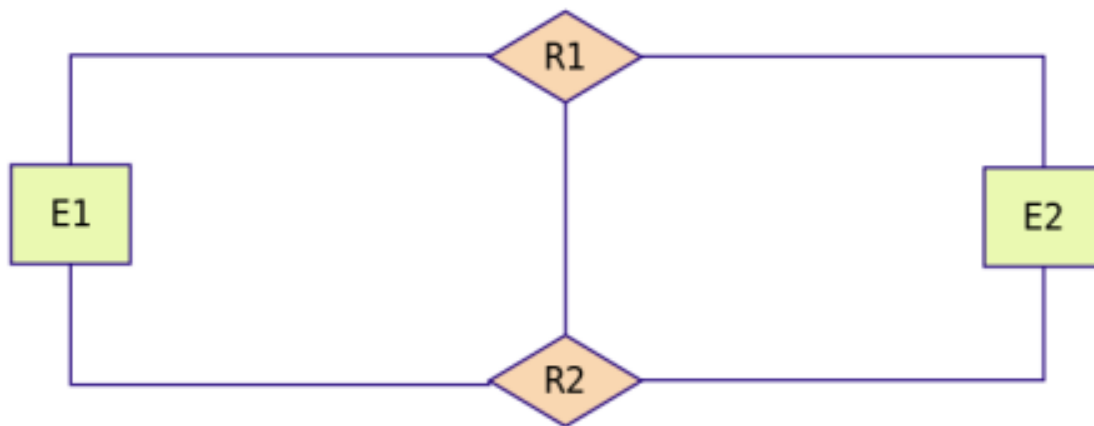
Restricción de inclusividad entre dos tipos de relaciones $R1$ y $R2$ respecto a la entidad $E1$. Para que la entidad $E1$ participe en la relación $R2$ debe participar previamente en la relación $R1$.



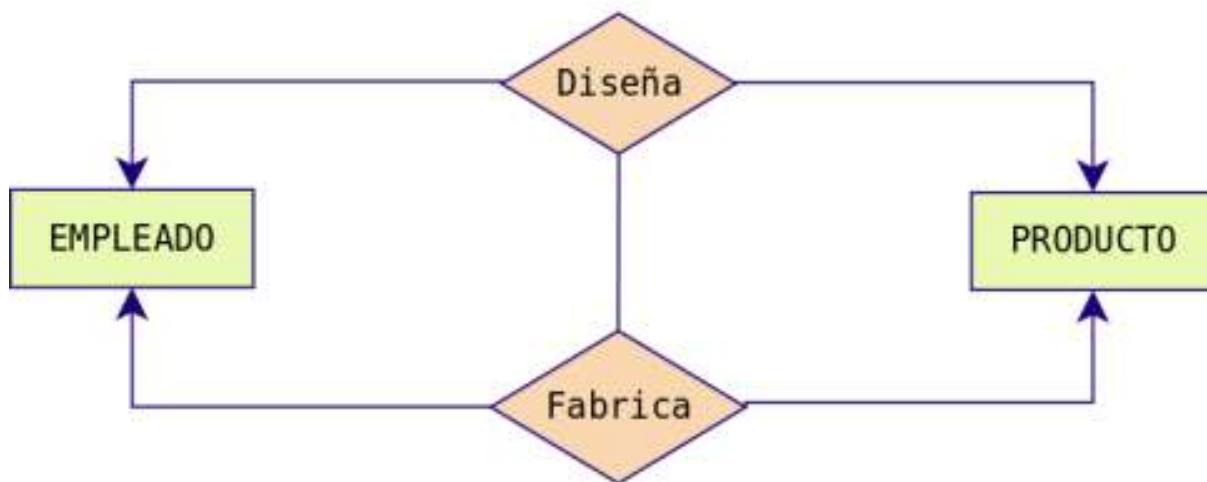
Ejemplo: Para que un empleado pueda trabajar como diseñador de productos debe haber asistido, al menos, a dos cursos.



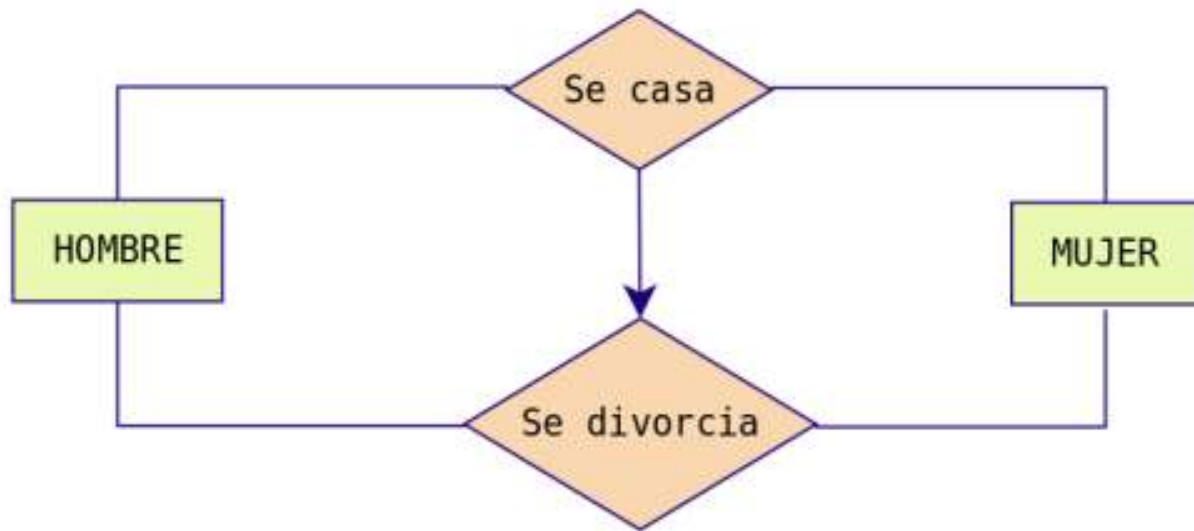
Restricción de exclusión entre dos tipos de relaciones R1 y R2. Significa que E1 está relacionada con E2 bien mediante R1, o bien mediante R2 pero que no pueden darse ambas relaciones simultáneamente.



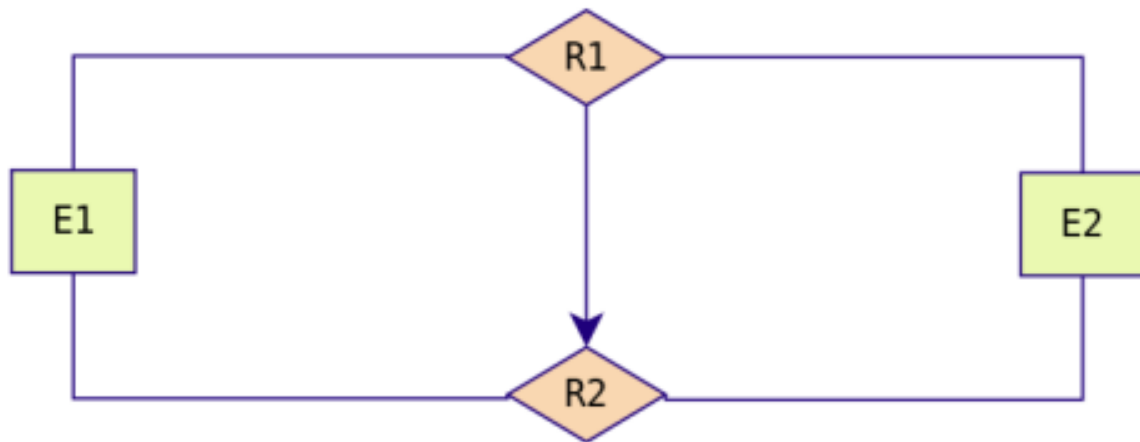
Ejemplo: Los empleados, en función de sus capacidades, o son diseñadores de productos o son operarios y los fabrican, no es posible que ningún empleado sea diseñador y fabricante a la misma vez.



Restricción de inclusión entre dos tipos de relaciones R1 y R2. Para que la entidad E1 participe en la relación R2 con E2 debe participar previamente en la relación R1.



Ejemplo: Para que un hombre se divorcie de una mujer, previamente ha de haberse casado con ella.

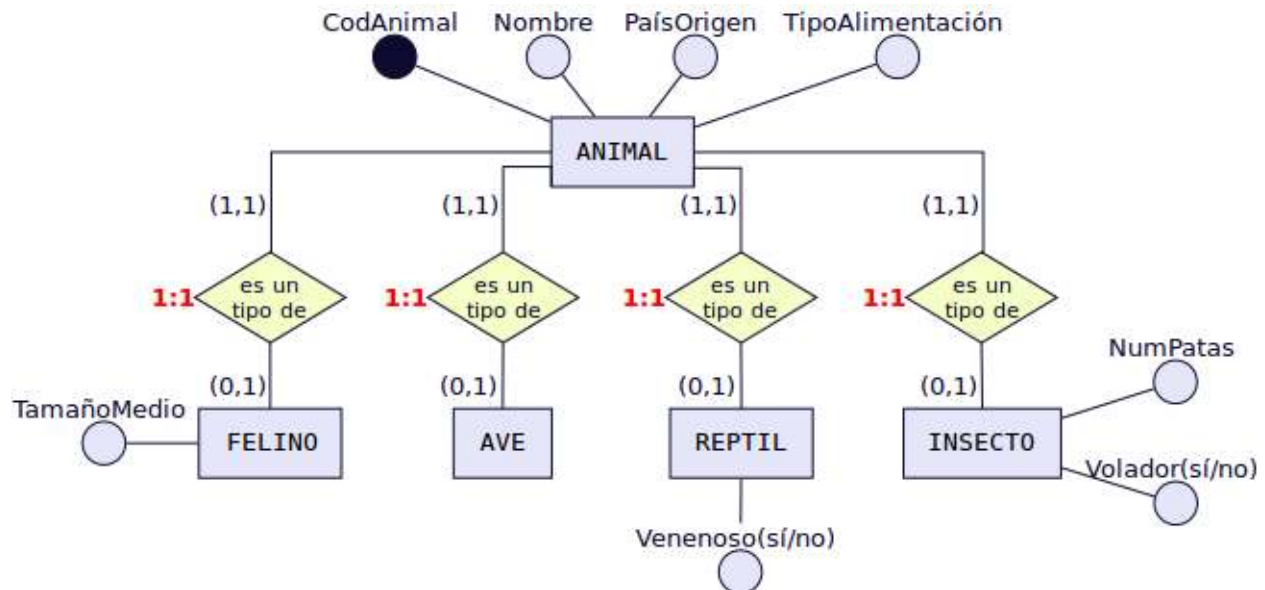


2.4 MODELO E/R EXTENDIDO

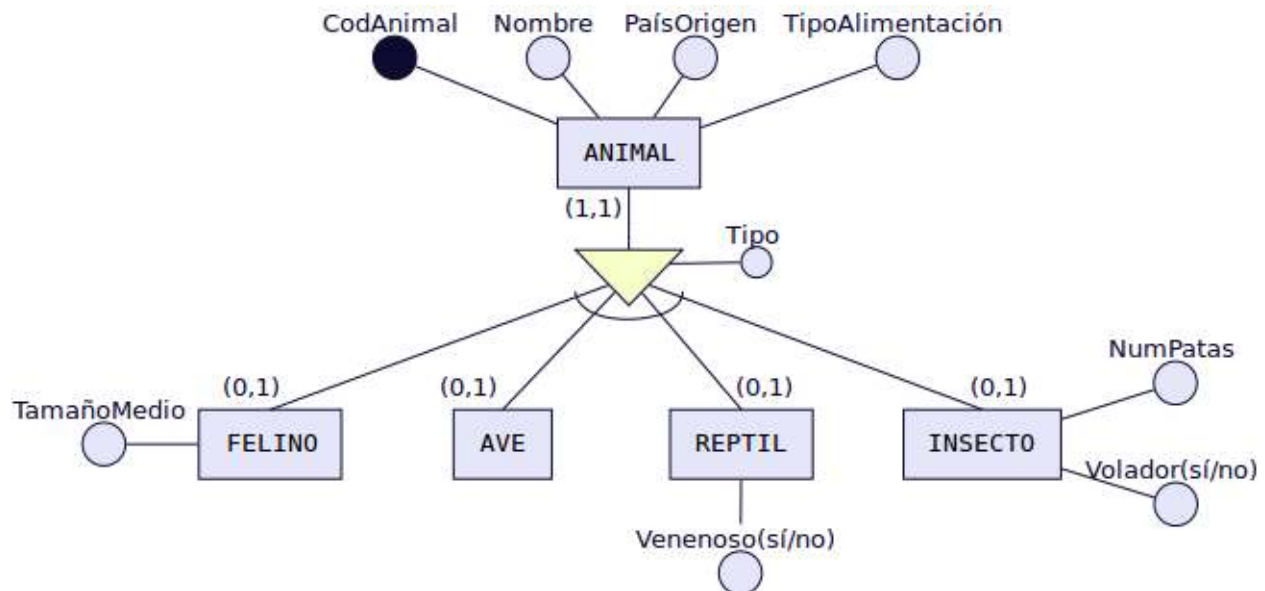
El modelo Entidad/Relación extendido incluye todo lo visto en el modelo Entidad/Relación pero además las **Relaciones de Jerarquía**. Una relación de jerarquía se produce cuando una entidad se puede relacionar con otras a través de una relación cuyo rol sería «Es un tipo de».

Por ejemplo, imaginemos la siguiente situación.

Queremos hacer una BD sobre los animales de un Zoo. Tenemos las entidades ANIMAL, FELINO, AVE, REPTIL, INSECTO. FELINO, AVE, REPTIL e INSECTO tendrían el mismo tipo de relación con ANIMAL: «son un tipo de». Ahora bien, su representación mediante el E/R clásico sería bastante engorrosa:



Para evitar tener que repetir tantas veces el rombo de la misma relación, se utilizan unos símbolos especiales para estos casos y se sustituyen todos los rombos de relación «es un tipo de» por un triángulo invertido, donde la entidades de abajo son siempre un tipo de la entidad de arriba y se llaman subtipo e entidades hijas. La de arriba se denominará supertipo o entidad padre. Las relaciones jerárquicas siempre se hacen en función de un atributo que se coloca al lado de la relación «es_un». En la figura siguiente sería «tipo». El ejemplo anterior quedaría del modo siguiente utilizando símbolos del E/R extendido.

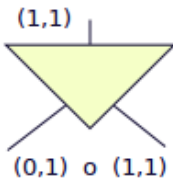
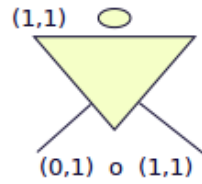
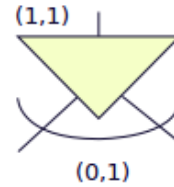
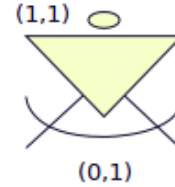


2.4.1 Relaciones de Jerarquía

Vamos a ver los distintos tipos de relaciones de jerarquía existentes:

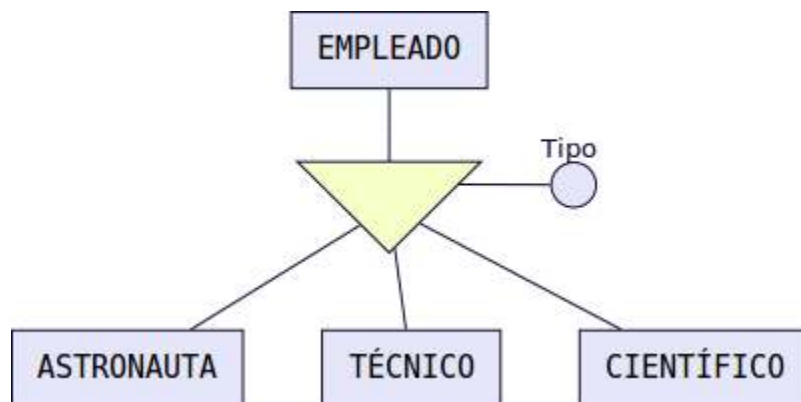
- **Total:** Subdividimos la entidad Empleado en: Ingeniero, Secretario y Técnico y en nuestra BD no hay ningún otro empleado que no pertenezca a uno de estos tres tipos.
- **Parcial:** Subdividimos la entidad Empleado en: Ingeniero, Secretario y Técnico pero en nuestra BD puede haber empleados que no pertenezcan a ninguno de estos tres tipos.

- **Solapada:** Subdividimos la entidad Empleado, en: Ingeniero, Secretario y Técnico y en nuestra BD puede haber empleados que sean a la vez Ingenieros y secretarios, o secretarios y técnicos, etc.
- **Exclusiva:** Subdividimos la entidad Empleado en: Ingeniero, Secretario y Técnico. En nuestra BD ningún empleado pertenece a más de una subentidad.

**Solapada y Parcial****Solapada y Total****Exclusiva y Parcial****Exclusiva y Total**

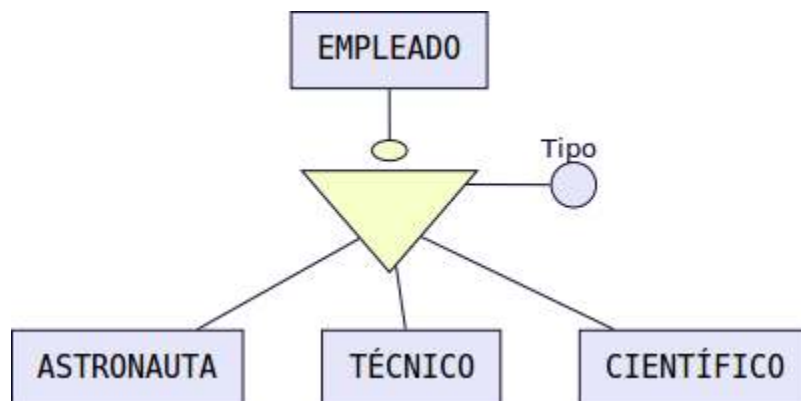
Ejemplos:

Jerarquía solapada y parcial



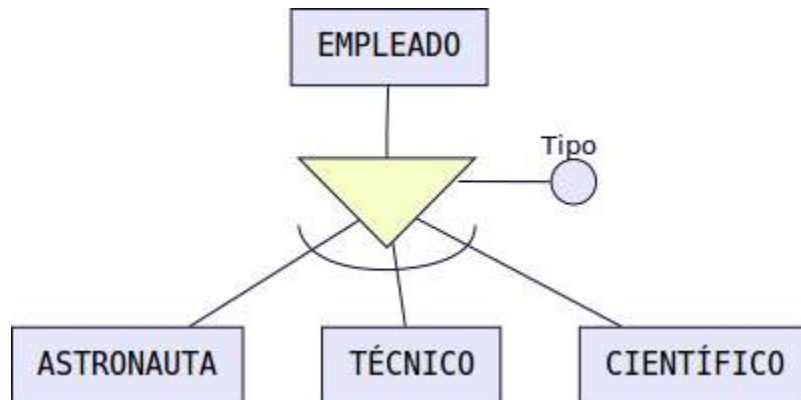
En esta BD un empleado podría ser simultáneamente técnico, científico y astronauta o técnico y astronauta, etc. (solapada). Además puede ser técnico, astronauta, científico o desempeñar otro empleo diferente (parcial).

Jerarquía solapada y total



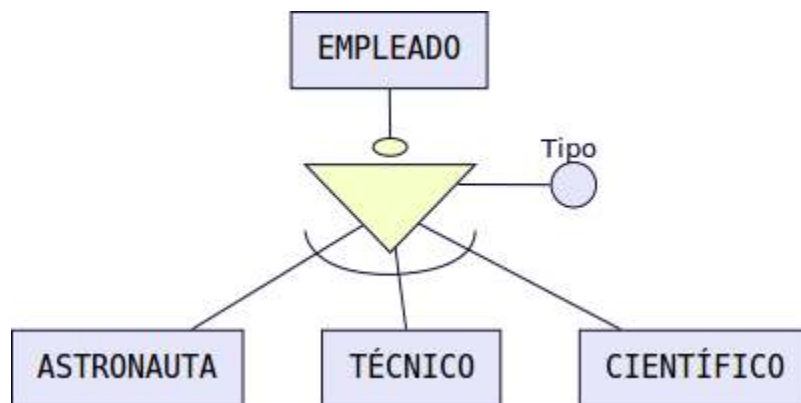
En esta BD un empleado podría ser simultáneamente técnico, científico y astronauta o técnico y astronauta, etc. (solapada). Además puede ser solamente técnico, astronauta o científico (total).

Jerarquía exclusiva y parcial



En esta BD un empleado sólo puede desempeñar una de las tres ocupaciones (exclusiva) . Además puede ser técnico, o ser astronauta, o ser científico o también desempeñar otro empleo diferente, por ejemplo, podría ser FÍSICO (parcial).

Jerarquía exclusiva y total



Un empleado puede ser solamente técnico, astronauta o científico (total) y no ocupar más de un puesto (exclusiva)

Nota: Podéis observar que en los ejemplos hemos omitido las participaciones. La mayoría de las veces estas no se ponen.

2.5 MODELO RELACIONAL

2.5.1 Introducción

Los SGBD se pueden clasificar de acuerdo con el modelo lógico que soportan, el número de usuarios, el número de puestos, el coste... La clasificación más importante de los SGBD se basa en el modelo lógico, siendo los principales modelos que se utilizan en el mercado los siguientes: Jerárquico, en Red, Relacional y Orientado a Objetos.

La mayoría de los SGBD comerciales actuales están basados en el modelo relacional, en el que nos vamos a centrar, mientras que los sistemas más antiguos estaban basados en el modelo de red o el modelo jerárquico.

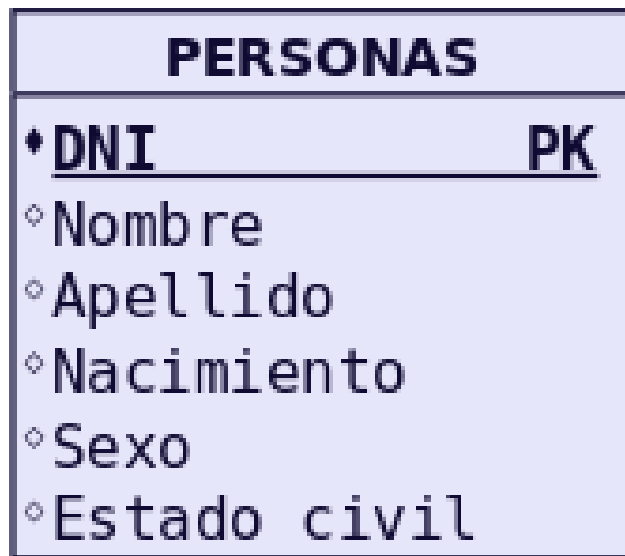
Los motivos del éxito del modelo relacional son fundamentalmente dos: - Se basan en el álgebra relacional que es un modelo matemático con sólidos fundamentos. En esta sección se presenta el modelo relacional. Realizaremos la descripción de los principios básicos del modelo relacional: la estructura de datos relacional y las reglas de integridad. Ofrecen sistemas simples y eficaces para representar y manipular los datos. - La estructura fundamental del modelo relacional es precisamente esa, la «**relación**», es decir una tabla bidimensional constituida por **filas** (registros o tuplas) y **columnas** (atributos o campos). Las relaciones o tablas representan las entidades del modelo E/R, mientras que los atributos de la relación representarán las propiedades o atributos de dichas entidades. Por ejemplo, si en la base de datos se tienen que representar la entidad PERSONA, está pasará a ser una relación o tabla llamada «PERSONAS», cuyos atributos describen las características de las personas (tabla siguiente). Cada tupla o registro de la relación «PERSONAS» representará una persona concreta.

2.5.2 Estructura de datos relacional

Tabla 1: PERSONAS

D.N.I.	Nombre	Apellido	Nacimiento	Sexo	Estado civil
52.768.987	Juan	Loza	15/06/1976	H	Soltero
06.876.983	Isabel	Gálvez	23/12/1969	M	Casada
34.678.987	Micaela	Ruiz	02/10/1985	M	Soltera

En realidad, siendo rigurosos, una RELACIÓN del MODELO RELACIONAL es sólo la definición de la estructura de la tabla, es decir su nombre y la lista de los atributos que la componen. Una representación de la definición de esa relación podría ser la siguiente:



Para distinguir un registro de otro, se usa la «**clave primaria o clave principal**».

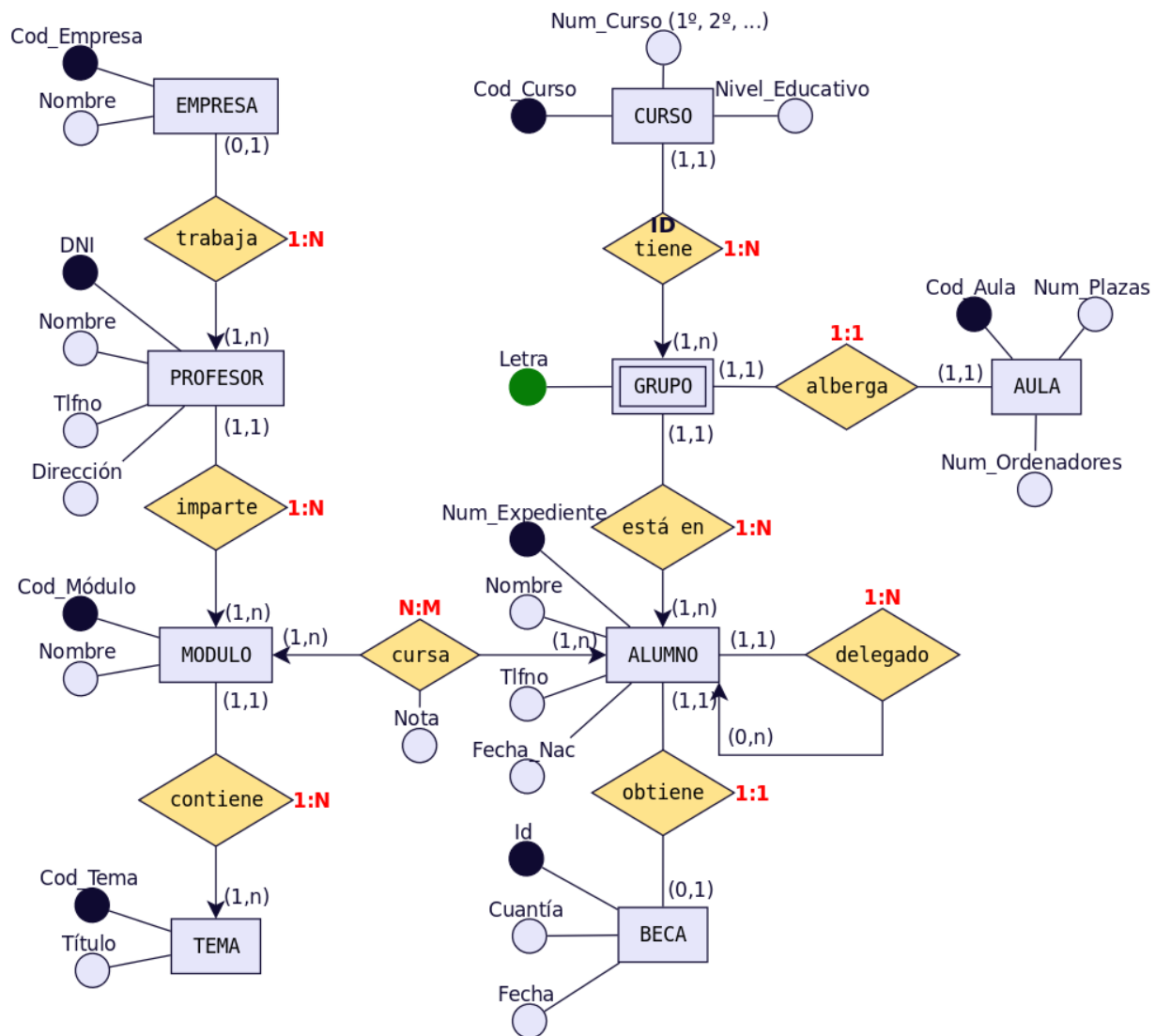
En una relación puede haber más combinaciones de atributos que permitan identificar unívocamente una fila (estos se llamarán «llaves o claves candidatas»), pero entre éstas se elegirá una sola para utilizar como llave primaria. Los atributos de la llave primaria no pueden asumir el valor nulo.

2.5.3 Elementos y propiedades del modelo relacional

- **Relación (tabla):** Representan las entidades de las que se quiere almacenar información en la BD. Esta formada por:
 - **Filas (Registros o Tuplas):** Corresponden a cada ocurrencia de la entidad.
 - **Columnas (Atributos o campos):** Corresponden a las propiedades de la entidad. Siendo rigurosos una relación está constituida sólo por los atributos, sin las tuplas.
- Las relaciones tienen las siguientes **propiedades**:
 - Cada relación tiene un nombre y éste es distinto del nombre de todas las demás relaciones de la misma BD.
 - No hay dos atributos que se llamen igual en la misma relación.
 - El orden de los atributos no importa: los atributos no están ordenados.
 - Cada tupla es distinta de las demás: no hay tuplas duplicadas. (Como mínimo se diferenciarán en la clave principal)
 - El orden de las tuplas no importa: las tuplas no están ordenadas.
- **Clave candidata:** atributo que identifica unívocamente una tupla. Cualquiera de las claves candidatas se podría elegir como clave principal.
- **Clave Principal:** Clave candidata que elegimos como identificador de la tuplas.
- **Clave Alternativa:** Toda clave candidata que no es clave primaria (las que no hayamos elegido como clave principal)
- Una clave principal no puede asumir el valor nulo (**Integridad de la entidad**).
- **Dominio de un atributo:** Conjunto de valores que pueden ser asumidos por dicho atributo.
- **Clave Externa o foránea o ajena:** el atributo o conjunto de atributos que forman la clave principal de otra relación. Que un atributo sea clave ajena en una tabla significa que para introducir datos en ese atributo, previamente han debido introducirse en la tabla de origen. Es decir, los valores presentes en la clave externa tienen que corresponder a valores presentes en la clave principal correspondiente (**Integridad Referencial**).

2.5.4 Transformación de un esquema E/R a esquema relacional

Pasamos ya a enumerar las normas para traducir del Modelo E/R al modelo relacional, ayudándonos del siguiente ejemplo:



Nota: Al pasar del esquema E/R al esquema Relacional deberemos añadir las **claves foráneas** necesarias para establecer las interrelaciones entre las tablas. Dichas claves foráneas no aparecen representadas en el esquema E/R.

Importante: Se deben elaborar los diagramas relacionales de tal forma que, posteriormente al introducir datos, **no quede ninguna clave foránea a valor nulo (NULL)**. Para ello se siguen las reglas que se muestran a continuación.

Entidades

Cada entidad se transforma en una tabla. El identificador (o identificadores) de la entidad pasa a ser la clave principal de la relación y aparece subrayada o con la indicación: **PK (Primary Key)**. Si hay clave alternativa esta se pone en «negrita».

Ejemplo: Todas las entidades del ejemplo anterior generan tabla. En concreto, la entidad AULA genera la siguiente

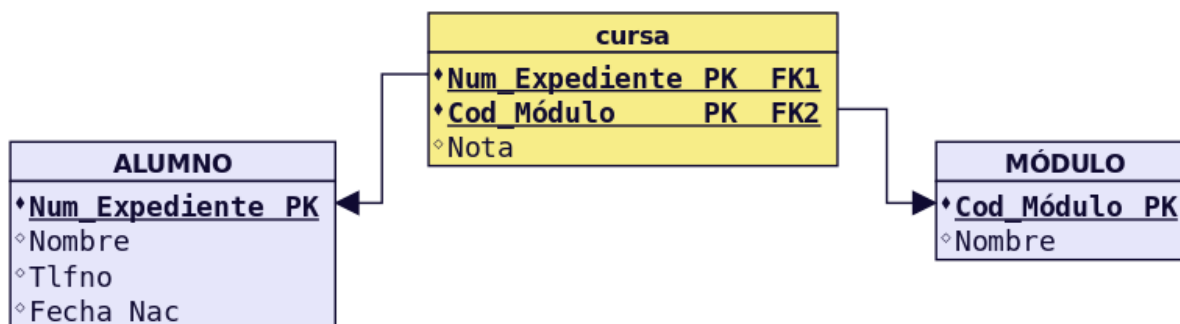
tabla:



Relaciones binarias (de grado 2)

Relaciones N:M: Es el caso más sencillo. **Siempre generan tabla.** Se crea una tabla que incorpora como claves ajenas o foráneas **FK (Foreign Key)** cada una de las claves de las entidades que participan en la relación. La clave principal de esta nueva tabla está compuesta por dichos campos. Es importante resaltar que no se trata de 2 claves primarias, sino de una clave primaria compuesta por 2 campos. Si hay atributos propios, pasan a la tabla de la relación. Se haría exactamente igual si hubiera participaciones mínimas 0. Orden de los atributos en las claves compuestas: Se deben poner a la izquierda todos los atributos que forman la clave. El orden de los atributos que forman la clave vendrá determinado por las consultas que se vayan a realizar. Las tuplas de la tabla suelen estar ordenadas siguiendo como índice la clave. Por tanto, conviene poner primero aquel/los atributos por los que se va a realizar la consulta.

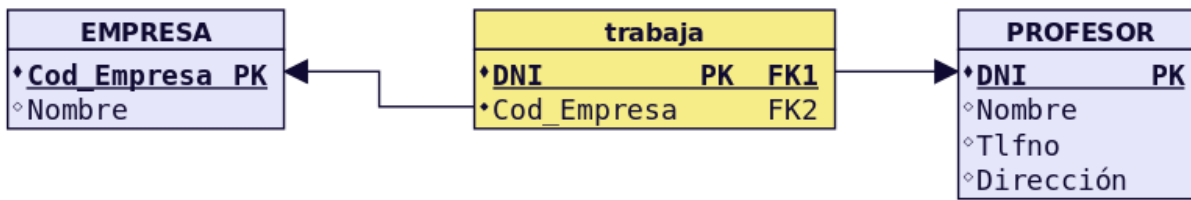
Ejemplo: Realicemos el paso a tablas de la relación N:M entre MÓDULO (1,n) y ALUMNO (1,n). Este tipo de relación siempre genera tabla y los atributos de la relación, pasan a la tabla que ésta genera.



Relaciones 1:N: Por lo general no generan tabla. Se dan 2 casos:

- Caso 1: Si la entidad del lado «1» presenta participación (0,1), entonces se crea una nueva tabla para la relación que incorpora como claves ajenas las claves de ambas entidades. La clave principal de la relación será sólo la clave de la entidad del lado «N».
- Caso 2: Para el resto de situaciones, la entidad del lado «N» recibe como clave ajena la clave de la entidad del lado «1». Los atributos propios de la relación pasan a la tabla donde se ha incorporado la clave ajena.

Ejemplo de caso 1: Realicemos el paso a tablas de la relación 1:N entre PROFESOR (1,n) y EMPRESA (0,1). Como en el lado «1» encontramos participación mínima 0, se generará una nueva tabla.



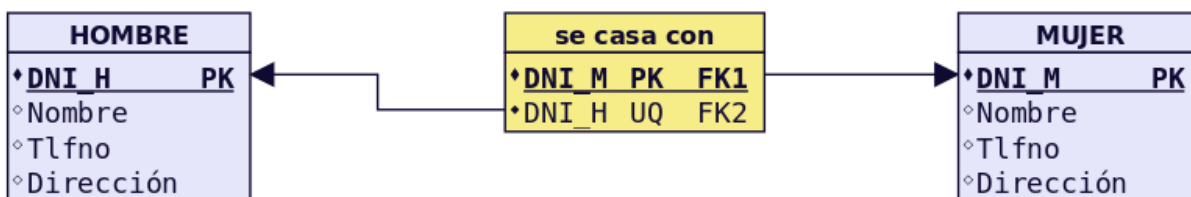
Ejemplo de caso 2: Realicemos el paso a tablas de la relación 1:N entre MÓDULO (1,1) y TEMA (1,n). Como no hay participación mínima «0» en el lado 1, no genera tabla y la clave principal del lado «1» pasa como foránea al lado «n».



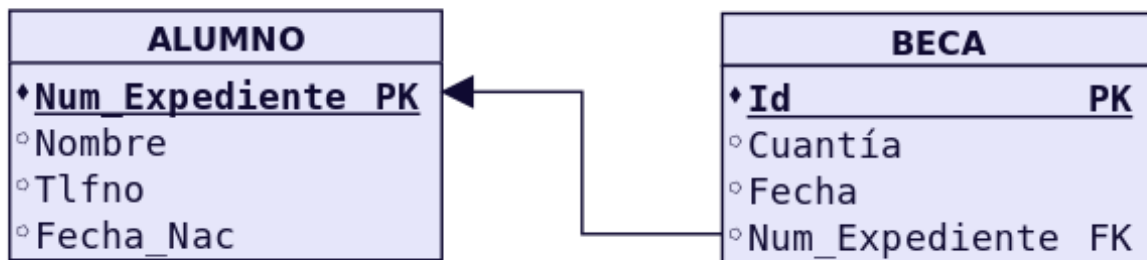
Relaciones 1:1: Por lo general no generan tabla. Se dan 3 casos:

- Caso 1: Si las dos entidades participan con participación (0,1), entonces se crea una nueva tabla para la relación.
- Caso 2: Si alguna entidad, pero no las dos, participa con participación mínima 0 (0,1), entonces se pone la clave ajena en dicha entidad, para evitar en lo posible, los valores nulos.
- Caso 3: Si tenemos una relación 1:1 y ninguna tiene participación mínima 0, elegimos la clave principal de una de ellas y la introducimos como clave ajena en la otra tabla. Se elegirá una u otra forma en función de como se quiera organizar la información para facilitar las consultas. Los atributos propios de la relación pasan a la tabla donde se introduce la clave ajena.

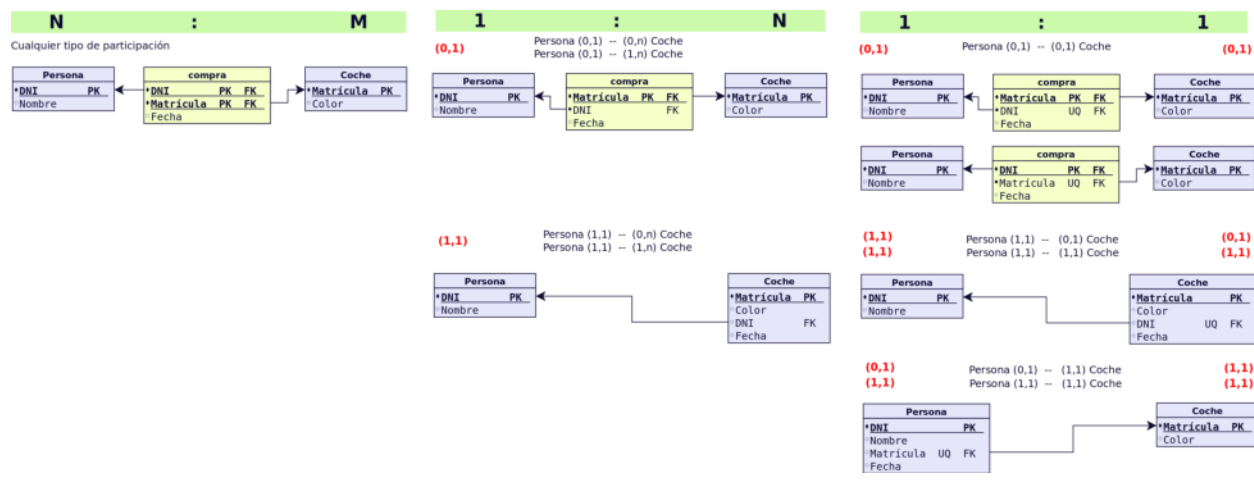
Ejemplo de caso 1: No se presenta ninguna situación así en el esquema estudiado. Una situación donde puede darse este caso es en HOMBRE (0,1) se casa con MUJER (0,1). Es similar al caso 1 del apartado anterior en relaciones 1:N, aunque en este caso debemos establecer una restricción de valor único para FK2.



Ejemplo de caso 2 (y 3): Realicemos el paso a tablas de la relación 1:1 entre ALUMNO (1,1) y BECA (0,1). Como BECA tiene participación mínima 0, incorporamos en ella, como clave foránea, la clave de ALUMNO. Esta forma de proceder también es válida para el caso 3, pudiendo acoger la clave foránea cualquiera de las entidades.



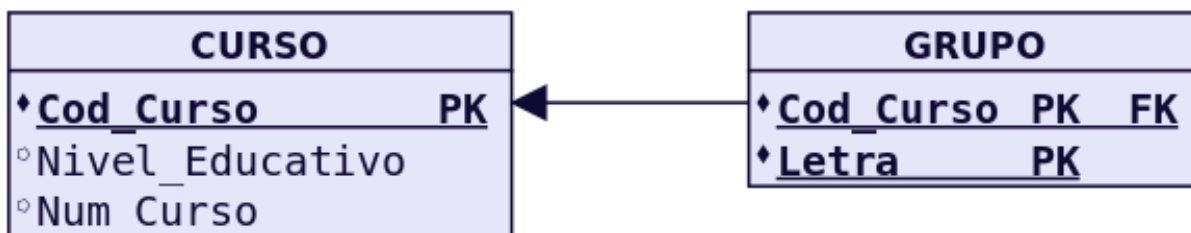
Truco: A continuación se muestra un resumen de los casos disponibles en relaciones N:M, 1:N y 1:1.



Relaciones de dependencia (Siempre de grado 2 y cardinalidad 1:N)

Relaciones de dependencia en existencia: Se comportan como una 1:N normal. La clave principal del lado 1 pasa al lado «N» como foránea (hacia adonde apunta la flecha) Ejemplo: No encontramos ningún ejemplo, reseñado como tal, en el supuesto anterior. Ahora bien, se comportan en el paso a tablas como cualquier otra relación 1:N. Sólo se tendría en cuenta, el hecho de ser débil en existencia para en el momento de creación de la BD, imponer que al borrar una ocurrencia en el lado «1», se borren las asociadas en el lado «n».

Relaciones de dependencia en identificación: Por lo general no generan tablas, porque suelen ser 1:1 o 1:N. Como en toda relación 1:N, La clave de la entidad fuerte debe introducirse en la tabla de la entidad débil como foránea y, además en este caso, formar parte de la clave de ésta. En las entidades débiles, la clave de la entidad fuerte debe ir la primera y, a continuación, los discriminadores de la débil. Ejemplo: Realicemos el paso a tablas de la relación débil en identificación entre CURSO Y GRUPO.



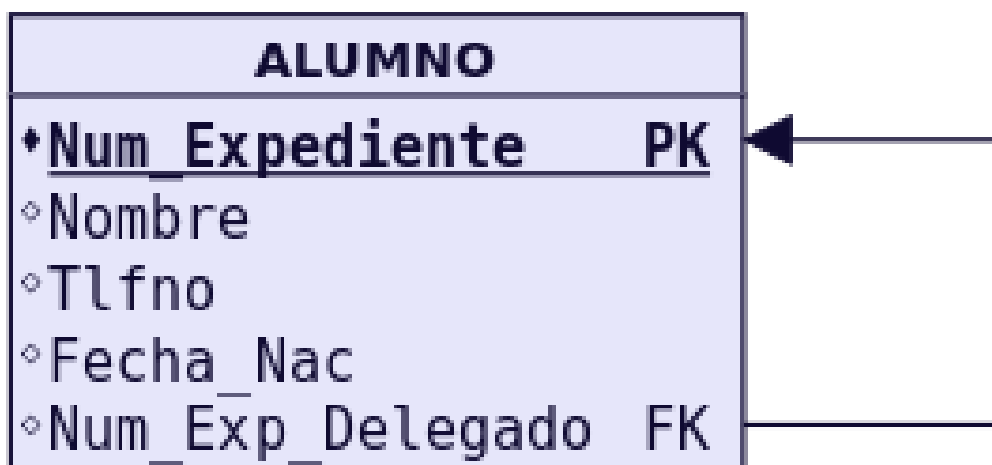
Relaciones de grado mayor que 2

Relaciones n-arias (solo veremos hasta grado 3): Siempre generan tabla. Las claves principales de las entidades que participan en la relación pasan a la nueva tabla como claves foráneas. Y solo las de los lados «n» forman la principal. Si hay atributos propios de la relación, estos se incluyen en esa tabla. Ejemplo: No encontramos ningún ejemplo de relación de más de grado 2 en el supuesto anterior. Se verán cuando aparezcan en algún ejercicio.

Relaciones reflexivas

Relaciones Reflexivas o Recursivas: Generan tabla o no en función de la cardinalidad. Si es 1:1, no genera tabla. En la entidad se introduce dos veces la clave, una como clave principal y otra como clave ajena. Se suele introducir una modificación en el nombre por diferenciarlas. Si es 1:N, se puede generar tabla o no. Si hubiese participación 0 en el lado 1, obligatoriamente se generaría tabla. Si es N:N, la relación genera tabla.

Ejemplo: Realicemos el paso a tablas de la relación reflexiva de ALUMNO. Como no tiene participación mínima «0» en el lado 1, no genera tabla. La clave principal de ALUMNOS, volverá a aparecer en ALUMNOS como clave foránea, igual que en cualquier relación 1:N. Ahora bien, como no puede haber dos campos con el mismo nombre en la misma tabla, deberemos cambiar un poco el nombre de la clave principal, para que haga referencia al papel que ocupa como clave foránea.



Jerarquías

Eliminación de las relaciones jerárquicas: Las relaciones jerárquicas son un caso especial. Se pueden dar algunas guías que sirvan de referencia, pero en la mayoría de los casos, va a depender del problema concreto. Estas guías son: Se creará una tabla para la entidad supertipo. A no ser que tenga tan pocos atributos que dejarla sea una complicación. Si la entidad subtipo no tiene atributos y no está relacionada con ninguna otra entidad, desaparece. Si la entidad subtipo tiene algún atributo, se crea una tabla. Si no tiene clave propia, hereda la de la entidad supertipo. Si la relación es exclusiva, el atributo que genera la jerarquía se incorpora en la tabla de la entidad supertipo. Si se ha creado una tabla por cada una de las entidades subtipo, no es necesario incorporar dicho atributo a la entidad supertipo. Ejemplo: No encontramos ningún ejemplo de relación de jerarquía 2 en el supuesto anterior. Su paso a tablas, se verán en cuando aparezcan en los ejemplos concretos.

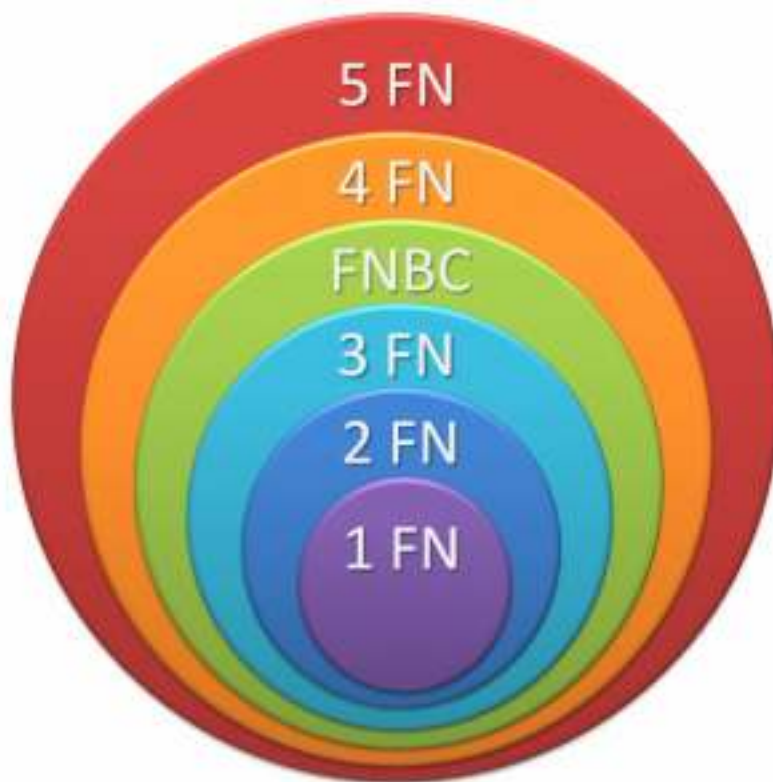
2.6 NORMALIZACIÓN

El diseño de una BD relacional se puede realizar aplicando al mundo real, en una primera fase, un modelo como el modelo E/R, a fin de obtener un esquema conceptual; en una segunda fase, se transforma dicho esquema al modelo relacional mediante las correspondientes reglas de transformación. También es posible, aunque quizás menos recomendable, obtener el esquema relacional sin realizar ese paso intermedio que es el esquema conceptual. En ambos casos, es conveniente (obligatorio en el modelo relacional directo) aplicar un conjunto de reglas, conocidas como Teoría de normalización, que nos permiten asegurar que un esquema relacional cumple unas ciertas propiedades, evitando:

- La redundancia de los datos: repetición de datos en un sistema.
- Anomalías de actualización: inconsistencias de los datos como resultado de datos redundantes y actualizaciones parciales.
- Anomalías de borrado: pérdidas no intencionadas de datos debido a que se han borrado otros datos.
- Anomalías de inserción: imposibilidad de adicionar datos en la base de datos debido a la ausencia de otros datos.

En la práctica, si la BD se ha diseñado haciendo uso de modelos semánticos como el modelo E/R no suele ser necesaria la normalización. Por otro lado **si nos proporcionan una base de datos creada sin realizar un diseño previo, es muy probable que necesitemos normalizar.**

En la teoría de bases de datos relacionales, las **formas normales (FN)** proporcionan los criterios para determinar el grado de vulnerabilidad de una tabla a inconsistencias y anomalías lógicas. Cuanto más alta sea la forma normal aplicable a una tabla, menos vulnerable será a inconsistencias y anomalías. Edgar F. Codd originalmente definió las tres primeras formas normales (**1FN**, **2FN**, y **3FN**) en 1970. Estas formas normales se han resumido como requiriendo que **todos los atributos sean atómicos, dependan de la clave completa y en forma directa (no transitiva)**. La forma normal de Boyce-Codd (**FNBC**) fue introducida en 1974 por los dos autores que aparecen en su denominación. Las cuarta y quinta formas normales (**4FN** y **5FN**) se ocupan específicamente de la representación de las relaciones muchos a muchos y uno a muchos entre los atributos y fueron introducidas por Fagin en 1977 y 1979 respectivamente. Cada forma normal incluye a las anteriores.



Antes de dar los conceptos de formas normales veamos unas definiciones previas:

- **Dependencia funcional:** $A \rightarrow B$, representa que B es funcionalmente dependiente de A. Para **un valor de A** siempre aparece **un valor de B**. Ejemplo: Si A es el D.N.I., y B el Nombre, está claro que para un número de D.N.I., siempre aparece el mismo nombre de titular.
- **Dependencia funcional completa:** $A \rightarrow B$, si B depende de A en su totalidad. Ejemplo: Tiene sentido plantearse este tipo de dependencia cuando A está compuesto por más de un atributo. Por ejemplo, supongamos que A corresponde al atributo compuesto: D.N.I. Empleado + Cod._Dpto. y B es Nombre_Dpto. En este caso B depende del Cod_Dpto., pero no del D.N.I. Empleado. Por tanto no habría dependencia funcional completa.
- **Dependencia transitiva:** $A \rightarrow B \rightarrow C$. Si $A \rightarrow B$ y $B \rightarrow C$, Entonces decimos que C depende de forma transitiva de A. Ejemplo: Sea A el D.N.I. de un alumno, B la localidad en la que vive y C la provincia. Es un caso de dependencia transitiva $A \rightarrow B \rightarrow C$.
- **Determinante funcional:** todo atributo, o conjunto de ellos, de los que depende algún otro atributo. Ejemplo: El D.N.I. es un determinante funcional pues atributos como nombre, dirección, localidad, etc, dependen de él.
- **Dependencia multivaluada:** $A \twoheadrightarrow B$. Son un tipo de dependencias en las que un determinante funcional no implica un único valor, sino un conjunto de ellos. **Un valor de A** siempre implica **varios valores de B**. Ejemplo: CursoBachillerato $\twoheadrightarrow$ Modalidad. Para primer curso siempre va a aparecer en el campo Modalidad uno de los siguientes valores: Ciencias, Humanidades/Ciencias Sociales o Artes. Igual para segundo curso.

2.6.1 Primera Forma Normal: 1FN

Una Relación está en 1FN si y sólo si cada atributo es atómico.

Ejemplo: Supongamos que tenemos la siguiente tabla con datos de alumnado de un centro de enseñanza secundaria.

Tabla 2: Alumnos

DNI	Nom- bre	Cur- so	FechaMa- trícula	Tutor	LocalidadA- lumno	ProvinciaA- lumno	Teléfonos
11111111	AEva	1ESO-A	01-Julio-2016	Isabel	Écija	Sevilla	660111222
22222222	BAAna	1ESO-A	09-Julio-2016	Isabel	Écija	Sevilla	660222333 660333444 660444555
33333333	CSusana	1ESO-B	11-Julio-2016	Roberto	Écija	Sevilla	
44444444	DJuan	2ESO-A	05-Julio-2016	Federico	El Villar	Córdoba	
55555555	EJosé	2ESO-A	02-Julio-2016	Federico	El Villar	Córdoba	661000111 661000222

Como se puede observar, esta tabla no está en 1FN puesto que el campo Teléfonos contiene varios datos dentro de una misma celda y por tanto no es un campo cuyos valores sean atómicos. La solución sería la siguiente:

Tabla 3: Alumnos

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno
11111111A	Eva	1ESO-A	01-Julio-2016	Isabel	Écija	Sevilla
22222222B	Ana	1ESO-A	09-Julio-2016	Isabel	Écija	Sevilla
33333333C	Susana	1ESO-B	11-Julio-2016	Roberto	Écija	Sevilla
44444444D	Juan	2ESO-A	05-Julio-2016	Federico	El Villar	Córdoba
55555555E	José	2ESO-A	02-Julio-2016	Federico	El Villar	Córdoba

Tabla 4: Teléfonos

DNI	Teléfono
11111111A	660111222
22222222B	660222333
22222222B	660333444
22222222B	660444555
55555555E	661000111
55555555E	661000222

2.6.2 Segunda Forma Normal: 2FN

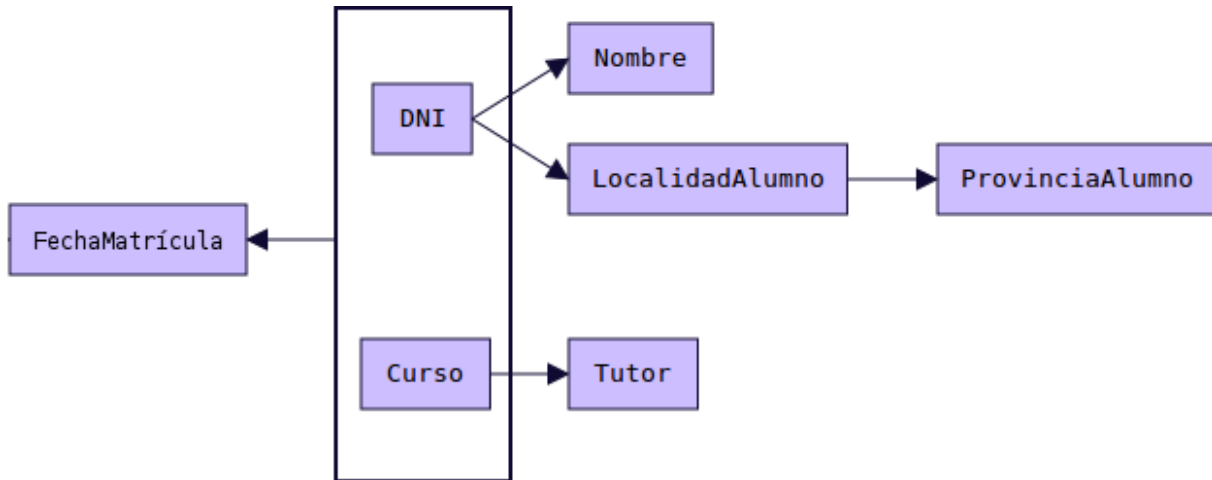
Una Relación esta en 2FN si y sólo si está en 1FN y todos los atributos que no forman parte de la Clave Principal tienen dependencia funcional completa de ella.

Ejemplo: Seguimos con el ejemplo anterior. Trabajaremos con la siguiente tabla:

Tabla 5: Alumnos

DNI	Nombre	Curso	FechaMatrícula	Tutor	LocalidadAlumno	ProvinciaAlumno
11111111A	Eva	1ESO-A	01-Julio-2016	Isabel	Écija	Sevilla
22222222B	Ana	1ESO-A	09-Julio-2016	Isabel	Écija	Sevilla
33333333C	Susana	1ESO-B	11-Julio-2016	Roberto	Écija	Sevilla
44444444D	Juan	2ESO-A	05-Julio-2016	Federico	El Villar	Córdoba
55555555E	José	2ESO-A	02-Julio-2016	Federico	El Villar	Córdoba

Vamos a examinar las dependencias funcionales. El gráfico que las representa es el siguiente:



- Siempre que aparece un DNI aparecerá el Nombre correspondiente y la LocalidadAlumno correspondiente. Por tanto $DNI \rightarrow Nombre$ y $DNI \rightarrow LocalidadAlumno$. Por otro lado siempre que aparece un Curso aparecerá el Tutor correspondiente. Por tanto $Curso \rightarrow Tutor$. Los atributos Nombre y LocalidadAlumno no dependen funcionalmente de Curso, y el atributo Tutor no depende funcionalmente de DNI.
- El único atributo que sí depende de forma completa de la clave compuesta DNI y Curso es FechaMatrícula: $(DNI, Curso) \rightarrow FechaMatrícula$.

A la hora de establecer la Clave Primaria de una tabla debemos escoger un atributo o conjunto de ellos de los que dependan funcionalmente el resto de atributos. Además debe ser una dependencia funcional completa. Si escogemos DNI como clave primaria, tenemos un atributo (Tutor) que no depende funcionalmente de él. Si escogemos Curso como clave primaria, tenemos otros atributos que no dependen de él.

Si escogemos la combinación (DNI, Curso) como clave primaria, entonces sí tenemos todo el resto de atributos con dependencia funcional respecto a esta clave. Pero es una dependencia parcial, no total (salvo FechaMatrícula, donde sí existe dependencia completa). Por tanto esta tabla no está en 2FN. La solución sería la siguiente:

Tabla 6: Alumnos

DNI	Nombre	Localidad	Provincia
11111111A	Eva	Écija	Sevilla
22222222B	Ana	Écija	Sevilla
33333333C	Susana	El Villar	Córdoba
44444444D	Juan	El Villar	Córdoba
55555555E	José	Écija	Sevilla

Tabla 7: Matrículas

DNI	Curso	FechaMatrícula
11111111A	1ESO-A	01-Julio-2016
22222222B	1ESO-A	09-Julio-2016
33333333C	1ESO-B	11-Julio-2016
44444444D	2ESO-A	05-Julio-2016
55555555E	2ESO-A	02-Julio-2016

Tabla 8: Cursos

Curso	Tutor
1ESO-A	Isabel
1ESO-B	Roberto
2ESO-A	Federico

2.6.3 Tercera Forma Normal: 3FN

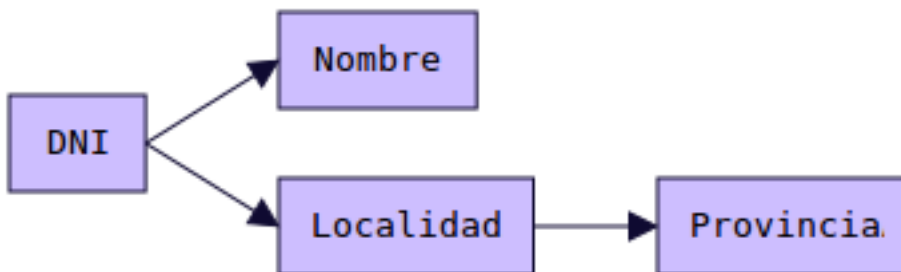
Una Relación esta en 3FN si y sólo si está en 2FN y no existen dependencias transitivas. Todas las dependencias funcionales deben ser respecto a la clave principal.

Ejemplo: Seguimos con el ejemplo anterior. Trabajaremos con la siguiente tabla:

Tabla 9: Alumnos

DNI	Nombre	Localidad	Provincia
11111111A	Eva	Écija	Sevilla
22222222B	Ana	Écija	Sevilla
33333333C	Susana	El Villar	Córdoba
44444444D	Juan	El Villar	Córdoba
55555555E	José	Écija	Sevilla

Las dependencias funcionales existentes son las siguientes. Como podemos observar existe una dependencia funcional transitiva: $DNI \rightarrow Localidad \rightarrow Provincia$



Para que la tabla esté en 3FN, no pueden existir dependencias funcionales transitivas. Para solucionar el problema deberemos crear una nueva tabla. El resultado es:

Tabla 10: Alumnos

DNI	Nombre	Localidad
11111111A	Eva	Écija
22222222B	Ana	Écija
33333333C	Susana	El Villar
44444444D	Juan	El Villar
55555555E	José	Écija

Tabla 11: Localidades

Localidad	Provincia
Écija	Sevilla
El Villar	Córdoba

RESULTADO FINAL

Tabla 12: Alumnos

DNI	Nombre	Localidad
11111111A	Eva	Écija
22222222B	Ana	Écija
33333333C	Susana	El Villar
44444444D	Juan	El Villar
55555555E	José	Écija

Tabla 13: Localidades

Localidad	Provincia
Écija	Sevilla
El Villar	Córdoba

Tabla 14: Teléfonos

DNI	Teléfono
11111111A	660111222
22222222B	660222333
22222222B	660333444
22222222B	660444555
55555555E	661000111
55555555E	661000222

Tabla 15: Matrículas

DNI	Curso	FechaMatrícula
11111111A	1ESO-A	01-Julio-2016
22222222B	1ESO-A	09-Julio-2016
33333333C	1ESO-B	11-Julio-2016
44444444D	2ESO-A	05-Julio-2016
55555555E	2ESO-A	02-Julio-2016

Tabla 16: Cursos

Curso	Tutor
1ESO-A	Isabel
1ESO-B	Roberto
2ESO-A	Federico



2.6.4 Forma Normal de Boyce-Codd: FNBC

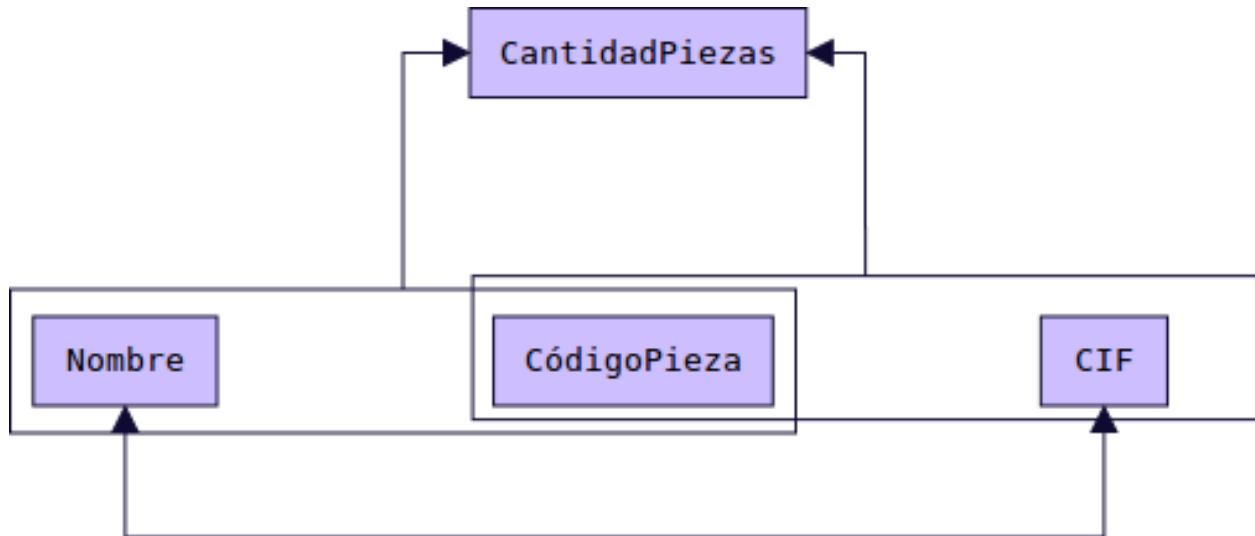
Una Relación esta en FNBC si está en 3FN y no existe solapamiento de claves candidatas. Solamente hemos de tener en cuenta esta forma normal cuando tenemos varias claves candidatas compuestas y existe solapamiento entre ellas. Pocas veces se da este caso.

Ejemplo: Tenemos una tabla con información de proveedores, códigos de piezas y cantidades de esa pieza que proporcionan los proveedores. Cada proveedor tiene un nombre único. Los datos son:

Tabla 17: Suministros

CIF	Nombre	CódigoPieza	CantidadPiezas
S-11111111A	Ferroman	1	10
B-22222222B	Ferrotex	1	7
M-33333333C	Ferropet	3	4
S-11111111A	Ferroman	2	20
S-11111111A	Ferroman	3	15
B-22222222B	Ferrotex	2	8
B-22222222B	Ferrotex	3	4

El gráfico de dependencias funcionales es el siguiente:



El atributo CantidadPiezas tiene dependencia funcional de dos claves candidatas compuestas, que son:

- (NombreProveedor, CódigoPieza)
- (CIFProveedor, CódigoPieza)

Existe también una dependencia funcional en doble sentido (que no nos afecta): NombreProveedor $\leftrightarrow$ CIFProveedor.

Para esta tabla existe un solapamiento de 2 claves candidatas compuestas. Para evitar el solapamiento de claves candidatas dividimos la tabla. La solución es:

Tabla 18: Proveedores

CIF	Nombre
S-11111111A	Ferroman
B-22222222B	Ferrotex
M-33333333C	Ferropet

Tabla 19: Suministros

CIF	CódigoPieza	CantidadPiezas
S-11111111A	1	10
B-22222222B	1	7
M-33333333C	3	4
S-11111111A	2	20
S-11111111A	3	15
B-22222222B	2	8
B-22222222B	3	4

2.6.5 Cuarta Forma Normal: 4FN

Una Relación está en 4FN si y sólo si está en 3FN (o FNBC) y las únicas dependencias multivaluadas son aquellas que dependen de las claves candidatas.

Ejemplo: Tenemos una tabla con la información de nuestros alumnos y alumnas y las asignaturas que cursan así como los deportes que practican.

Tabla 20: Alumnado

Estudiante	Asignatura	Deporte
11111111A	Matemáticas, Lengua	Baloncesto
22222222B	Matemáticas	Fútbol, Natación

Tabla 21: Alumnado

Estudiante	Asignatura	Deporte
11111111A	Matemáticas	Natación
11111111A	Matemáticas	Baloncesto
11111111A	Lengua	Natación
11111111A	Lengua	Baloncesto
22222222B	Matemáticas	Fútbol
22222222B	Matemáticas	Natación

Para normalizar esta tabla, debemos darnos cuenta que la oferta de asignaturas está compuesta por un conjunto de valores limitado. Igual sucede con los deportes. Por tanto existen dos dependencias multivaluadas:

- Estudiante $\rightarrow \rightarrow$ Asignatura
- Estudiante $\rightarrow \rightarrow$ Deporte

Por otro lado no existe ninguna dependencia entre la asignatura cursada y el deporte practicado. Para normalizar a 4FN creamos 2 tablas:

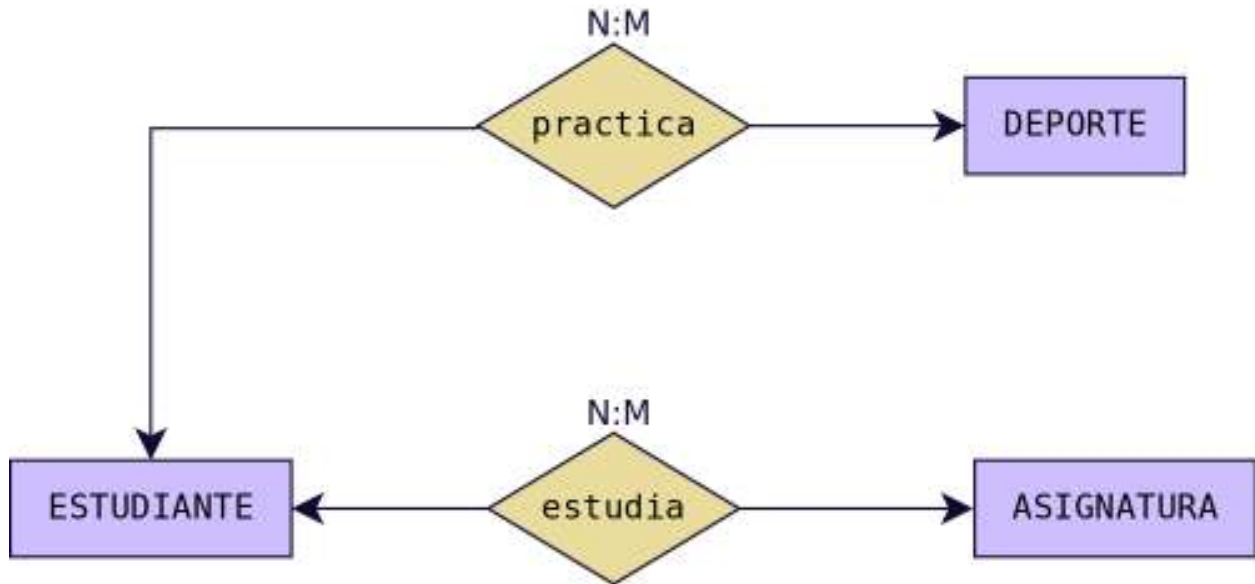
Tabla 22: EstudiaAsignatura

Estudiante	Asignatura
11111111A	Matemáticas
11111111A	Lengua
22222222B	Matemáticas

Tabla 23: PracticaDeporte

Estudiante	Deporte
11111111A	Natación
11111111A	Baloncesto
22222222B	Fútbol
22222222B	Natación

Diagrama E/R equivalente



2.6.6 Quinta Forma Normal: 5FN

La quinta forma normal (5FN), es una generalización de la anterior. También conocida como forma normal de proyección-uni3n (PJ/NF). Una tabla se dice que est1 en 5NF si y s3lo si est1 en 4NF y cada dependencia de uni3n (join) en ella es implicada por las claves candidatas. Ejemplo: Tenemos una tabla con varios proveedores que nos proporcionan piezas para distintos proyectos. Asumimos que un Proveedor suministra ciertas Piezas en particular, un Proyecto usa ciertas Piezas, y un Proyecto es suplido por ciertos Proveedores, entonces tenemos las siguientes dependencias multivaluadas:

- Proveedor $\twoheadrightarrow$ Pieza
- Pieza $\twoheadrightarrow$ Proyecto
- Proyecto $\twoheadrightarrow$ Proveedor

Se puede observar como se produce un ciclo:

- Proveedor $\twoheadrightarrow$ Pieza $\twoheadrightarrow$ Proyecto $\twoheadrightarrow$ Proveedor (nuevamente)

Tabla 24: Suministros

Proveedor	Pieza	Proyecto
E1, E4, E6	PI3, PI6	PR2, PR4
E2, E5	PI1, PI2	PR1, PR3
E3, E7	PI4, PI5	PR5, PR6

Tabla 25: Suministros

Proveedor	Pieza	Proyecto
E1	PI3	PR2
E1	PI3	PR4
E1	PI6	PR2
E1	PI6	PR4
E4	PI3	PR2
E4	PI3	PR4
E4	PI6	PR2
E4	PI6	PR4
E6	PI3	PR2
E6	PI3	PR4
E6	PI6	PR2
E6	PI6	PR4
E2	PI1	PR1
E2	PI1	PR3
E2	PI2	PR1
E2	PI2	PR3
E5	PI1	PR1
E5	PI1	PR3
E5	PI2	PR1
E5	PI2	PR3
E3	PI4	PR5
E3	PI4	PR6
E3	PI5	PR5
E3	PI5	PR6
E7	PI4	PR5
E7	PI4	PR6
E7	PI5	PR5
E7	PI5	PR6

Descomponemos la tabla en 3 tabla nuevas: Proveedor-Pieza, Pieza-Proyecto, Proyecto-Proveedor.

Tabla 26: Proveedor-Pieza

Proveedor	Pieza
E1	PI3
E1	PI6
E4	PI3
E4	PI6
E6	PI3
E6	PI6
E2	PI1
E2	PI2
E5	PI1
E5	PI2
E3	PI4
E3	PI5
E7	PI4
E7	PI5

Tabla 27: Pieza-Proyecto

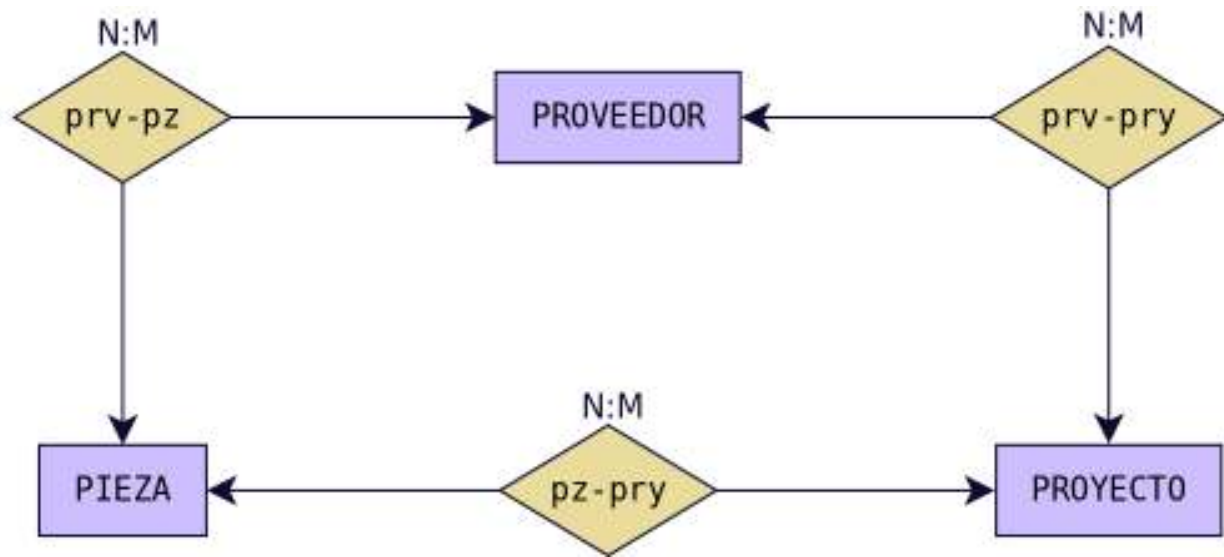
Pieza	Proyecto
PI3	PR2
PI3	PR4
PI6	PR2
PI6	PR4
PI1	PR1
PI1	PR3
PI2	PR1
PI2	PR3
PI4	PR5
PI4	PR6
PI5	PR5
PI5	PR6

Tabla 28: Proyecto-Proveedor

Proyecto	Proveedor
PR2	E1
PR4	E1
PR2	E4
PR4	E4
PR2	E6
PR4	E6
PR1	E2
PR3	E2
PR1	E5
PR3	E5
PR5	E3
PR6	E3
PR5	E7
PR6	E7

El producto natural de estas 3 tablas nos da la tabla original. Proveedor-Pieza |x| Pieza-Proyecto |x| Proyecto-Proveedor
= Suministros

Diagrama E/R equivalente



2.7 ACTIVIDADES PROPUESTAS

2.7.1 Test

MODELO ENTIDAD-RELACIÓN

Para cada una de las siguientes cuestiones elige razonadamente cada una de las respuestas correctas.

1. Un modelo conceptual de datos:
 - a. Define una serie de símbolos para describir la realidad de la BD que se desea crear.
 - b. Es un modelo que describe como se almacenan los datos a nivel físico.
 - c. Permite realizar una representación del mundo real.
2. El modelo Entidad/Relación:
 - a. Utiliza rombos para representar las entidades.
 - b. Utiliza círculos para representar las relaciones.
 - c. Cuenta con símbolos diferentes para representar las entidades fuertes y las débiles.
3. Las relaciones del modelo E/R...
 - a. Son objetos reales o abstractos de los que se desea guardar información en una BD.
 - b. Pueden ser fuerte o débiles.
 - c. Pueden ser de dependencia en identificación o en existencia.
4. Los atributos del modelo E/R...
 - a. Que identifican unívocamente cada ocurrencia de la entidad se llaman Clave principal.
 - b. Aparecen sólo en las entidades.
 - c. Aparecen sólo en las relaciones.
5. La cardinalidad...

- 1:1 es una cardinalidad binaria que significa que a cada ocurrencia de una entidad le corresponde una sola ocurrencia de la otra entidad.
- En el caso de relaciones entre tres entidades pueden ser de los tipos: 1:1, 1:N o N:M.
- Toma las participaciones máximas de cada entidad.

2.7.2 Cuestiones

MODELO ENTIDAD-RELACIÓN

1. Define brevemente los siguientes conceptos:

- Entidad.
- Relación.
- Atributo de una entidad.
- Identificador de una entidad.
- Atributo de una relación.
- Rol de una entidad en una relación.
- Participación de una entidad en una relación.
- Cardinalidad de una relación.

2. Indica cuáles son los dos tipos posibles de entidades y explica brevemente cada una de ellas.

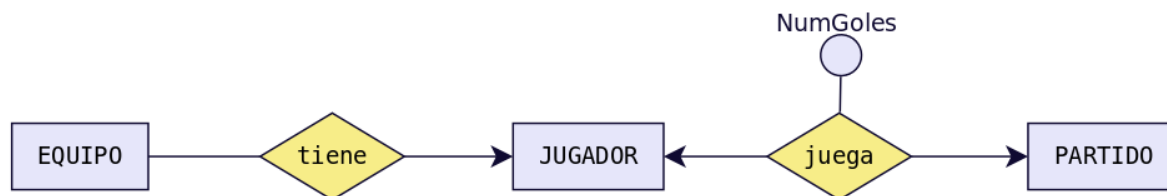
3. Clasifica los distintos tipos de relaciones existentes entre dos entidades según su cardinalidad y pon un ejemplo de cada una de ellas distinto de los vistos en el tema.

4. Clasifica los distintos tipos de relaciones de dependencia existentes y pon un ejemplo de cada una de ellas distinto de los vistos en el tema.

5. Explica brevemente la Restricción de exclusividad entre dos tipos de relaciones R1 y R2 respecto a la entidad E1. Pon un ejemplo distinto del visto en el tema.

6. Explica brevemente la Restricción de inclusión entre dos tipos de relaciones R1 y R2. Pon un ejemplo distinto del visto en el tema.

7. Dado el siguiente esquema:



- Indica cuáles son las entidades del modelo, diferenciado entre entidades fuertes y débiles, si las hubiera.

- b. Señala las relaciones e indica cual es la cardinalidad de cada una. Trata de indicar también la participación de cada entidad en las relaciones así como su rol.
- c. Señala si hay alguna relación de dependencia o reflexiva.
- d. Trata de escribir atributos lógicos para cada una de las entidades e indica en cada caso cual podría ser el identificador.
- e. ¿Qué significado tiene el atributo “NºGoles”? ¿Por qué está en la relación en lugar de estar en JUGADOR o en PARTIDO?

8. Obtén el diagrama E/R con las tres entidades siguientes:

- **ALUMNO** (Núm_Matricula, Nombre, FechaNacimiento, Teléfono)
- **ASIGNATURA** (Código_asignatura, Nombre)
- **PROFESOR** (Id_P, NIF_P, Nombre, Especialidad, Teléfono)

Teniendo en cuenta:

- Un alumno puede estar matriculado de una o varias asignaturas.
- Además puede estar matriculado en la misma asignatura más de un curso escolar (si repite).
- Se quiere saber el curso escolar en el que cada alumno está matriculado de cada asignatura.
- En una asignatura habrá como mínimo 10 y como máximo 25 alumnos.
- Una asignatura es impartida por un único profesor.
- Un profesor podrá impartir varias asignaturas.

9. Obtén el diagrama E/R con las cuatro entidades siguientes:

- **REGIÓN** (Nombre_Región)
- **PROVINCIA** (CódigoProvincia, Nombre_provincia)
- **LOCALIDAD** (Código_localidad, Nombre)
- **EMPLEADO** (Id_E, DNI_E, Nombre, Teléfono, Salario)

Se quiere guardar información de la localidad donde ha nacido cada uno de los empleados teniendo en cuenta que:

- Un empleado ha nacido en una sola localidad.
- Cada localidad pertenece a una única provincia.
- Cada provincia pertenece a una única región del país.

10. Obtén el diagrama E/R con las dos entidades siguientes:

- **EMPLEADO** (Id_E, DNI_E, Nombre, Teléfono, Salario)
- **DEPARTAMENTO** (Código_D, Nombre, Localización)

Teniendo en cuenta:

- Un empleado pertenece a un único departamento y en un departamento puede haber varios empleados. Pero sólo uno será el jefe del departamento.
- Un empleado podrá ser jefe o no. Si no es jefe, su jefe será el del departamento al que pertenece.

11. Obtén el diagrama E/R para el siguiente supuesto.

Una empresa dedicada a la instalación de dormitorios juveniles a medida quiere realizar una base de datos donde se reflejen las ventas y montajes, para lo cual se tiene en cuenta:

- Cada modelo de dormitorio lo debe montar, al menos, dos montadores.
- El mismo montador puede montar varios modelos de dormitorios.
- De cada modelo dormitorio nos interesa conocer su código de modelo.
- El mismo montador puede montar el mismo modelo en diferentes fechas. Nos interesa conocer la fecha en la que realiza cada montaje.
- De un montador nos interesa su NIF, nombre, dirección, teléfono de contacto y el número de dormitorios que ha montado de cada modelo.
- Cada modelo de dormitorio puede ser comprado por uno o varios clientes y el mismo cliente podrá comprar uno o varios dormitorios. De un cliente nos interesa su NIF, nombre, dirección, teléfono y fecha de compra de cada modelo.

12. Se desea diseñar una base de datos sobre la información de las reservas de una empresa dedicada al alquiler de automóviles teniendo en cuenta que:

- Un determinado cliente puede tener en un momento dado hechas varias reservas.
- De cada cliente se desea almacenar su DNI, nombre, dirección y teléfono.
- Además dos clientes se diferencian por un único código.
- De cada reserva es importante registrar su número de identificación, la fecha de inicio y final de la reserva, el precio total.
- De cada coche se requiere la matrícula, el modelo, el color y la marca. Cada coche tiene un precio de alquiler por hora.
- Además en una reserva se pueden incluir varios coches de alquiler. Queremos saber los coches que incluye cada reserva y los litros de gasolina en el depósito en el momento de realizar la reserva, pues se cobrarán aparte.
- Cada cliente puede ser avalado por otro cliente de la empresa.

13. Tenemos esta información sobre una cadena editorial:

- La editorial tiene varias sucursales, con su domicilio, teléfono y un código de sucursal.
- Cada sucursal tiene varios empleados, de los cuales tendremos sus datos personales, DNI y teléfono. Un empleado trabaja en una única sucursal.
- En cada sucursal se publican varias revistas, de las que almacenaremos su título, número de registro, periodicidad y tipo.
- La editorial tiene periodistas (que no trabajan en las sucursales) que pueden escribir artículos para varias revistas. Almacenaremos los mismos datos que para los empleados, añadiendo su especialidad.
- Para cada revista, almacenaremos información de cada número, que incluirá la fecha, número de páginas y el número de ejemplares vendidos.

14. La cadena de Video-Clubs Glob-Gusters ha decidido, para mejorar su servicio, emplear una base de datos para almacenar la información referente a las películas que ofrece en alquiler.

Esta información es la siguiente:

- Una película se caracteriza por su título, nacionalidad, productora y fecha. Puede haber varias películas con el mismo título pero rodadas en fechas distintas.
- En una película pueden participar varios actores (nombre, nacionalidad, sexo) algunos de ellos como actores principales.
- Una película está dirigida por un director (nombre, nacionalidad).
- De cada película se dispone de uno o varios ejemplares diferenciados por un número de ejemplar y caracterizados por su estado de conservación.
- Un ejemplar se puede encontrar alquilado a algún socio (DNI, nombre, dirección, teléfono) . Se desea almacenar la fecha de comienzo del alquiler y la de devolución.
- Un socio tiene que ser avalado por otro socio que responda de él en caso de tener problemas en el alquiler.

15. Diseñar un esquema E/R que recoja la organización de un sistema de información en el que se quiere tener los datos sobre municipios, viviendas y personas.

Cada persona sólo puede habitar una vivienda, pero puede ser propietaria de varias. También nos interesa la relación de las personas con su cabeza de familia.

16. Se desea diseñar una BD de una entidad bancaria que contenga información sobre los clientes, las cuentas, las sucursales y las transacciones producidas.

Construir el Modelo E/R teniendo en cuenta las siguientes restricciones:

- Una transacción viene determinada por un número de transacción (único para cada cuenta), la fecha y la cantidad.
- Un cliente puede tener muchas cuentas.
- Una cuenta puede ser de muchos clientes.
- Una cuenta sólo puede estar en una sucursal.

17. Una base de datos para una pequeña empresa debe contener información acerca de clientes, artículos y pedidos.

Hasta el momento se registran los siguientes datos en documentos varios:

- Para cada cliente: Número de cliente (único), Direcciones de envío (varias por cliente), Saldo, Límite de crédito, Descuento.
- Para cada artículo: Número de artículo (único), Fábricas que lo distribuyen, Existencias de ese artículo en cada fábrica, Descripción del artículo.
- Para cada pedido: Cada pedido se registrará en un documento impreso que tiene una cabecera y el cuerpo del pedido. - Para generar dicho informe se necesitará la siguiente información:
- La cabecera está formada por el número de cliente, dirección de envío y fecha del pedido.
- El cuerpo del pedido son varias líneas, en cada línea se especifican el número del artículo pedido y la cantidad.

- Además, se ha determinado que se debe almacenar la información de las fábricas. Sin embargo, dado el uso de distribuidores, se usará: Número de la fábrica (único) y Teléfono de contacto.
- Y se desean ver cuántos artículos (en total) provee la fábrica. También, por información estratégica, se podría incluir información de fábricas alternativas respecto de las que ya fabrican artículos para esta empresa.

18. Se pide hacer el diagrama ER para la base de datos que represente esta información. Le contratan para hacer una BD que permita apoyar la gestión de un sistema de ventas.

La empresa necesita llevar un control de proveedores, clientes, productos y ventas. Un proveedor tiene un código único, nombre, dirección, teléfono y página web. Un cliente también tiene un código único, nombre, dirección, pero puede tener varios teléfonos de contacto. La dirección se entiende por calle, número, comuna y ciudad.

Un producto tiene un id único, nombre, precio actual, stock y nombre del proveedor. Además se organizan en categorías, y cada producto va sólo en una categoría. Una categoría tiene id, nombre y descripción. Por razones de contabilidad, se debe registrar la información de cada venta con un id, fecha, cliente, descuento y monto final. Además se debe guardar el precio al momento de la venta, la cantidad vendida y el monto total por el producto.

MODELO ENTIDAD-RELACIÓN EXTENDIDO

19. El departamento de formación de una empresa desea construir una base de datos para planificar y gestionar la formación de sus empleados.

La empresa organiza cursos internos de formación de los que se desea conocer el código de curso, el nombre, una descripción, el número de horas de duración y el coste del curso.

Un curso puede tener como prerequisite haber realizado otro u otros previamente, y a su vez, la realización de un curso puede ser prerequisite de otros. Un curso que es un prerequisite de otro puede serlo de forma obligatoria o sólo recomendable.

Un mismo curso tiene diferentes ediciones, es decir, se imparte en diferentes lugares, fechas y con diferentes horarios (intensivo, de mañana o de tarde). En una misma fecha de inicio sólo puede impartirse una edición de un mismo curso.

Los cursos se imparten por personal de la propia empresa.

De los empleados se desea almacenar su código de empleado, nombre y apellidos, dirección, teléfono, NIF (Número de Identificación Fiscal), fecha de nacimiento, nacionalidad, sexo, firma y salario, así como si está o no capacitado para impartir cursos.

Un mismo empleado puede ser docente en una edición de un curso y alumno en otra edición, pero nunca puede ser ambas cosas a la vez (en una misma edición de curso o lo imparte o lo recibe).

Realiza el Modelo Entidad/Relación

20. Una Empresa decide informatizar su gestión de nóminas. Del resultado del análisis realizado, se obtienen las siguientes informaciones:

- A cada empleado se le entregan múltiples nóminas a lo largo de su vida laboral en la empresa y al menos una mensualmente.
- A cada empleado se le asigna un número de empleado en el momento de su incorporación a la empresa, y éste es el número usado a efectos internos de identificación. Además, se registran el Número de Identificación Fiscal del empleado, nombre, número de hijos, porcentaje de retención para Hacienda, datos de cuenta corriente en la que se le ingresa el dinero (banco, sucursal y número de cuenta) y departamentos en los que trabaja.
- Un empleado puede trabajar en varios departamentos y en cada uno de ellos trabajará con un función distinta.
- De un departamento se mantiene el nombre y cada una de sus posibles sedes.

- Son datos propios de una nómina el ingreso total percibido por el empleado y el descuento total aplicado.
- La distinción entre dos nóminas se hará, además de mediante el número de identificación del empleado, mediante el ejercicio fiscal y número de mes al que pertenece y con un número de orden en el caso de varias nóminas recibidas el mismo mes.
- Cada nómina consta de varias líneas (al menos una de ingresos) y cada línea se identifica por un número de línea dentro de la correspondiente nómina.
- Una línea puede corresponder a un ingreso o a un descuento. En ambos casos, se recoge la cantidad que corresponde a la línea (en positivo si se trata de un ingreso o en negativo si se trata de un descuento); en el caso de los descuentos, se recoge la base sobre la cual se aplica y el porcentaje que se aplica para el cálculo de éstos.
- Toda línea de ingreso de una nómina responde a un único concepto retributivo.
- En un mismo justificante, puede haber varias líneas que respondan al mismo concepto retributivo.
- De los conceptos retributivos se mantiene un código y una descripción.

Realiza el Modelo Entidad/Relación

21. La ministra de Medio Ambiente ha decidido crear un sistema de información sobre los parques naturales gestionados por cada comunidad autónoma.

Después de realizar un detallado análisis, se ha llegado a las siguientes conclusiones:

- Una comunidad autónoma (CA) puede tener varios parques naturales. En toda comunidad autónoma existe uno y sólo un organismo responsable de los parques. Un parque puede estar compartido por más de una comunidad.
- Un parque natural se identifica por un nombre, fue declarado en una fecha, se compone de varias áreas identificadas por un nombre y caracterizadas por una determinada extensión. Por motivos de eficiencia se desea favorecer las consultas referentes al número de parques existentes en cada comunidad y la superficie total declarada parque natural en cada CA.
- En cada área forzosamente residen especies que pueden ser de tres tipos: vegetales, animales y minerales. Cada especie tiene una denominación científica, una denominación vulgar y un número inventariado de individuos por área. De las especies vegetales se desea saber si tienen floración y en qué periodo se produce ésta; de las animales se desea saber su tipo de alimentación (herbívora, carnívora u omnívora) y sus periodos de celo; de las minerales se desea saber si se trata de cristales o de rocas.
- Además, interesa registrar qué especies sirven de alimento a otras especies, teniendo en cuenta que ninguna especie mineral se considera alimento de cualquier otra especie y que una especie vegetal no se alimenta de ninguna otra especie.
- Del personal del parque se guarda el DNI, número de seguridad social, nombre, dirección, teléfonos (domicilio, móvil) y sueldo. Se distinguen los siguientes tipos de personal:
 - Personal de gestión: registra los datos de los visitantes del parque y están destinados en una entrada del parque (las entradas se identifican por un número).
 - Personal de vigilancia: vigila un área determinada del parque que recorre en un vehículo (tipo y matrícula).
 - Personal investigador: Tiene una titulación que ha de recogerse y pueden realizar (incluso conjuntamente) proyectos de investigación sobre una determinada especie. Un proyecto de investigación tiene un presupuesto y un periodo de realización.
 - Personal de conservación: mantiene y conserva un área determinada del parque. Cada uno lo realiza en una especialidad determinada (limpieza, caninos...).
- Un visitante (DNI, nombre, domicilio y profesión) debe alojarse dentro de los alojamientos de que dispone el parque; éstos tienen una capacidad limitada y tienen una determinada categoría.

- Los alojamientos organizan excursiones al parque, en vehículo o a pie, en determinados días de la semana y a una hora determinada. A estas excursiones puede acudir cualquier visitante del parque.
- Por comodidad, suponemos que un visitante tiene, obligatoriamente, que alojarse en el parque. Suponemos también, que cada vigilante tiene su vehículo propio que sólo utiliza él.

MODELO RELACIONAL

22. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 8.

23. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 9.

24. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 10.

25. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 11.

26. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 12.

27. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 13.

28. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 14.

29. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 15.

30. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 16.

31. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 17.

32. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 18.

33. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 19.

34. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 20.

35. Obtén el diagrama Relacional a partir el E/R obtenido en la cuestión 21.

2.7.3 Prácticas

MODELO ENTIDAD-RELACIÓN

PRÁCTICA 1

PLANTEAMIENTO

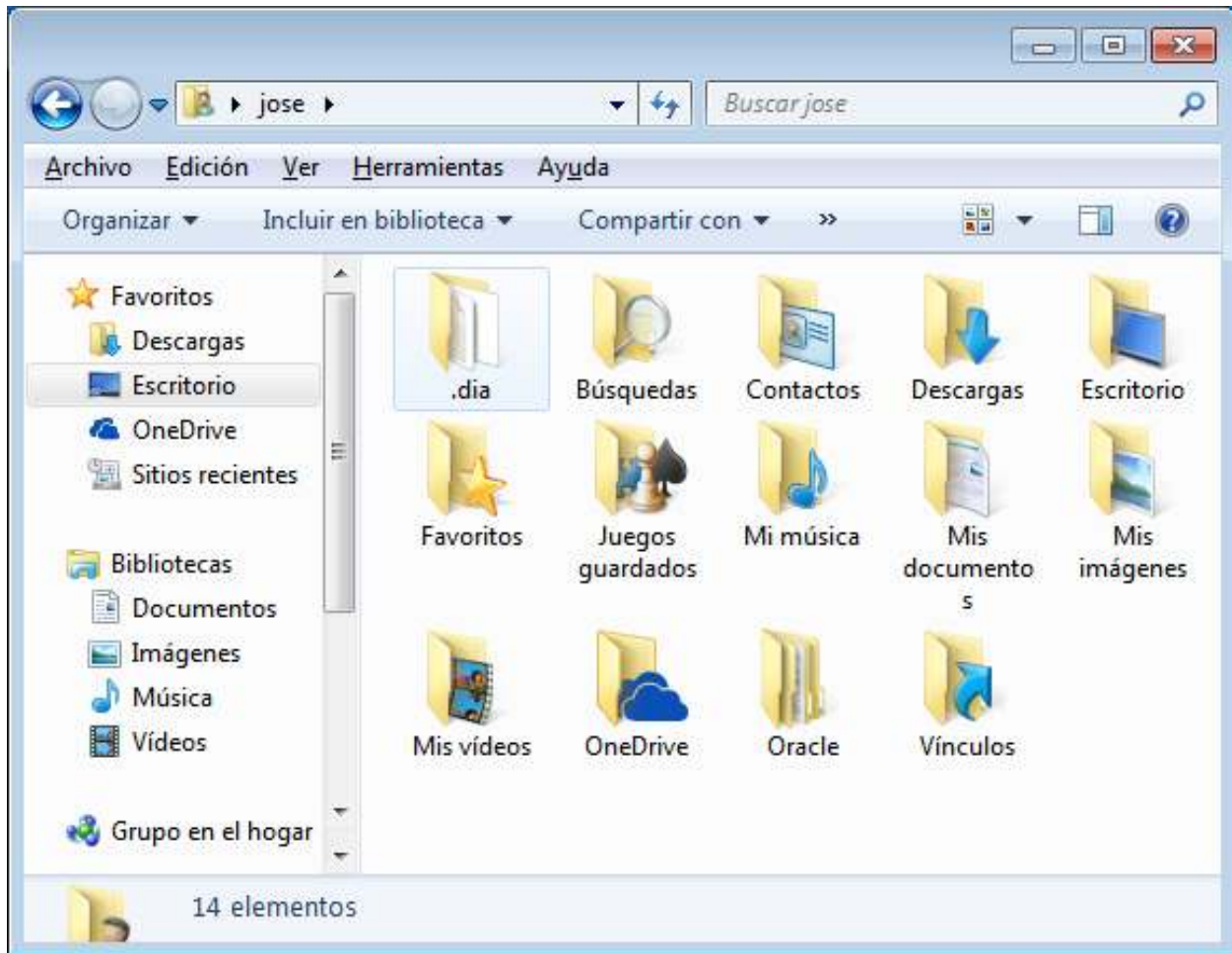
OBJETIVO: Aprender el uso básico del programa Dia, que utilizaremos para para realizar diagramas. En concreto diagramas ER extendidos y relacionales.

ENUNCIADO: Instala el programa Dia y la hoja de símbolos EER.zip para los símbolos utilizados en diagramas Entidad-Relación extendidos. Con la ayuda del profesor, examina la forma de uso de dicho programa.

Para ello deberás seguir los siguientes pasos:

1. Descarga de la plataforma Moodle el programa Dia.

2. Procede a su instalación.
3. Ejecútalo por primera vez para que se cree una subcarpeta .dia en tu directorio personal.

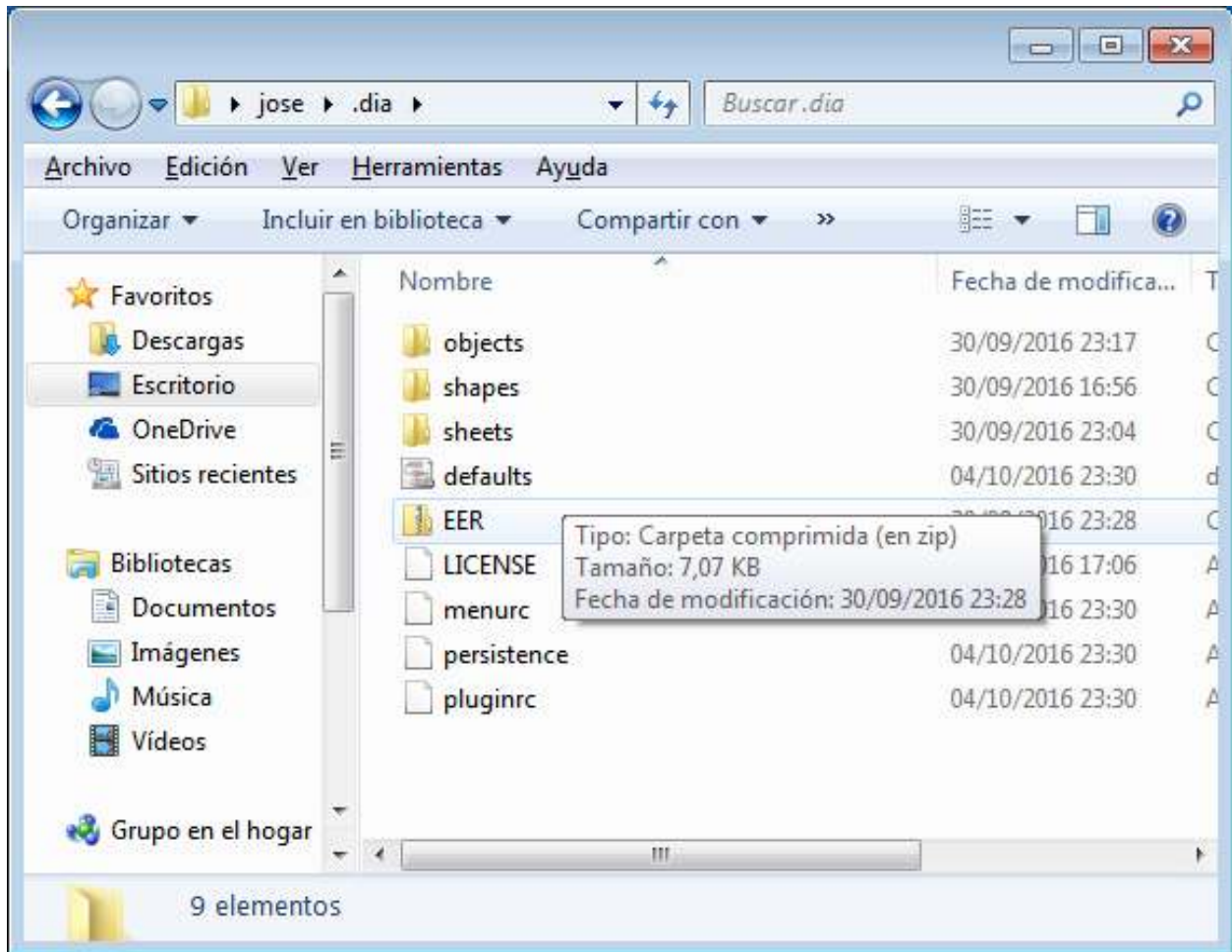


4. Descarga de la plataforma Moodle el archivo EER.zip que contiene los símbolos necesarios para diagramas E/R extendidos (Extended Entity-Relationship).

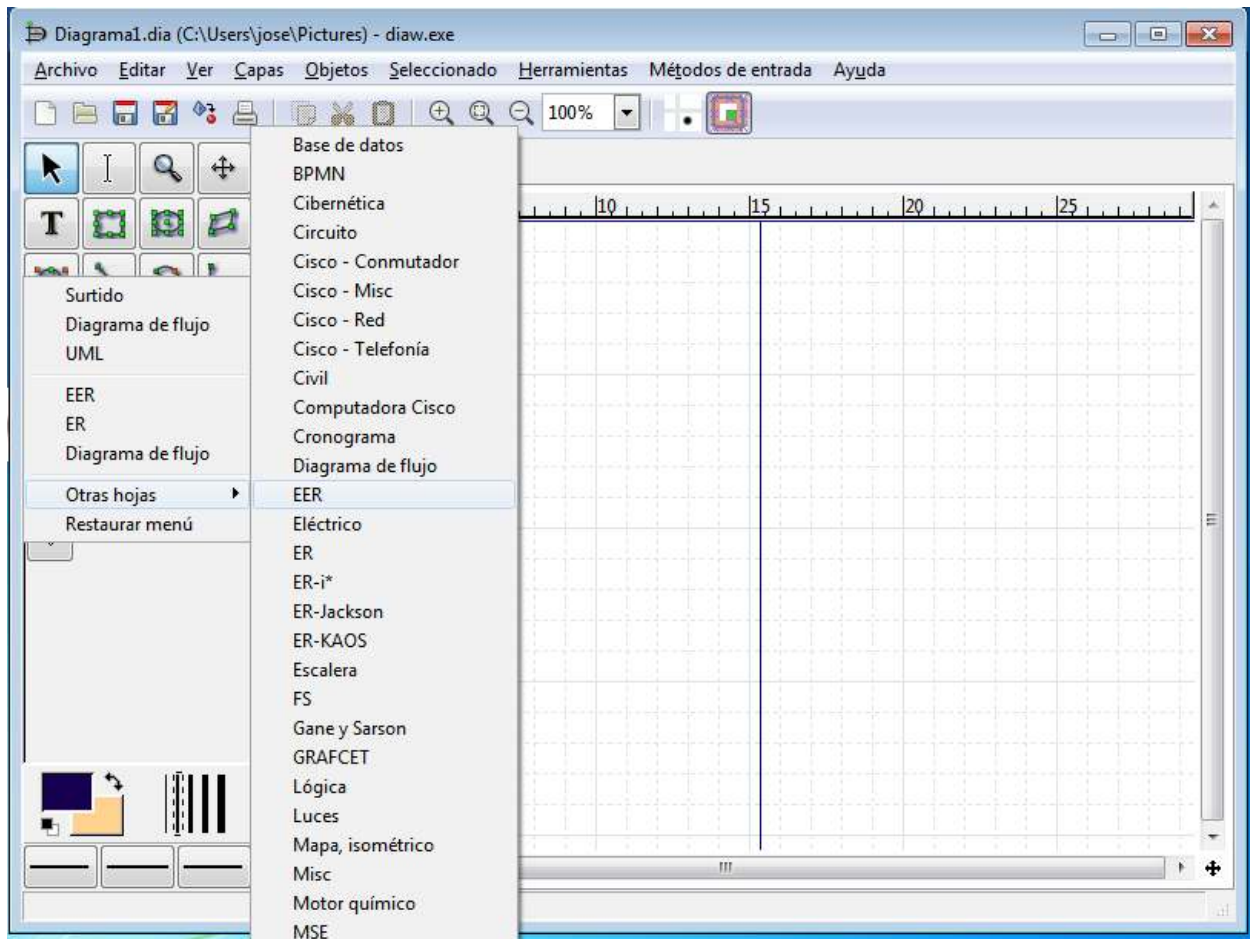
También puede descargarse desde el siguiente enlace:

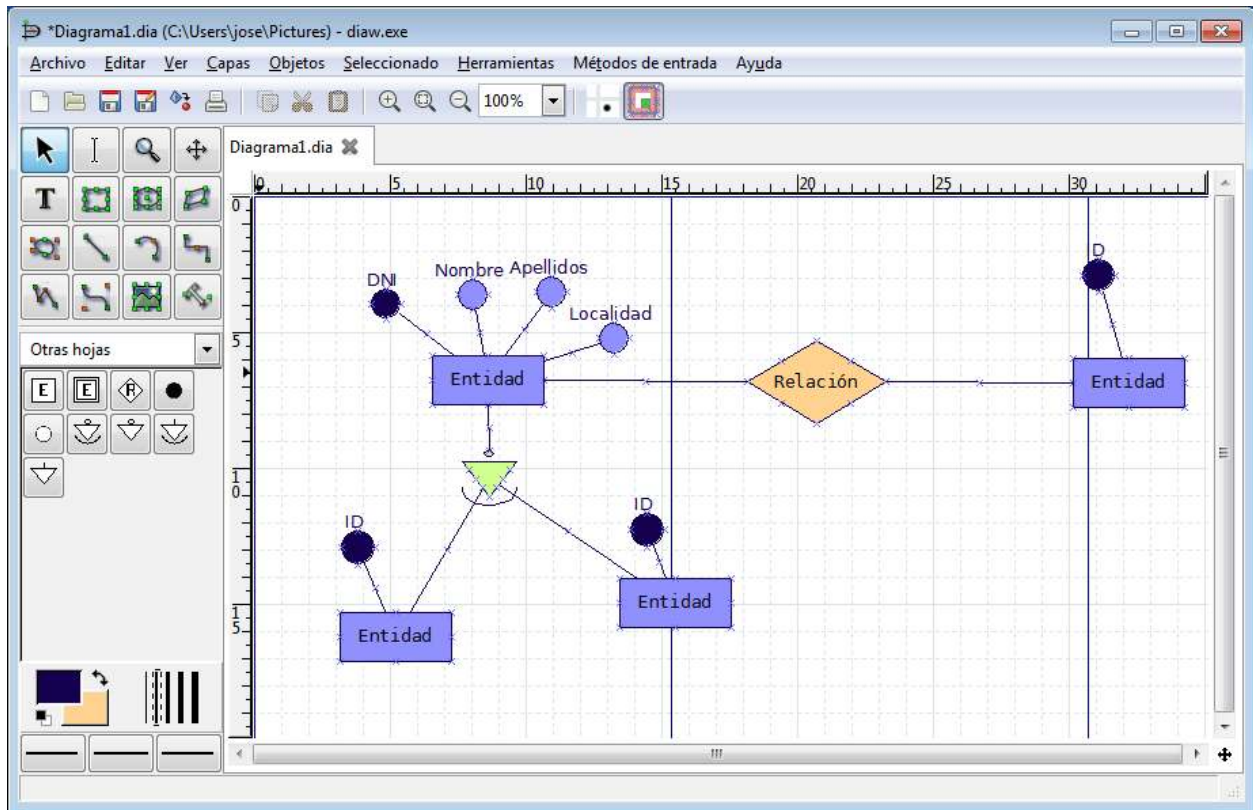
<https://github.com/jamj2000/GestionBasesDatos/blob/master/Tema2/EER.zip>

5. Copia este archivo a la subcarpeta .dia y descomprímelo ahí.



6. Se generará un nuevo archivo LICENSE y dos carpetas: shapes y sheets.
7. Reinicia el programa Dia.
8. Debajo de las herramientas, selecciona Otras hojas → EER.





PRÁCTICA 2

PLANTEAMIENTO

OBJETIVO: Recordar todo lo visto en el tema 1 ahora que ya somos capaces de crear diagramas que modelen la realidad de nuestros problemas.

ENUNCIADO: Responde a las siguientes cuestiones.

Con ayuda de el profesor y lo visto sobre el tema referente al modelo relacional deberás:

- Realizar el paso a tablas de la cuestión 6.2.8.
 - ALUMNO (Núm_Matrícula, Nombre, FechaNacimiento, Teléfono)
 - ASIGNATURA (Código_asignatura, Nombre)
 - PROFESOR (Id_P, NIF_P, Nombre, Especialidad, Teléfono)
- Crea la BD que resulta en Microsoft ACCESS o LibreOffice BASE eligiendo los tipos de datos y las restricciones.
- Introduce 7 registros en la tabla ASIGNATURAS, 4 en la tabla PROFESORES y 15 en la tabla ALUMNOS. Además resultará una tabla MATRÍCULAS que deberás completar con el curso escolar en que cada alumno ha estado matriculado de cada asignatura. Asígnalos como estimes más oportuno. Recuerda que en cada asignatura habrá un mínimo de 10 alumnos.
- En la tabla PROFESORES mueva la columna TELEFONO a la izquierda de la columna ESPECIALIDAD. Pruebe otros movimientos.

- e) Oculte las columnas Fecha_nac y Tlfno de la tabla ALUMNOS. Vuelva a mostrarlas. Pruebe otras.
- f) Diseñar una consulta del tipo Eliminación capaz de eliminar de la tabla ALUMNOS solo aquellos registros comprendidos entre dos fechasNac límite que nos deberá preguntar cada vez que ejecutemos la consulta (Parámetros).
- g) Crea una nueva consulta en la que muestres el no de matrícula, el nombre y la asignatura en la que está o ha estado matriculado cada alumno, incluyendo el curso de la matrícula.
- h) Crea un formulario para la consulta que hemos creado en el punto anterior. El formulario deberá ser de Tipo Tabular y con todos los campos de la consulta.
- i) Crea un informe para la consulta anterior. El informe será de tipo tabular con todos los campos de la consulta y deberá estar ordenado por NoMatrícula.
- j) Modifica el aspecto del titulo del formulario añadiendo colores, bordes y cambiando el tipo de letra.

PRÁCTICA 3

PLANTEAMIENTO

OBJETIVO: Recordar todo lo visto en el tema 1 ahora que ya somos capaces de crear diagramas que modelen la realidad de nuestro problemas.

ENUNCIADO: Responde a las siguientes cuestiones.

Con ayuda del profesor deberás:

- a) Realizar el paso a tablas de la cuestión 6.2.10.
- b) Crea la BD que resulta en Microsoft ACCESS o LibreOffice BASE eligiendo los tipos de datos y las restricciones.
- c) Introduce registros en cada una de las tablas.
- d) Inventa cinco consultas y ejecútalas.
- e) Para una de las consultas anteriores Crea un formulario de tipo Tabular y modifica un poco su aspecto.
- f) Para la misma consulta que hayas elegido en el apartado anterior, crea un informe de Tipo Tabular y con todos los campos de la consulta.

MODELO ENTIDAD-RELACIÓN EXTENDIDO

PRÁCTICA 4

PLANTEAMIENTO

OBJETIVO: Recordar todo lo visto previamente y ampliarlo con lo nuevo aprendido en este tema.

ENUNCIADO: Resuelve los apartados siguientes.

- a) Realizar el modelo Entidad-Relación para modelar la situación real siguiente:
 - Queremos crear una base de datos para una empresa que fabrica y distribuye electrodomésticos. Debe contener información acerca de los departamentos, los empleados, los artículos, los clientes y los pedidos.

- De los departamentos queremos saber su código de identificación y el presupuesto medio con el que cuenta. Dicho presupuesto medio no podrá superar nunca los 60.000 €. Los departamentos se agrupan en sectores: Financiero, Productivo, Recursos Humanos y Ventas. De los departamentos financieros queremos saber también y su dirección y la entidad bancaria con la que trabajan. De los departamentos del sector productivo queremos conocer los artículos que fabrican.
 - De los empleados guardaremos su NIF, nombre, dirección, fecha de nacimiento y departamento en el que trabajan. Cada empleado trabaja en un único departamento.
 - De cada artículo: Número de artículo (único), nombre, Departamento que lo fabrica y existencias de ese artículo en cada departamento.
 - Para cada cliente: Número de cliente (único), Direcciones de envío (varias por cliente), Saldo, Límite de crédito (depende del cliente, pero en ningún caso debe superar los 18.000 €), Descuento.
 - Para cada pedido: número del pedido (único para cada cliente), dirección de envío y fecha del pedido.
 - Además queremos saber el número de artículos de cada tipo que incluye cada pedido.
- b) Con ayuda de la profesora, obtendrás el modelo relacional que aprenderás a realizar un poco más adelante.
- c) Crea la BD que resulta en Microsoft ACCESS o LibreOffice BASE eligiendo los tipos de datos y las restricciones.
- d) Introduce registros en cada una de las tablas.
- e) Inventa cinco consultas y ejecútalas.
- f) Para una de las consultas anteriores Crea un formulario de tipo Tabular y modifica un poco su aspecto.
- g) Para la misma consulta que hayas elegido en el apartado anterior, crea un informe de Tipo Tabular y con todos los campos de la consulta.

MODELO RELACIONAL

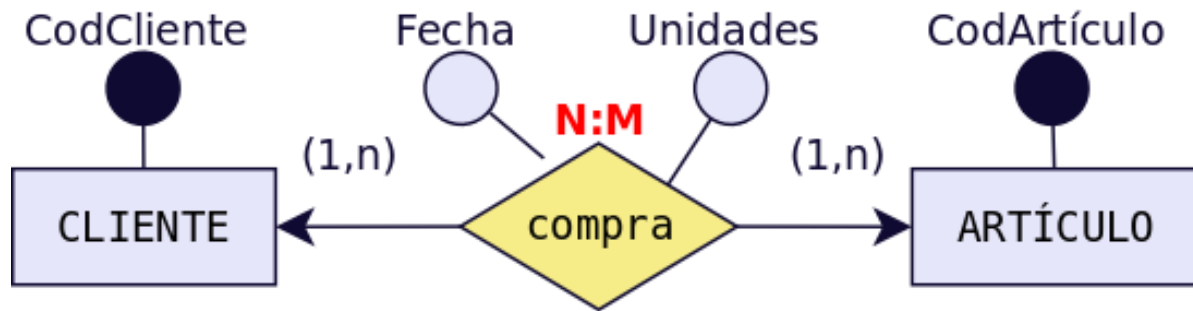
PRÁCTICA 5

PLANTEAMIENTO

OBJETIVO: Recordar todo lo visto y ampliarlo con lo nuevo aprendido.

ENUNCIADO: Resuelve los apartados siguientes.

A continuación mostramos un modelo E/R (hemos simplificado el número de atributos) del que se ha obtenido el correspondiente esquema relacional.



- a) Crea la BD en un SGBD doméstico (Microsoft ACCESS o LibreOffice BASE) teniendo en cuenta la siguiente información adicional:



Tabla 29: CLIENTE

CAMPO	TIPO	TAMAÑO	PREDETERMINADO	VALIDACIÓN
Código Cliente	Autonumérico			
Nombre	Texto	50		No vacío
Apellidos	Texto	50		
Empresa	Texto	50		
Puesto	Texto	50		
Dirección	Texto	50		
Población	Texto	25	Écija	
CP	Texto	5	41400	
Provincia	Texto	25	Sevilla	
Teléfono	Texto	9		
Fecha_Nacimiento	Fecha/hora			

Tabla 30: ARTÍCULO

CAMPO	TIPO	PROPIEDADES	PREDETERMINADO	VALIDACIÓN
Código Artículo	Autonumérico			
Nombre	Texto			No vacío
Descripción	Texto			No vacío
Precio/unidad	Moneda	No negativo		No vacío
Unidades en stock	Número	[0,100]		
Stock de Seguridad	Número	No inferior a 2	2	
Imagen	Objeto OLE			

Tabla 31: COMPRA

CAMPO	TIPO	PROPIEDADES	PREDETERMINADO	VALIDACIÓN
Código Cliente	Numérico	Se seleccionarán desde la tabla Cliente		
Código Artículo	Numérico	Se elegirán de la tabla Artículo		
Fecha	Fecha/hora		Fecha_Actual	
Unidades	Numérico	No negativo		No inferior a 1

b) Introduce los datos siguientes en la BD.

Tabla 32: CLIENTE

Cod_Cli	Nombre	Apellidos	Empresa	Puesto	Dirección	Población	CP	Provincia	Teléfono	Fecha_nac
1	José	Fernández Ruiz	Estudio Cero	Gerente	Cervantes, 13	Écija	41400	Sevilla	656789043	3/06/1968
2	Luis	Fernández Chacón	Beep	Dependiente	Aurora, 4	Écija	41400	Sevilla	675894562	4/05/1982
3	Antonio	Ruiz Gómez	Comar	Dependiente	Osuna, 23	Écija	41400	Sevilla	654345544	06/08/1989
4	Andrea	Romero Vázquez	Estudio Cero	Dependiente	Cervantes, 25	Écija	41400	Sevilla	646765652	23/11/1974
5	José	Pérez Pérez	Beep	Gerente	Córdoba, 10	Écija	41400	Sevilla	645345543	30/04/1978

Tabla 33: ARTÍCULO

Cod_Art	Nombre	Descripción	Pre-cio/Unidad	Unidades en stock	Stock Seg	Imagen
1	NETGEAR switch pro-safe	Switch 8 puertos GigabitEthernet	125 €	3	2	
2	Switch SRW224G4-EU de Linksys	CISCO switch 24 puertos 10/100	202,43 €	2	2	
3	Switch D-link	D-Link smart switch 16 puertos	149,90 €	7	4	
4	Switch D-link	D-Link smart switch 48 puertos	489,00 €	4	2	

Tabla 34: COMPRA

Cod_Cli	Cod_Art	Fecha	Unidad
1	1	13/10/2010	2
1	2	13/10/2010	1
2	3	15/10/2010	1
2	4	15/10/2010	1
3	1	15/10/2010	2
4	2	15/10/2010	1
5	3	15/10/2010	3
1	4	16/10/2010	1
1	1	16/10/2010	2
2	2	17/10/2010	1
3	3	18/10/2010	4
4	4	19/10/2010	2
5	1	19/10/2010	1

- c) Diseña un formulario para introducir los datos de cada compra.
d) Diseña un informe donde se resuman los pedidos para cada cliente.

PEDIDOS DE CADA CLIENTE

Encabezado del informe

Encabezado del informe

<i>Código Cliente</i>	<i>Fecha</i>	<i>Código del Producto</i>	<i>Unidades</i>
CLIENTE 1			
	13/10/2010	1	2
	13/10/2010	2	2
	16/10/2010	4	1
	16/10/2010	1	1

SUMA

Resumir por código cliente

SUMA TOTAL:

Al pié del informe

Incluir fecha y página a pie de página

- e) Realiza las siguientes consultas de la BD.
- 1) Mostrar los nombres y apellidos de los clientes llamados José o Luis ordenados alfabéticamente por nombres.
 - 2) Obtener el nombre y el teléfono de los clientes cuya edad está comprendida entre 20 y 25 años ordenados por edad.
 - 3) Mostrar nombre y apellidos de los clientes que no tengan teléfono.
 - 4) Mostrar aquellos productos cuyo stock en almacén sea menor que cuatro.
 - 5) Mostrar el nombre, la descripción y la imagen de los productos que valgan menos de 200 €.

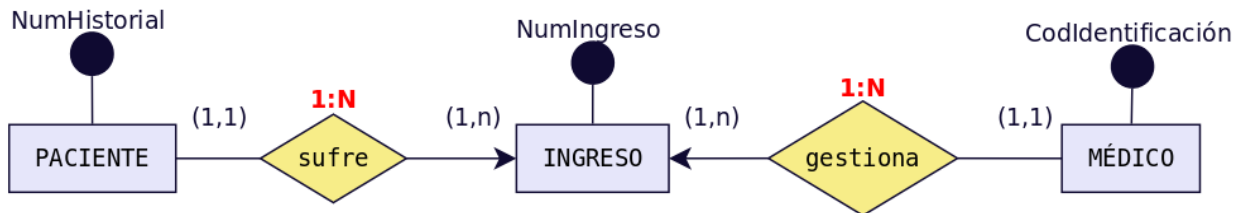
PRÁCTICA 6

PLANTEAMIENTO

OBJETIVO: Recordar todo lo visto hasta el momento.

ENUNCIADO: Resuelve los apartados siguientes.

Queremos un sistema de gestión de datos de un hospital. En él queremos guardar la información para cada uno de los ingresos hospitalarios indicando el paciente objeto del ingreso y el médico que autoriza el mismo. A continuación mostramos el modelo E/R que resulta del análisis de datos.



Las tablas que resultan para dicha BD tendrán los campos que se muestran a continuación:

Tabla 35: MÉDICOS

Campo	Tipo	Largo	Otros
Codigo identificación	Texto	4	Campo Clave
Nombre del Médico	Texto	15	
Apellidos del Médico	Texto	30	
Especialidad	Texto	25	
Fecha de ingreso	Fecha		
Cargo	Texto	25	
Número de Colegiado	Número		
Observaciones	Memo		

Tabla 36: PACIENTES

Campo	Tipo	Largo	Otros
N Seguridad Social	Texto	15	
Nombre	Texto	15	
Apellidos	Texto	30	
Domicilio	Texto	30	
Población	Texto	25	
Provincia	Texto	15	
Código Postal	Texto	5	
Teléfono	Texto	12	
Número de Historial	Texto	9	Campo Clave
Sexo	Texto	1	Regla de validación: «H» o «M»

Tabla 37: INGRESOS

Campo	Tipo	Largo	Otros
Número de Ingreso	Autonumérico		Campo Clave
Número de Historial	Texto	9	
Fecha de Ingreso	Fecha		
Código de Identificación	Texto	4	
Número de planta	Número		
Número de cama	Número		
Alérgico	Sí/No		
Observaciones	Memo		
Coste del tratamiento	Número		Formato de moneda
Diagnóstico	Texto	40	

- Crea el modelo Relacional a partir del cual se habrán deducido dichas tablas
- Crea la BD en Microsoft ACCESS o LibreOffice BASE teniendo en cuenta la información adicional que se muestra en las tablas anteriores.
- Introduce los datos siguientes en la BD.

Las tablas se dividen en dos porque contienen muchas columnas

Tabla 38: PACIENTES (inicio)

N Seguridad Social	Nombre	Apellidos	Domicilio	Población
08/7888888	José Eduardo	Romerales Pinto	C/ Azorín, 34 3o	Móstoles
08/7234823	Ángel	Ruíz Picasso	C/ Salmerón, 212	Madrid
08/7333333	Mercedes	Romero Carvajal	C/ Málaga, 13	Móstoles
08/7555555	Martín	Fernández López	C/ Sastres, 21	Madrid

Tabla 39: PACIENTES (continuación)

Provincia	Código Postal	Teléfono	Número de Historial	Sexo
Madrid	28935	91-345-87-45	10203-F	H
Madrid	28028	91-565-34-33	11454-L	H
Madrid	28935	91-455-67-45	14546-E	M
Madrid	28028	91-333-33-33	15413-S	H

Tabla 40: INGRESOS (inicio)

Número de Ingreso	Número de Hist.	Fecha de Ingreso	Código de Identi.	Número de planta
1	10203-F	23/01/2009	AJH	5
2	15413-S	13/03/2009	RLQ	2
3	11454-L	25/05/2009	RLQ	3
4	15413-S	29/01/2010	CEM	2
5	14546-E	24-02/2010	AJH	1

Tabla 41: INGRESOS (continuación)

Número de cama	Alérgico	Observaciones
121	No	Epiléptico
5	Sí	Alérgico a la penicilina
31	No	
13	No	
5	Sí	Alérgico al Paidoterín

Tabla 42: MÉDICOS (inicio)

Código de Ident.	Nombre del Médico	Apellidos del Médico	Especialidad	Fecha toma posesión
AJH	Antonio	Jaén Hernández	Pediatría	12-08-82
CEM	Carmen	Esteril Manrique	Psiquiatría	13-02-92
RLQ	Rocío	López Quijada	Médico de familia	23-09-94

Tabla 43: MÉDICOS (continuación)

Cargo	Número de Colegiado	Observaciones
Adjunto	2113	Está próxima su retirada
Jefe de sección	1231	
Titular	1331	

d) Realiza las siguientes consultas:

- 1) Nombre y fecha de toma de posesión de los médicos pediatras del hospital.
- 2) Nombre de los pacientes residentes en Madrid capital.
- 3) Nombre de los médicos que autorizaron ingresos entre enero y febrero de 2010.
- 4) Nombre y número de la Seguridad social de todos los pacientes.
- 5) Nombres y apellidos de los pacientes que ingresaron entre enero y mayo de 2009 y son alérgicos.
- 6) Habitación y la planta en la que ingresaron los pacientes de Móstoles.
- 7) Pacientes cuyo ingreso haya sido autorizado por el doctor Antonio Jaén Hernández.
- 8) Nombre y Teléfono de los pacientes que ingresaron en 2010.
- 9) Nombre de los pacientes ingresados que sufren epilepsia.
- 10) Nombre y fecha de ingreso de aquellos pacientes que hayan sido atendidos por un psiquiatra.

DISEÑO FÍSICO DE BASES DE DATOS. LENGUAJE DE DEFINICIÓN DE DATOS

3.1 INTRODUCCIÓN

En la fase de análisis hemos realizado la E.R.S. A partir de dicha especificación de requisitos, en la unidad anterior, aprendimos a realizar el diseño conceptual de una BD mediante el Modelo E/R y el Modelo E/R extendido respectivamente.

También vimos como hacer el modelo lógico mediante el modelo relacional que se obtenía a partir del modelo E/R y aprendimos a comprobar que dicho modelo estaba normalizado.

Siguiendo con el proceso de desarrollo, lo que debemos hacer ahora es pasar al diseño físico de la BD. Es decir, implementar la Base de Datos. Para ello, se programarán las diferentes tablas que constituirán la Base de Datos, se introducirán los datos y, más adelante, se construirán las consultas. Todo ello se hará programando en el lenguaje más extendido para la definición y manipulación de datos en SGBDR: **SQL**.

Las prácticas del módulo de Gestión Bases de Datos se van a realizar utilizando el **Sistema de Gestión de Bases de Datos Relacional (RDBMS) ORACLE**. Varias son las razones que justifican la impartición de estas prácticas utilizando ORACLE:

- En primer lugar, ORACLE es un producto comercial ampliamente extendido y utilizado, que cuenta con una importante cuota de mercado dentro del mundo de las bases de datos, estando disponible para prácticamente la totalidad de plataformas posibles (Windows, MAC, UNIX, LINUX, ...) con la ventaja de que las aplicaciones realizadas para una plataforma concreta pueden ser portadas de forma automática a cualquiera de las otras plataformas.
- ORACLE permite almacenar y manejar gran cantidad de información de forma rápida y segura, destacando además su valor educativo, ya que la herramienta que utiliza ORACLE para acceder a la base de datos es el lenguaje no procedural SQL, y este lenguaje implementa prácticamente toda la funcionalidad y características del modelo relacional teórico.

3.2 EL LENGUAJE SQL

3.2.1 Historia

El nacimiento del lenguaje SQL data de 1970 cuando E. F. Codd publica su libro: «Un modelo de datos relacional para grandes bancos de datos compartidos». Ese libro dictaría las directrices de las bases de datos relacionales. Apenas dos años después IBM (para quien trabajaba Codd) utiliza las directrices de Codd para crear el Standard English Query Language (Lenguaje Estándar Inglés para Consultas) al que se le llamó SEQUEL. Más adelante se le asignaron las siglas SQL (Standard Query Language, lenguaje estándar de consulta) aunque en inglés se siguen pronunciando secuel. En español se pronuncia esecuele.

En 1979 Oracle presenta la primera implementación comercial del lenguaje. Poco después se convertía en un estándar en el mundo de las bases de datos avalado por los organismos ISO y ANSI. En el año 1986 se toma como lenguaje estándar por ANSI de los SGBD relacionales. Un año después lo adopta ISO, lo que convierte a SQL en estándar mundial como lenguaje de bases de datos relacionales.

En 1989 aparece el estándar ISO (y ANSI) llamado SQL89 o SQL1. En 1992 aparece la nueva versión estándar de SQL (a día de hoy sigue siendo la más conocida) llamada SQL92. En 1999 se aprueba un nuevo SQL estándar que incorpora mejoras que incluyen triggers, procedimientos, funciones,... y otras características de las bases de datos objeto-relacionales; dicho estándar se conoce como SQL99. El último estándar es el del año 2011 (SQL2011) Elementos de SQL

SQL se basa en la Teoría Matemática del Álgebra Relacional. El lenguaje SQL consta de varios elementos:

- **Lenguaje de definición de datos (DDL):** proporciona órdenes para definir, modificar o eliminar los distintos objetos de la base de datos (tablas, vistas, índices...).
- **Lenguaje de Manipulación de Datos (DML):** proporciona órdenes para insertar, suprimir y modificar registros o filas de las tablas. También contempla la realización de consultas sobre la BD.
- **Lenguaje de Control de Datos (DCL):** permite establecer derechos de acceso de los usuarios sobre los distintos objetos de la base de datos. Lo forman las instrucciones **GRANT** y **REVOKE**.
- Oracle contempla además s**entencias para transacciones**. Administran las modificaciones creadas por las instrucciones DML. Lo forman las instrucciones **ROLLBACK**, **COMMIT** y **SAVEPOINT**.

3.2.2 Proceso de ejecución de sentencia SQL

El proceso de una instrucción SQL es el siguiente:

- 1) Se analiza la instrucción. Para comprobar la sintaxis de la misma
- 2) Si es correcta se valora si los metadatos de la misma son correctos. Se comprueba esto en el diccionario de datos.
- 3) Si es correcta, se optimiza, a fin de consumir los mínimos recursos posibles.
- 4) Se ejecuta la sentencia y se muestra el resultado al emisor de la misma.

3.2.3 Criterios de notación

La notación utilizada para la especificación de los comandos de SQL es la siguiente:

CRITERIOS DE NOTACIÓN

- Palabras clave de la sintaxis SQL en MAYÚSCULAS.
- Los corchetes [] indican opcionalidad.

- Las llaves { } delimitan alternativas separadas por | de las que se debe elegir una.
- Los puntos suspensivos ... indican repetición varias veces de la opción anterior.

3.2.4 Normas de escritura

En SQL no se distingue entre mayúsculas y minúsculas. Da lo mismo como se escriba. El final de una instrucción o sentencia lo marca el signo de punto y coma.

Las sentencias SQL (SELECT, INSERT, ...) se pueden escribir en varias líneas siempre que las palabras no sean partidas. Los comentarios en el código SQL pueden ser de 2 tipos:

- de bloque: comienzan por /* y terminan por */
- de línea: comienzan por – y terminan en final de línea

Ejemplos:

```
/*
  Esto es un comentario
  de varias líneas.

  Fin.
*/

-- Esto es un comentario de una línea
```

3.3 LENGUAJE DE DEFINICIÓN DE DATOS: DDL

3.3.1 Tipos de datos y conversión entre tipos

Los tipos de datos principales de ORACLE son los siguientes:

- **CHAR(n)**
Cadena de caracteres de longitud fija. Se puede especificar el número de caracteres que tendrá (n).
- **VARCHAR2(n)**
Cadena de caracteres de longitud variable. Se debe especificar el número de caracteres que tendrá (n).
- **NUMBER(n)**
Dato de tipo numérico de un máximo de 40 dígitos, además del signo y el punto decimal. Se puede utilizar notación científica (1.273E2 es igual a 127.3). Se usa para Números enteros. Se puede especificar el número de dígitos (n).
- **NUMBER(p,d)**
Números reales. Donde “p” especifica el número total de dígitos (máximo 38 dígitos) y “d” el número total de decimales. Por ejemplo NUMBER(4,2) tiene como máximo valor 99.99.
- **DATE**. El tipo DATE permite almacenar fechas.

Comparativa estándar SQL y Oracle SQL

Descripción	Tipos Estándar SQL	Oracle SQL
Texto		
Texto de anchura fija	CHARACTER(<i>n</i>) CHAR(<i>n</i>)	CHAR(<i>n</i>)
Texto de anchura variable	CHARACTER VARYING(<i>n</i>) VARCHAR(<i>n</i>)	VARCHAR2(<i>n</i>)
Texto de anchura fija para caracteres nacionales	NATIONAL CHARACTER(<i>n</i>) NATIONAL CHAR(<i>n</i>) NCHAR(<i>n</i>)	NCHAR(<i>n</i>)
Texto de anchura variable para caracteres nacionales	NATIONAL CHARACTER VARYING(<i>n</i>) NATIONAL CHAR VARYING(<i>n</i>) NCHAR VARYING(<i>n</i>)	NVARCHAR2(<i>n</i>)
Números		
Enteros pequeños (2 bytes)	SMALLINT	
Enteros normales (4 bytes)	INTEGER INT	
Enteros largos (8 bytes)	BIGINT (en realidad no es estándar, pero es muy utilizado en muchas bases de datos)	
Enteros precisión decimal		NUMBER(<i>n</i>)
Decimal de coma variable	FLOAT DOUBLE DOUBLE PRECISION REAL	NUMBER
Decimal de coma fija	NUMERIC(<i>m,d</i>) DECIMAL(<i>m,d</i>)	NUMBER(<i>m,d</i>)
Fechas		
Fechas	DATE	DATE
Fecha y hora	TIMESTAMP	TIMESTAMP
Intervalos	INTERVAL	INTERVAL
Booleanos y binarios		
Lógicos	BOOLEAN BOOL	
Binarios	BIT BIT VARYING(<i>n</i>) VARBIT(<i>n</i>)	
Datos de gran tamaño		
Texto gran longitud	CHARACTER LARGE OBJECT CLOB	LONG (en desuso) CLOB
Binario de gran longitud	BINARY LARGE OBJECT BLOB	RAW (en desuso) LONG RAW (en desuso) BLOB

Cadenas de caracteres: CHAR(n) y VARCHAR(n)

Las cadenas de caracteres se delimitan utilizando comillas simples.

Por ejemplo: “Hola”, “Una cadena”.

Conviene poner suficiente espacio para almacenar los valores. En el caso de los VARCHAR, Oracle no malgasta espacio por poner más espacio del deseado ya que si el texto es más pequeño que el tamaño indicado, el resto del espacio se ocupa.

Además de los operadores de igualdad (=, !=, ...) otras funciones útiles para trabajar con cadenas son:

- **cad1 || cad2** : concatena dos cadenas.
- **LENGTH(cad)**: devuelve la longitud de la cadena.
- **LOWER(cad)**: convierte todas las letras de la cadena a minúsculas.
- **UPPER(cad)**: ídem a mayúsculas.

Números: NUMBER

El tipo NUMBER es un formato versátil que permite representar todo tipo de números. Su rango recoge números de entre 1×10^{-130} to $9.99 \dots 9 \times 10^{125}$. Fuera de estos rangos Oracle devuelve un error.

Los números decimales (números de coma fija) se indican con NUMBER(p,d), donde p es la precisión máxima y d es el número de decimales a la derecha de la coma. Por ejemplo, NUMBER (8,3) indica que se representan números de ocho cifras de precisión y tres decimales. Los decimales en Oracle se presenta con el punto y no con la coma.

Para números enteros se indica NUMBER(p) donde p es el número de dígitos. Eso es equivalente a NUMBER(p,0).

Para números de coma flotante (equivalentes a los float o double de muchos lenguajes de programación) simplemente se indica el texto NUMBER sin precisión ni escala.

Además de las operaciones típicas con valores numéricos (+, -, *, /), otras funciones útiles son:

- **ABS(num)**: devuelve el valor absoluto.
- **SQRT(num)**: devuelve la raíz cuadrada.
- **POWER(b,e)**: devuelve la potencia de b elevado a e.

Existen otras funciones para grupos de valores (suma, media, máximo, ...) que se verán en apartados posteriores.

Fechas: DATE

Las fechas se pueden escribir en formato día, mes y año entre comillas simples. El separador puede ser una barra de dividir, un guión y casi cualquier símbolo.

Tanto el día como el año tiene formato numérico y el mes se indica con las tres primeras letras del nombre del mes en el idioma soportado por el servidor ORACLE.

Ejemplos: “1-JAN-96”, “28-jul-74”. Además de esta información, un valor de tipo fecha almacena también la hora en formato hh:mm:ss.

Las fechas se pueden comparar con los operadores típicos de comparación (<, >, !=, =, ...).

La función **SYSDATE devuelve la fecha actual** (fecha y hora). Con las fechas es posible realizar operaciones aritméticas como sumas y restas de fechas, teniendo en cuenta que a una fecha se le suman días y que la diferencia entre dos fechas se devuelve también en días. Por ejemplo SYSDATE + 1 devuelve la fecha de mañana.

Datos de gran tamaño

Son tipos pensados para almacenar datos de tamaño muy grande. No pueden poseer índices ni ser parte de claves.

- **CLOB** (Character Large Object)

Utilizado para almacenar datos de texto de gran tamaño (hasta hasta 128 TB texto)

- **BLOB** (Binary Large Object)

Utilizado para guardar datos binarios de hasta 128 TB de tamaño. Se utilizan para almacenar datos binarios, típicamente imágenes, vídeos, documentos con formato como PDF o similares, ...

Conversión entre datos

Oracle permite tanto la conversión de tipos implícita como la explícita.

- La **conversión de tipos implícita** (Oracle la hace automáticamente) significa que cuando Oracle encuentra en un lugar determinado (por ejemplo en una expresión) un dato de un tipo diferente al esperado, entonces aplica una serie de reglas para intentar convertir ese dato al tipo esperado. Por ejemplo, si un atributo de una tabla determinada es de tipo NUMBER y se intenta introducir el valor de tipo carácter "1221", entonces automáticamente se convierte en su valor numérico equivalente sin producirse ningún error.
- La **conversión de tipos explícita** se realiza básicamente con las siguientes funciones, y se verá en profundidad más adelante:
 - Conversión número-cadena: **TO_CHAR** (número [, formato]).
 - Conversión cadena-número: **TO_NUMBER** (cadena [,formato]).
 - Conversión fecha-cadena: **TO_CHAR** (fecha [, formato]).
 - Conversión cadena-fecha: **TO_DATE** (cadena [, formato]).

3.3.2 Expresiones y operadores condicionales

Las condiciones son expresiones lógicas (devuelven verdadero o falso) que se sitúan normalmente junto a una cláusula SQL que utilizan muchos comandos. Dentro del DDL se utilizarán con la cláusula CHECK que sirve para establecer las condiciones que deben cumplir sobre los valores que se almacenarán en una tabla.

Las condiciones se construyen utilizando los operadores de comparación y los operadores lógicos. A continuación se describen los operadores más importantes junto con ejemplos de su utilización.

Operadores de comparación: =, <>, !=, <=, >=, < y >

Con ellos podemos realizar comparaciones de igualdad, desigualdad, ...

Ejemplos:

```
horas >= 10.5
nombre = 'PEPE'
fecha < '1-ene-93'
```

[NOT] IN *lista_valores*

Comprueba la pertenencia a la lista de valores. Generalmente, los valores de la lista se habrán obtenido como resultado de un comando SELECT (comando de consulta).

Ejemplo:

```
nombre NOT IN ('PEPE', 'LOLA')
```

oper {ANY | SOME} *lista_valores*

Comprueba que se cumple la operación *oper* con algún elemento de la lista de valores. *oper* puede ser =, <>, !=, <, >, <=, >=.

Ejemplo:

```
nombre = ANY ('PEPE', 'LOLA')
```

oper ALL *lista_valores*

Comprueba que se cumple la operación *oper* con todos los elementos de la lista de valores. *oper* puede ser =, <>, !=, <, >, <=, >=.

Ejemplo:

```
nombre <> ALL ('PEPE', 'LOLA')
```

[NOT] BETWEEN *x* AND *y*

Comprueba la pertenencia al rango *x* - *y*.

Ejemplo:

```
horas BETWEEN 10 AND 20    -- que equivale a horas >= 10 AND horas <= 20
```

[NOT] EXISTS *lista_valores*

Comprueba si la lista de valores contiene algún elemento.

Ejemplos:

```
EXISTS ('ALGO')           -- devuelve verdadero.  
NOT EXISTS ('ALGO')       -- devuelve falso.
```

[NOT] LIKE *texto*

Permite comparar cadenas alfanuméricas haciendo uso de símbolos comodín.

Los símbolos comodín que pueden usarse son dos:

- `_` : sustituye a un único carácter.
- `%` : sustituye a varios caracteres.

Ejemplos:

```
nombre LIKE 'Pedro%'
codigo NOT LIKE 'cod1_'
```

Si dentro de una cadena se quieren utilizar los caracteres “%” o “_” tienen que ser escapados utilizando el símbolo “\”.

IS [NOT] NULL

Cuando el valor de un atributo, o es desconocido, o no es aplicable esa información, se hace uso del valor nulo (NULL). Para la comparación de valores nulos se utiliza el operador IS [NOT] NULL.

Ejemplo:

```
teléfono IS NULL
```

Operadores lógicos: OR, AND y NOT

Los operadores lógicos junto con el uso de paréntesis permiten combinar condiciones simples obteniendo otras más complejas. Los operadores lógicos son:

- *condición1* **OR** *condición2*: Cierto en todos los casos, salvo que las 2 condiciones sean falsas.
- *condición1* **AND** *condición2*: Falso en todos los casos, salvo que las 2 condiciones sean ciertas.
- **NOT** *condición*: Invierte la condición.

Ejemplos:

```
nombre = 'PEPE' OR horas BETWEEN 10 AND 20
horas > 10 AND telefono IS NULL
NOT (nombre IN ('PEPE', 'LUIS'))
```

3.3.3 Creación, Modificación y Eliminación de bases de datos

En Oracle la creación, eliminación y modificación de una base de datos resulta una tarea relativamente compleja. Por ahora sólo se comenta de forma muy simple.

Creación de una Base de datos

Crear la base de datos implica indicar los archivos y ubicaciones que se utilizarán para la misma, además de otras indicaciones técnicas y administrativas que no se comentarán en este tema. Lógicamente sólo es posible crear una base de datos si se tienen privilegios DBA (DataBase Administrator) (SYSDBA en el caso de Oracle).

El comando SQL de creación de una base de datos es **CREATE DATABASE**. Este comando crea una base de datos con el nombre que se indique. Ejemplo:

```
CREATE DATABASE prueba;
```

Pero normalmente se indican más parámetros. Ejemplo (parámetros de Oracle):

```
CREATE DATABASE prueba
LOGFILE prueba.log
MAXLOGFILES 25
MAXINSTANCES 10
ARCHIVELOG
CHARACTER SET AL32UTF8
NATIONAL CHARACTER SET UTF8
DATAFILE prueba1.dbf AUTOEXTEND ON MAXSIZE 500MB;
```

Eliminación de una Base de datos

La sentencia que se utiliza para ello es **DROP DATABASE**.

```
DROP DATABASE prueba;
```

Modificación de una Base de datos

Se utiliza la sentencia **ALTER DATABASE** que posee innumerables cláusulas.

```
ALTER DATABASE prueba ...;
```

3.3.4 Creación, Modificación y Eliminación de esquemas

Según los estándares actuales, una base de datos es un conjunto de objetos pensados para gestionar datos. Estos objetos (tablas, vistas, secuencias, ...) están contenidos en esquemas, los esquemas suelen estar asociados al perfil de un usuario en particular. En Oracle, cuando se crea un usuario, se crea un esquema cuyo nombre es idéntico al del usuario.

Creación de un Esquema

En Oracle para crear un esquema o usuario se utiliza la sentencia **CREATE USER**. La forma más sencilla de uso es:

```
CREATE USER nombre IDENTIFIED BY contraseña;
```

Aunque, con frecuencia, se añaden diversas cláusulas. Una sentencia más detallada es:

```
CREATE USER nombre
IDENTIFIED BY clave
DEFAULT TABLESPACE users
QUOTA 10M ON users
TEMPORARY TABLESPACE temp
QUOTA 5M ON temp
PASSWORD EXPIRE;
```

Eliminación de un Esquema

Se realiza mediante la sentencia **DROP USER**:

```
DROP USER usuario [CASCADE];
```

La opción **CASCADE** elimina los objetos del esquema del usuario antes de eliminar al propio usuario. Es obligatorio si el esquema contiene objetos.

Modificación de un Esquema

Cada parámetro indicado en la creación del esquema puede modificarse mediante la instrucción **ALTER USER**, que se utiliza igual que **CREATE USER**. Ejemplo:

```
ALTER USER nombre IDENTIFIED BY "nuevaclave";
ALTER USER nombre QUOTA UNLIMITED ON users;
```

3.3.5 Creación, Modificación y Eliminación de tablas

En este apartado veremos los comandos SQL que se utilizarán para crear y modificar la definición de una tabla, así como para eliminarla de la base de datos.

Creación de Tablas

El nombre de las tablas debe cumplir las siguientes reglas:

- Deben comenzar con una letra
- No deben tener más de 30 caracteres
- Sólo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (también el signo \$ y #, pero esos se utilizan de manera especial por lo que no son recomendados)
- No puede haber dos tablas con el mismo nombre para el mismo usuario (pueden coincidir los nombres si están en distintos esquemas)
- No puede coincidir con el nombre de una palabra reservada de SQL

Para la creación de tablas con SQL se utiliza el comando **CREATE TABLE**. Este comando tiene una sintaxis más compleja de la que aquí se expone, pero vamos a comenzar por la sintaxis básica. Sintaxis básica de creación de tablas:

```
CREATE TABLE nombre_tabla (
  columna1 tipo_dato [ restricciones de columna1 ],
  columna2 tipo_dato [ restricciones de columna2 ],
  columna3 tipo_dato [ restricciones de columna3 ],
  ...
  [ restricciones de tabla ]
);
```

Para realizar las separaciones se utiliza la coma. **La última línea, antes del paréntesis de cierre, no lleva coma.**

Donde las restricciones de columna tienen la siguiente sintaxis:

```
CONSTRAINT nombre_restricción {
  [NOT] NULL | UNIQUE | PRIMARY KEY | DEFAULT valor | CHECK (condición)
}
```

Y las restricciones de tabla tienen la siguiente sintaxis:

```
CONSTRAINT nombre_restricción {  
  PRIMARY KEY (columna1 [,columna2] ... )  
| UNIQUE (columna1 [,columna2] ... )  
| FOREIGN KEY (columna1 [,columna2] ... )  
  REFERENCES nombre_tabla (columna1 [,columna2] ... )  
  [ON DELETE {CASCADE | SET NULL}]  
| CHECK (condición)  
}
```

Obligatoriamente debemos crear una restricción de tabla cuando una misma restricción afecte a varias columnas. Por ejemplo si tenemos una clave primaria compuesta por varios campos, debemos establecer una restricción de tabla, no de columna.

El significado de las distintas opciones que aparecen en la sintaxis CREATE TABLE es:

- **PRIMARY KEY**: establece ese atributo o conjunto de atributos como la clave primaria de la tabla. Esta restricción ya implica las restricciones UNIQUE y NOT NULL.
- **UNIQUE**: impide que se introduzcan valores repetidos para ese atributo o conjunto de atributos. No se puede utilizar junto con PRIMARY KEY. Se utiliza para claves alternativas.
- **NOT NULL**: evita que se introduzcan filas en la tabla con valor NULL para ese atributo. No se utiliza con PRIMARY KEY.
- **DEFAULT** valor_por_defecto: permite asignar un valor por defecto al campo que se está definiendo.
- **CHECK** (condición): permite establecer condiciones que deben cumplir los valores de la tabla que se introducirán en dicha columna.
 - Si un CHECK se especifica como una restricción de columna, la condición sólo se puede referir a esa columna.
 - Si el CHECK se especifica como restricción de tabla, la condición puede afectar a todas las columnas de la tabla.
 - Sólo se permiten condiciones simples, por ejemplo, no está permitido referirse a columnas de otras tablas o formular subconsultas dentro de un CHECK.
 - Además las funciones SYSDATE y USER no se pueden utilizar dentro de la condición. En principio están permitidas comparaciones simples de atributos y operadores lógicos (AND, OR y NOT).
- **FOREIGN KEY**: define una clave externa de la tabla respecto de otra tabla. Esta restricción especifica una columna o una lista de columnas como clave externa de una tabla referenciada. No se puede definir una restricción de integridad referencial que se refiere a una tabla antes de que dicha tabla haya sido creada. Es importante resaltar que una clave externa debe referenciar a una clave primaria completa de la tabla padre, y nunca a un subconjunto de los atributos que forman esta clave primaria.
 - **ON DELETE CASCADE**: especifica que se mantenga automáticamente la integridad referencial borrando los valores de la llave externa correspondientes a un valor borrado de la tabla referenciada (tabla padre). Si se omite esta opción no se permitirá borrar valores de una tabla que sean referenciados como llave externa en otras tablas.
 - **ON DELETE SET NULL**: especifica que se ponga a NULL los valores de la llave externa correspondientes a un valor borrado de la tabla referenciada (tabla padre).

Nota: En Oracle, el valor predeterminado es que las filas de la tabla principal no se pueden eliminar si existe una fila en la tabla secundaria que se refiere a esta fila principal, si no indicamos ON DELETE CASCADE o ON DELETE SET NULL. El estándar SQL define muchas más opciones.

Nota: El estándar SQL define 5 opciones para manejar esta situación de tablas principal/secundaria de varias maneras. Estas opciones son:

- **ON DELETE CASCADE:** si se elimina una fila de la tabla principal, se eliminan todas las filas coincidentes en la tabla secundaria.
- **ON DELETE SET NULL:** si se elimina una fila de la tabla principal, todas las columnas de referencia en todas las filas coincidentes de la tabla secundaria se establecen en NULL.
- **ON DELETE SET DEFAULT:** si se elimina una fila de la tabla principal, todas las columnas de referencia en todas las filas coincidentes de la tabla secundaria se configuran en el valor predeterminado de la columna.
- **ON DELETE RESTRICT:** está prohibido eliminar una fila de la tabla principal si esa fila tiene alguna fila coincidente en la tabla secundaria. El punto en el tiempo cuando realiza la comprobación se puede aplazar hasta que se realice COMMIT.
- **ON DELETE NO ACTION** (el valor predeterminado): se prohíbe eliminar una fila de la tabla primaria si esa fila tiene filas coincidentes en la tabla secundaria.

Análoga a la opción ON DELETE hay una opción ON UPDATE. Define las mismas 5 opciones para el caso de cambiar una columna en la tabla principal a la que hace referencia la columna de una tabla secundaria.

- **ON UPDATE CASCADE:** Cualquier cambio en una columna referenciada en la tabla primaria provoca el mismo cambio en la columna de referencia correspondiente en las filas coincidentes de la tabla secundaria.
- **ON UPDATE SET NULL:** Cualquier cambio en una columna referenciada en la tabla primaria provoca que la columna de referencia correspondiente en las filas coincidentes de la tabla secundaria se establezca como nula.
- **ON UPDATE SET DEFAULT:** Cualquier cambio en una columna referenciada en la tabla principal provoca que la columna de referencia correspondiente en las filas coincidentes de la tabla de secundaria se establezca en su valor predeterminado.
- **ON UPDATE RESTRICT:** está prohibido cambiar una fila de la tabla principal si esa fila tiene filas coincidentes en la tabla secundaria. El punto en el tiempo cuando se realiza la comprobación se puede aplazar hasta que se realice COMMIT.
- **ON UPDATE NO ACTION** (valor predeterminado): está prohibido cambiar una fila de la tabla principal si esa fila tiene alguna fila coincidente en la tabla secundaria.

Si ON DELETE o ON UPDATE no están especificados, se producirá la acción predeterminada NO ACTION. En algunos sistemas, NO ACTION se implementa en el sentido de la opción RESTRICT.

En la definición de una tabla pueden aparecer varias cláusulas FOREIGN KEY, tantas como llaves externas tenga la tabla, sin embargo sólo puede existir una llave primaria, si bien esta llave primaria puede estar formada por varios atributos.

La utilización de la cláusula **CONSTRAINT nombre_restricción** establece un nombre determinado para la restricción de integridad, lo cual permite buscar en el Diccionario de Datos de la base de datos con posterioridad y fácilmente las restricciones introducidas para una determinada tabla.

Ejemplos:

```
CREATE TABLE usuarios (
  id          NUMBER          PRIMARY KEY,
  dni         CHAR(9)         UNIQUE,
  nombre      VARCHAR2(50)    NOT NULL,
  edad        NUMBER          CHECK (edad>=0 and edad<120)
);
```

En el caso anterior no hemos asignado nombre a las restricciones, así que Oracle le asignará un nombre de la forma SYS_Cn, donde n es un número. Esta forma no es recomendable puesto que si deseamos modificar posteriormente el diseño de la tabla nos será muy difícil gestionar las restricciones.

Otra forma más adecuada es dando nombre a las restricciones:

```
CREATE TABLE usuarios (  
  id          NUMBER      CONSTRAINT usu_id_pk  PRIMARY KEY,  
  dni         CHAR (9)    CONSTRAINT usu_dni_uq  UNIQUE,  
  nombre      VARCHAR2 (50) CONSTRAINT usu_nom_nn NOT NULL,  
  edad        NUMBER      CONSTRAINT usu_edad_ck  
                                CHECK (edad>=0 and edad<120)  
);
```

La vista **USER_TABLES** contiene una lista de las tablas del usuario actual (o del esquema actual). Así para sacar la lista de tablas del usuario actual, se haría:

```
SELECT * FROM USER_TABLES;
```

Esta vista obtiene numerosas columnas, en concreto la columna TABLES_NAME muestra el nombre de cada tabla. La vista ALL_TABLES mostrará una lista de todas las tablas de la base de datos (no solo del usuario actual), aunque oculta las que el usuario no tiene derecho a ver.

Finalmente la vista DBA_TABLES es una tabla que contiene absolutamente todas las tablas del sistema; esto es accesible sólo por el usuario administrador (DBA).

Nota: El comando **DESCRIBE**, permite obtener la estructura de una tabla.

Ejemplo:

```
DESCRIBE COCHES;
```

Y aparecerán los campos de la tabla COCHES

Criterios de notación para los nombres de restricciones

Para la Restricción de Clave principal (solo una en cada tabla):

```
CONSTRAINT tabla_campo_pk PRIMARY KEY ...
```

Para Restricciones de Clave foránea (puede haber varias en cada tabla):

```
CONSTRAINT tabla_campo_fk1 FOREIGN KEY ...  
CONSTRAINT tabla_campo_fk2 FOREIGN KEY ...  
CONSTRAINT tabla_campo_fk3 FOREIGN KEY ...  
...
```

Para Restricciones de tipo CHECK (puede haber varias en cada tabla)

```
CONSTRAINT tabla_campo_ck1 CHECK ...  
CONSTRAINT tabla_campo_ck2 CHECK ...  
...
```

Para Restricciones de tipo UNIQUE (puede haber varias en cada tabla)

```
CONSTRAINT tabla_campo_uq1 UNIQUE ...  
CONSTRAINT tabla_campo_uq2 UNIQUE ...  
...
```

Ejemplo:

```
CREATE TABLE COCHES (
    matricula          VARCHAR2(8),
    marca              VARCHAR2(15) NOT NULL,
    color              VARCHAR2(15),
    codTaller          VARCHAR2(10),
    codProp            VARCHAR2(10),
    CONSTRAINT coches_mat_pk PRIMARY KEY (matricula),
    CONSTRAINT coches_codtaller_fk1 FOREIGN KEY (codTaller)
        REFERENCES TALLER(codTaller),
    CONSTRAINT coches_codprop_fk2 FOREIGN KEY (codProp)
        REFERENCES PROPIETARIO(codProp),
    CONSTRAINT coches_color_ck1
        CHECK (color IN ('ROJO', 'AZUL', 'BLANCO', 'GRIS', 'VERDE', 'NEGRO'))
);
```

Se puede utilizar la vista USER_CONSTRAINTS del diccionario de datos para identificar las restricciones colocadas por el usuario. La vista ALL_CONSTRAINTS permite mostrar las restricciones de todos los usuarios, pero sólo está permitida a los administradores). Además, la vista USER_CONS_COLUMNS, nos muestra información sobre las columnas que participan en una restricción.

Eliminación de Tablas

La sentencia en SQL para eliminar tablas es **DROP TABLE**. Su sintaxis es:

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

La opción **CASCADE CONSTRAINTS** permite eliminar una tabla que contenga atributos referenciados por otras tablas, eliminando también todas esas referencias.

Si la clave principal de la tabla es una clave foránea en otra tabla y no utiliza la opción **CASCADE CONSTRAINTS**, entonces no se podrá eliminar la tabla.

Peligro: El borrado de una tabla es irreversible y no hay ninguna petición de confirmación, por lo que conviene ser muy cuidadoso con esta operación. Al borrar una tabla se borran todos los datos que contiene.

Ejemplos:

```
DROP TABLE COCHES;
```

Se eliminará la tabla COCHES, siempre que su clave principal no sea clave foránea de ninguna tabla de la BD.

```
DROP TABLE COCHES CASCADE CONSTRAINTS;
```

Se eliminará la tabla COCHES aunque su clave principal sea clave foránea de alguna tabla de la BD. Automáticamente se borrará la restricción de clave foránea asociada.

Modificación de Tablas

Cambiar de nombre una tabla

La orden **RENAME** permite el cambio de nombre de cualquier objeto. Sintaxis:

```
RENAME nombre TO nombre_nuevo;
```

Ejemplo:

```
RENAME COCHES TO AUTOMOVILES;
```

Cambia el nombre de la tabla COCHES y a partir de ese momento se llamará AUTOMOVILES

Borrar el contenido de una tabla

Peligro: La orden TRUNCATE TABLE seguida del nombre de una tabla, hace que se elimine el contenido de la tabla, pero no la tabla en sí. Incluso borra del archivo de datos el espacio ocupado por la tabla. (Esta orden no puede anularse con un ROLLBACK)

Ejemplo:

```
TRUNCATE TABLE AUTOMOVILES;
```

Borra los datos de la tabla AUTOMOVILES.

Trabajo con columnas y restricciones

La cláusula **ALTER TABLE** permite hacer cambios en la estructura de una tabla: añadir columna, borrar columna, modificar columna.

Añadir Columnas

```
ALTER TABLE nombre ADD (  
    columna1 tipo [ restricciones ],  
    columna2 tipo [ restricciones ]  
    ... ]  
);
```

Permite añadir nuevas columnas a la tabla. Se deben indicar su tipo de datos y sus propiedades si es necesario (al estilo de CREATE TABLE). Las nuevas columnas se añaden al final, no se puede indicar otra posición.

Ejemplos:

Añadimos la columna “fechaMatric” a la tabla VEHÍCULOS:

```
ALTER TABLE VEHICULOS ADD ( fechaMatric DATE );
```

Añadimos las columnas “fechaMatric” y “tipoFaros” a la tabla VEHÍCULOS:

```
ALTER TABLE VEHICULOS ADD (  
    fechaMatric DATE,  
    tipoFaros VARCHAR2(20) NOT NULL  
);
```

Borrar Columnas

```
ALTER TABLE nombre_tabla DROP (nombre_columna, nombre_columna2, ...);
```

Elimina la columna indicada de manera irreversible e incluyendo los datos que contenía. No se pueden eliminar todas las columnas, para la última columna habrá que usar DROP TABLE.

Ejemplo:

```
ALTER TABLE VEHICULOS DROP (tipoFaros);
```

Borra la columna “tipoFaros” de la tabla VEHICULOS y los datos que contuviera de manera irreversible.

Modificar columnas

Permite cambiar el tipo de datos y propiedades de una determinada columna. Sintaxis:

```
ALTER TABLE nombre_tabla MODIFY (
  columna1 tipo_dato [ restricciones de columna1 ] [,
  columna2 tipo_dato [ restricciones de columna2 ]
  ... ]
);
```

Ejemplo:

```
ALTER TABLE AUTOMOVILES
MODIFY (color VARCHAR2(20) NOT NULL, codTaller VARCHAR2(15));
```

Modifica dos campos o columnas de la tabla AUTOMOVILES cambiando su tamaño y además en Color, añadiendo la condición de que sea no nulo.

Los cambios que se permiten son:

- Incrementar precisión o anchura de los tipos de datos
- Sólo se puede reducir la anchura máxima de un campo si esa columna posee nulos en todos los registros, o no hay registros.
- Se puede pasar de CHAR a VARCHAR2 y viceversa (si no se modifica la anchura).
- Se puede pasar de DATE a TIMESTAMP y viceversa.

Añadir Comentarios a la Tabla

Se le pueden poner comentarios a las tablas y las columnas. Un comentario es un texto descriptivo utilizado para documentar la tabla. Sintaxis:

```
COMMENT ON {TABLE nombre_tabla | COLUMN nombre_tabla.columna }
IS 'Comentario';
```

Para mostrar los comentarios puestos se realizan consultas al diccionario de datos mediante la instrucción SELECT usando las siguientes vistas:

- USER_TAB_COMMENTS. Comentarios de las tablas del usuario actual.
- USER_COL_COMMENTS. Comentarios de las columnas del usuario actual.
- ALL_TAB_COMMENTS. Comentarios de las tablas de todos los usuarios (sólo administradores)
- ALL_COL_COMMENTS. Comentarios de las columnas de todos los usuarios (sólo administradores).

Añadir o Modificar Restricciones

Sabemos que una restricción es una condición de obligado cumplimiento para una o más columnas de la tabla. A cada restricción se le pone un nombre, en el caso de no poner un nombre (en las que eso sea posible) entonces el propio Oracle le coloca el nombre que es un nemotécnico con el nombre de tabla, columna y tipo de restricción.

Hemos visto que se pueden añadir al crear la tabla, o bien, podemos hacerlo mediante modificación posterior de la tabla. También se puede modificar una restricción creada. Su sintaxis general es:

```
ALTER TABLE nombre_tabla { ADD | MODIFY } (  
    CONSTRAINT nombre_restricción1 tipo_restricción (columnas)  
    [, CONSTRAINT nombre_restricción2 tipo_restricción (columnas) ] ...  
);
```

Borrar Restricciones

Su sintaxis es la siguiente:

```
ALTER TABLE nombre_tabla  
DROP {  
    PRIMARY KEY  
| UNIQUE (columnas)  
| CONSTRAINT nombre_restricción [ CASCADE ]  
}
```

La opción PRIMARY KEY elimina una clave principal (también quitará el índice UNIQUE sobre los campos que formaban la clave). UNIQUE elimina índices únicos. La opción CONSTRAINT elimina la restricción indicada. La opción CASCADE hace que se eliminen en cascada las restricciones de integridad que dependen de la restricción eliminada. Por ejemplo en:

```
CREATE TABLE CURSOS (  
    codCurso      CHAR(7) CONSTRAINT cursos_pk PRIMARY KEY,  
    fechaIni      DATE,  
    fechaFin      DATE,  
    titulo        VARCHAR2(60),  
    codSigCurso   CHAR(7),  
    CONSTRAINT cursos_ck1 CHECK (fechaFin > FechaIni),  
    CONSTRAINT cursos_fk1 FOREIGN KEY (codSigCurso)  
        REFERENCES CURSOS ON DELETE SET NULL  
);
```

Tras esa definición la siguiente instrucción produce error:

```
ALTER TABLE CURSOS DROP PRIMARY KEY;
```

```
ORA-02273: a esta clave única/primaria hacen referencia algunas claves ajenas
```

Para ello habría que utilizar esta instrucción:

```
ALTER TABLE CURSOS DROP PRIMARY KEY CASCADE;
```

Esa instrucción elimina la clave secundaria antes de eliminar la principal.

También produciría error esta instrucción:

```
ALTER TABLE CURSOS DROP (fechaIni);
```

```
ERROR en línea 1:  
ORA-12991: se hace referencia a la columna en una restricción de multicolumna
```

El error se debe a que no es posible borrar una columna que forma parte de la definición de una restricción. La solución es utilizar **CASCADE CONSTRAINTS** para eliminar las restricciones en las que la columna a borrar estaba implicada:

```
ALTER TABLE CURSOS DROP COLUMN (fechaIni) CASCADE CONSTRAINTS;
```

Esta instrucción elimina la restricción de tipo CHECK en la que aparecía la fecha_inicio y así se puede eliminar la columna.

Desactivar Restricciones

A veces conviene temporalmente desactivar una restricción para saltarse las reglas que impone. La sintaxis es:

```
ALTER TABLE nombre_tabla DISABLE CONSTRAINT restricción [ CASCADE ];
```

La opción **CASCADE** hace que se desactiven también las restricciones dependientes de la que se desactivó.

Activar Restricciones

Anula la desactivación. Formato:

```
ALTER TABLE nombre_tabla ENABLE CONSTRAINT restricción [ CASCADE ];
```

Sólo se permite volver a activar si los valores de la tabla cumplen la restricción que se activa. Si hubo desactivado en cascada, habrá que activar cada restricción individualmente.

Cambiar de nombre la Restricciones

Para hacerlo se utiliza este comando:

```
ALTER TABLE nombre_tabla  
RENAME CONSTRAINT nombre_restricción TO nombre_restricción_nuevo;
```

3.3.6 Creación, Modificación y Eliminación de vistas

Una vista no es más que una consulta almacenada a fin de utilizarla tantas veces como se desee. Una vista no contiene datos sino la instrucción SELECT necesaria para crear la vista, eso asegura que los datos sean coherentes al utilizar los datos almacenados en las tablas.

Las vistas se emplean para:

- Realizar consultas complejas más fácilmente
- Proporcionar tablas con datos completos
- Utilizar visiones especiales de los datos

Hay dos tipos de vistas:

- **Simples.** Las forman una sola tabla y no contienen funciones de agrupación. Su ventaja es que permiten siempre realizar operaciones DML sobre ellas.
- **Complejas.** Obtienen datos de varias tablas, pueden utilizar funciones de agrupación. No siempre permiten operaciones DML.

Creación de Vistas

Sintaxis:

```
CREATE [ OR REPLACE ] VIEW nombre_vista [ (alias1 [, alias2] ...) ]  
AS SELECT ...
```

- **OR REPLACE.** Especifique OR REPLACE para volver a crear la vista si ya existe. Puede utilizar esta cláusula para cambiar la definición de una vista existente sin eliminar, volver a crear y volver a conceder los privilegios de objeto previamente concedidos.
- **alias.** Lista de alias que se establecen para las columnas devueltas por la consulta SELECT en la que se basa esta vista. El número de alias debe coincidir con el número de columnas devueltas por SELECT. La sentencia SELECT la trataremos en profundidad en el tema siguiente.

Lo bueno de las vistas es que tras su creación se utilizan como si fueran una tabla. La **vista USER_VIEWS** del diccionario de datos permite mostrar una lista de todas las vistas que posee el usuario actual. La columna TEXT de esa vista contiene la sentencia SQL que se utilizó para crear la vista (sentencia que es ejecutada cada vez que se invoca a la vista).

Eliminación de Vistas

Se utiliza el comando **DROP VIEW**:

```
DROP VIEW nombre_vista;
```

Modificación de Vistas

Sólo se utiliza la instrucción **ALTER VIEW** para recompilar explícitamente una vista que no es válida. Si desea cambiar la definición de una vista se debe ejecutar la sentencia **CREATE OR REPLACE nombre_vista**.

La sentencia ALTER VIEW le permite localizar errores de recompilación antes de la ejecución. Para asegurarse de que la alteración no afecta a la vista u otros objetos que dependen de ella, puede volver a compilar explícitamente una vista después de alterar una de sus tablas base.

Para utilizar la instrucción ALTER VIEW, la vista debe estar en su esquema, o debe tener el privilegio del sistema ALTER ANY TABLE.

3.3.7 Creación, Modificación y Eliminación de índices

Los índices son objetos que forman parte del esquema que hacen que las bases de datos aceleren las operaciones de consulta y ordenación sobre los campos a los que el índice hace referencia.

Se almacenan aparte de la tabla a la que hace referencia, lo que permite crearles y borrarles en cualquier momento.

Lo que realizan es una lista ordenada por la que Oracle puede acceder para facilitar la búsqueda de los datos. cada vez que se añade un nuevo registro, los índices involucrados se actualizan a fin de que su información esté al día. De ahí que cuantos más índices haya, más le cuesta a Oracle añadir registros, pero más rápidas se realizan las instrucciones de consulta.

La mayoría de los índices se crean de manera implícita, como consecuencia de las restricciones PRIMARY KEY, UNIQUE y FOREIGN KEY. Estos son índices obligatorios, por los que los crea el propio SGBD.

Creación de Índices

Aparte de los índices obligatorios comentados anteriormente, se pueden crear índices de forma explícita. Éstos se crean para aquellos campos sobre los cuales se realizarán búsquedas e instrucciones de ordenación frecuente.

Sintaxis:

```
CREATE INDEX nombre  
ON tabla (columna1 [, columna2] ...)
```

Ejemplo:

```
CREATE INDEX nombre_completo
ON clientes (apellido1, apellido2, nombre);
```

El ejemplo crea un índice para los campos apellido1, apellido2 y nombre. Esto no es lo mismo que crear un índice para cada campo, este índice es efectivo cuando se buscan u ordenan clientes usando los tres campos (apellido1, apellido2, nombre) a la vez.

Se aconseja crear índices en campos que:

- Contengan una gran cantidad de valores
- Contengan una gran cantidad de nulos
- Sean parte habitual de cláusulas WHERE, GROUP BY u ORDER BY
- Sean parte de listados de consultas de grandes tablas sobre las que casi siempre se muestran como mucho un 4 % de su contenido.

No se aconseja en campos que:

- Pertenezcan a tablas pequeñas
- No se usen a menudo en las consultas
- Pertenecen a tablas cuyas consultas muestran menos de un 4 % del total de registros
- Pertenecen a tablas que se actualizan frecuentemente
- Se utilizan en expresiones

Los índices se pueden crear utilizando expresiones complejas:

```
CREATE INDEX nombre_complejo
ON clientes (UPPER(nombre));
```

Esos índices tienen sentido si en las consultas se utilizan exactamente esas expresiones. Para ver la lista de índices en Oracle se utiliza la vista USER_INDEXES. Mientras que la **vista** USER_IND_COLUMNS muestra la lista de columnas que son utilizadas por índices.

Eliminación de Índices

La instrucción DROP INDEX seguida del nombre del índice permite eliminar el índice en cuestión.

```
DROP INDEX nombre_indice;
```

3.3.8 Creación, Modificación y Eliminación de secuencias

Una secuencia sirve para generar automáticamente números distintos. Se utilizan para generar valores para campos que se utilizan como clave forzada (claves cuyo valor no interesa, sólo sirven para identificar los registros de una tabla). Es decir se utilizan en los identificadores de las tablas (campos que comienzan con la palabra id), siempre y cuando no importe qué número se asigna a cada fila.

Es una rutina interna de la base de datos la que realiza la función de generar un número distinto cada vez. Las secuencias se almacenan independientemente de la tabla, por lo que la misma secuencia se puede utilizar para diversas tablas.

Creación de Secuencias

Sintaxis:

```
CREATE SEQUENCE secuencia
[ INCREMENT BY n ]
[ START WITH n ]
[ { MAXVALUE n | NOMAXVALUE } ]
[ { MINVALUE n | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ];
```

Donde:

- **secuencia.** Es el nombre que se le da al objeto de secuencia
- **INCREMENT BY.** Indica cuánto se incrementa la secuencia cada vez que se usa. Por defecto se incrementa de uno en uno
- **START WITH.** Indica el valor inicial de la secuencia (por defecto 1)
- **MAXVALUE.** Máximo valor que puede tomar la secuencia. Si no se toma NOMAXVALUE que permite llegar hasta 1027
- **MINVALUE.** Mínimo valor que puede tomar la secuencia. Si el incremento es negativo y no se toma NOMINVALUE permite llegar hasta -1026
- **CYCLE.** Hace que la secuencia vuelva a empezar si se ha llegado al máximo valor.

Ejemplo:

```
CREATE SEQUENCE numeroPlanta
INCREMENT BY 100
START WITH 100
MAXVALUE 2000;
```

En el diccionario de datos de Oracle tenemos la **vista USER_SEQUENCES** que muestra la lista de secuencias actuales. La columna **LAST_NUMBER** muestra cual será el siguiente número de secuencia disponible uso de la secuencia. Los métodos **NEXTVAL** y **CURRVAL** se utilizan para obtener el siguiente número y el valor actual de la secuencia respectivamente. Ejemplo de uso (Oracle):

```
SELECT numeroPlanta.NEXTVAL FROM DUAL;
```

En SQL estándar:

```
SELECT nextval('numeroPlanta');
```

Eso muestra en pantalla el siguiente valor de la secuencia. Realmente **NEXTVAL** incrementa la secuencia y devuelve el valor actual. **CURRVAL** devuelve el valor de la secuencia, pero sin incrementar la misma.

Ambas funciones pueden ser utilizadas en:

- Una consulta **SELECT** que no lleve **DISTINCT**, ni grupos, ni sea parte de una vista, ni sea subconsulta de otro **SELECT**, **UPDATE** o **DELETE**
- Una subconsulta **SELECT** en una instrucción **INSERT**
- La cláusula **VALUES** de la instrucción **INSERT**
- La cláusula **SET** de la instrucción **UPDATE**
- No se puede utilizar (y siempre hay tentaciones para ello) como valor para la cláusula **DEFAULT** de un campo de tabla.

Su uso más habitual es como apoyo al comando INSERT (en Oracle):

```
INSERT INTO plantas (num, uso)
VALUES ( numeroPlanta.NEXTVAL, 'Suites' );
```

Eliminación de Secuencias

Lo hace el comando DROP SEQUENCE seguido del nombre de la secuencia a borrar.

```
DROP SEQUENCE nombre_secuencia;
```

Modificación de Secuencias

Se pueden modificar las secuencias, pero la modificación sólo puede afectar a los futuros valores de la secuencia, no a los ya utilizados. Sintaxis:

```
ALTER SEQUENCE secuencia
[ INCREMENT BY n ]
[ START WITH n ]
[ { MAXVALUE n | NOMAXVALUE } ]
[ { MINVALUE n | NOMINVALUE } ]
[ { CYCLE | NOCYCLE } ]
```

3.3.9 Creación, Modificación y Eliminación de sinónimos

En Oracle, un sinónimo es un nombre que se asigna a un objeto cualquiera. Normalmente es un nombre menos descriptivo que el original a fin de facilitar la escritura del nombre del objeto en diversas expresiones.

Creación de Sinónimos

Sintaxis:

```
CREATE [PUBLIC] SYNONYM nombre FOR objeto;
```

objeto es el objeto al que se referirá el sinónimo. La cláusula PUBLIC hace que el sinónimo esté disponible para cualquier usuario (sólo se permite utilizar si disponemos de privilegios administrativos). La vista USER_SYNONYMS permite observar la lista de sinónimos del usuario, la vista ALL_SYNONYMS permite mostrar la lista completa de sinónimos de todos los esquemas a los que tenemos acceso.

Eliminación de Sinónimos

```
DROP SYNONYM nombre;
```

Modificación de Sinónimos

Existe una sentencia para la modificación de sinónimos, aunque su uso es escaso. Se trata de la sentencia ALTER SYNONYM.

```
ALTER [PUBLIC] SYNONYM nombre [{COMPILE|EDITABLE|NONEDITABLE}];
```

3.4 ACTIVIDADES PROPUESTAS

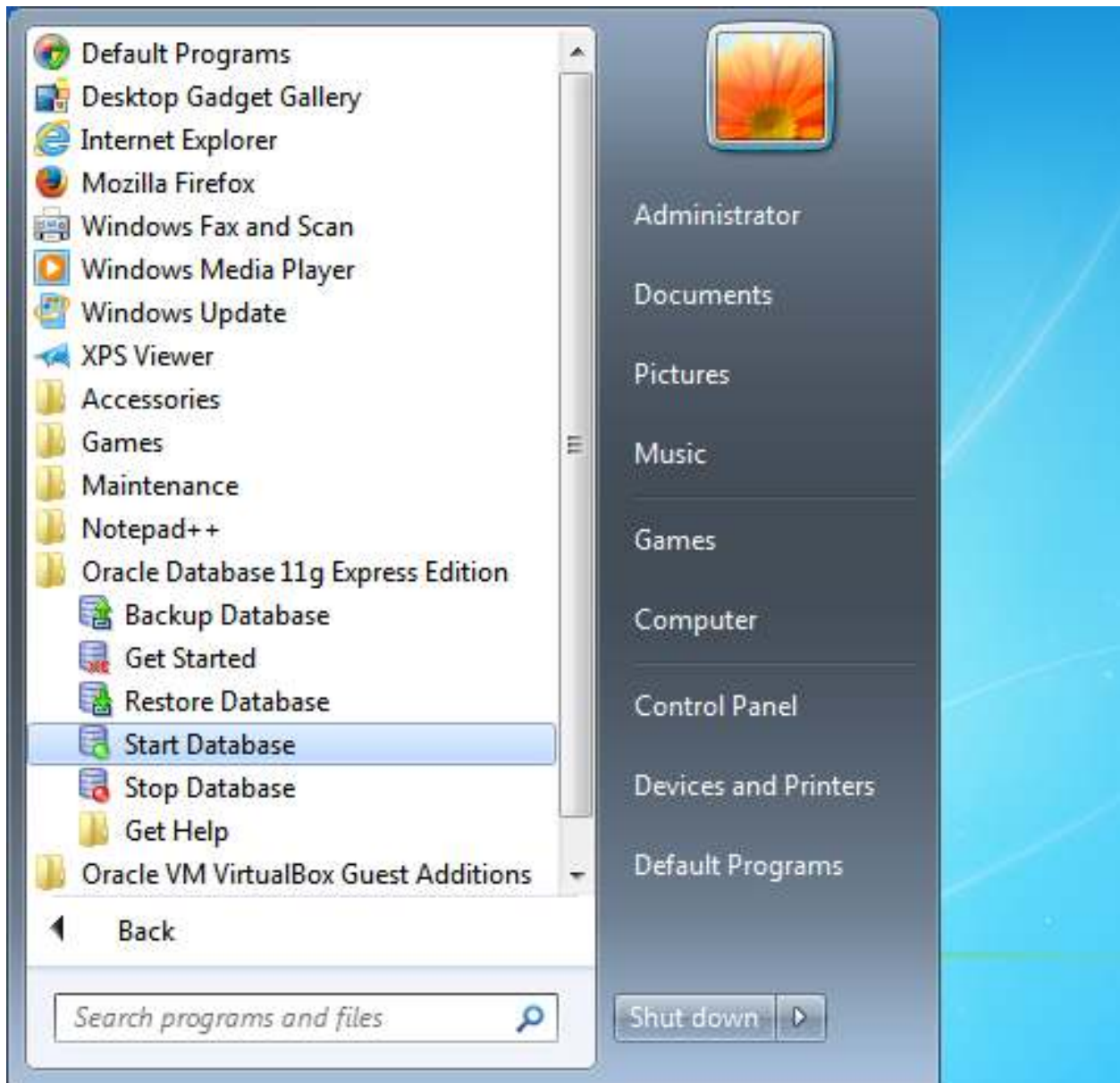
IMPORTANTE

Todos los scripts que aparecen a continuación deben ejecutarse en SQL*Plus.

3.4.1 Cuestiones (I)

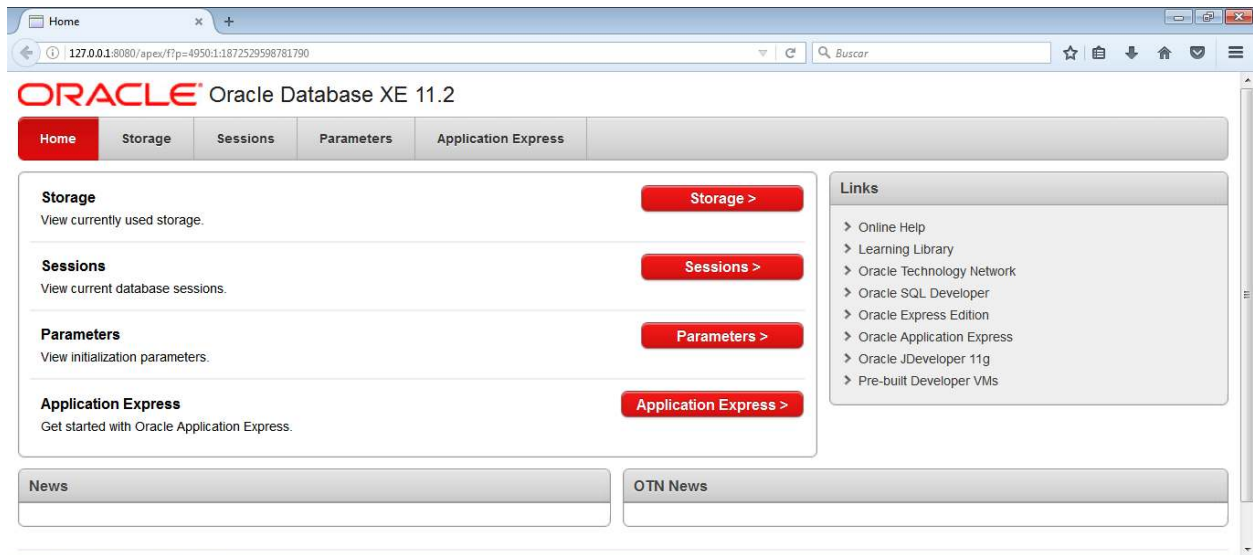
1. Pon un ejemplo de un tipo de dato numérico, otro alfanumérico y otro fecha/hora.
2. ¿Qué diferencia hay entre VARCHAR y CHAR?
3. Realiza la instalación de Oracle Database 11g Express Edition sobre Windows.

4. Una vez instalado dicho Sistema Gestor de Bases de Datos Relacional (RDBMS), lo iniciaremos pulsando en «Start Database».



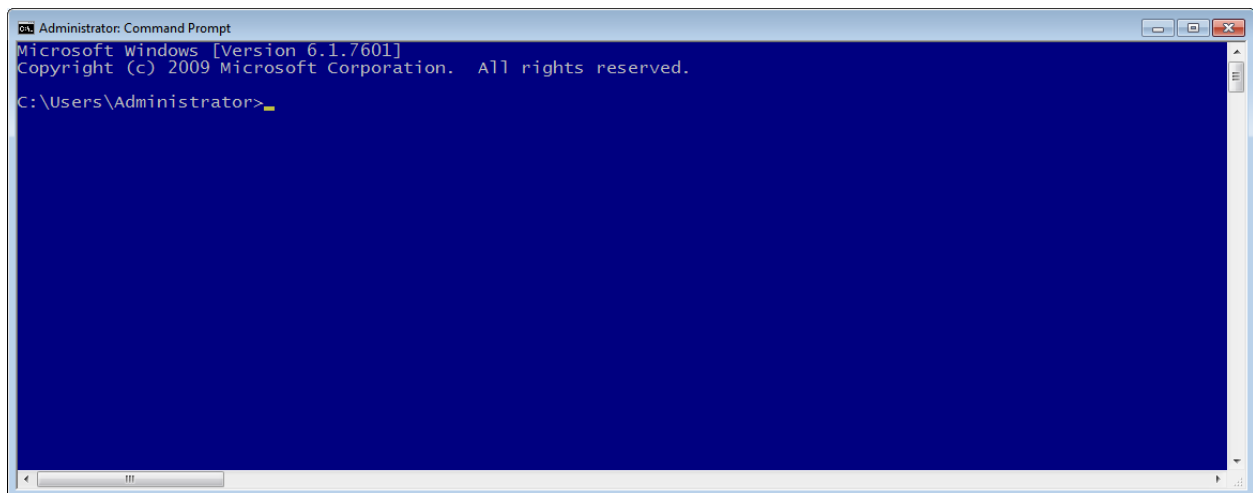
5. Y después iniciamos la interfaz web APEX (APplication EXpress) para trabajar con dicho SGBD.

Para ello pulsamos sobre «Get Started». Y se abrirá el navegador web con esta URL: <http://127.0.0.1:8080/apex/f?p=4950>.

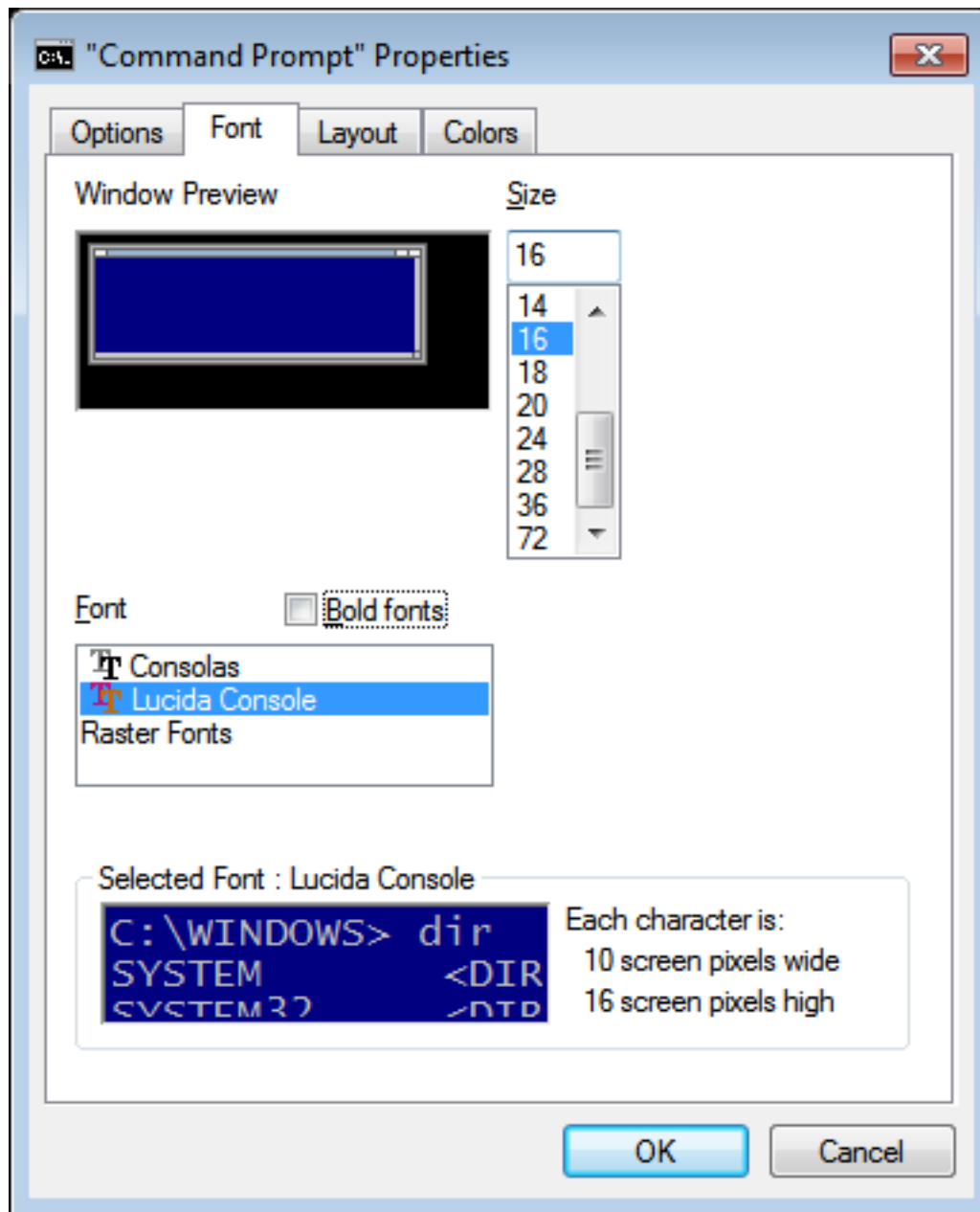


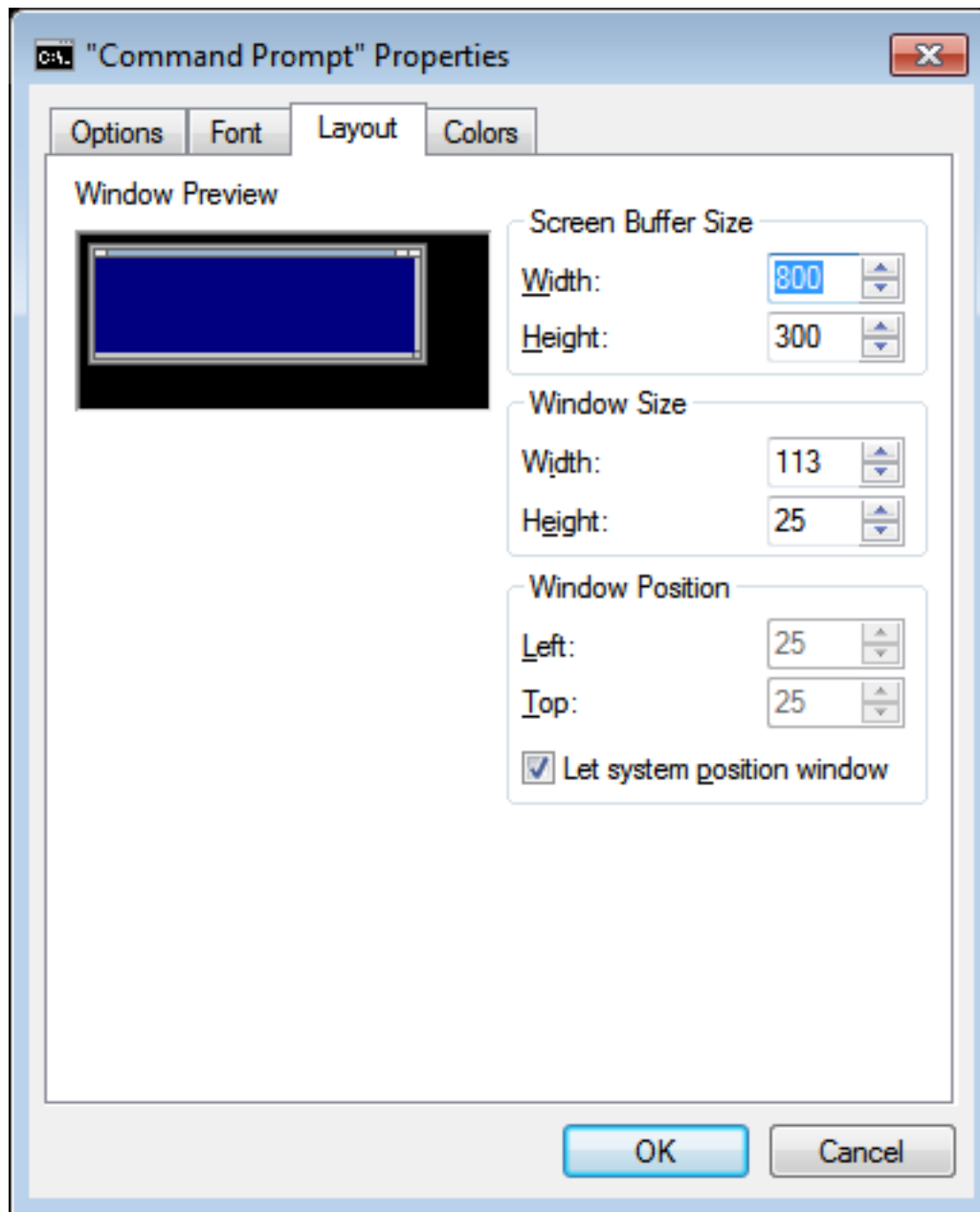
6. A continuación configuraremos el Terminal CMD. En Windows, ejecuta el programa CMD.EXE y realiza la configuración siguiente:

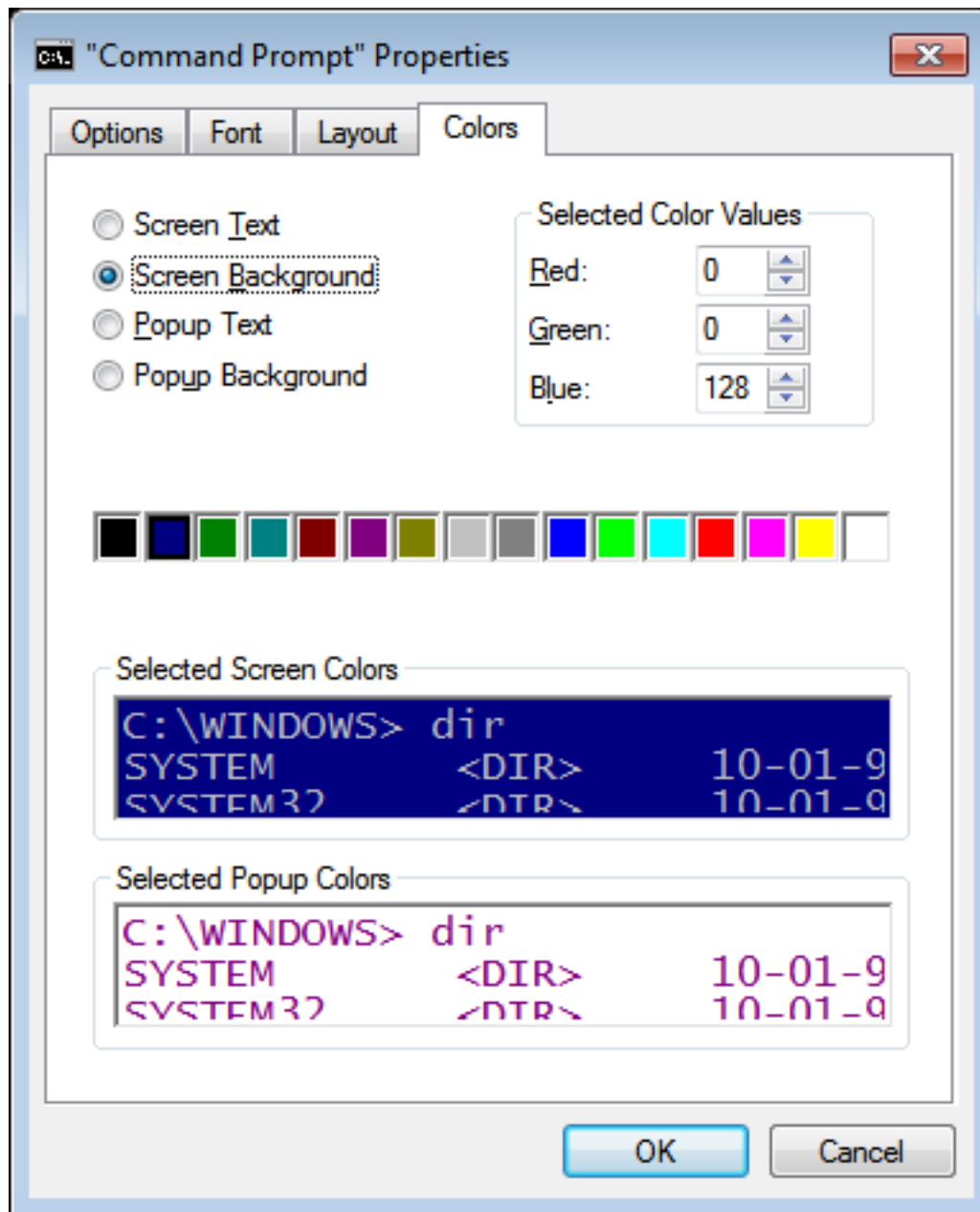
- Pulsa con el botón secundario del ratón en la barra de título y luego haz clic en Propiedades.



- Configura la Fuente, la Disposición y los Colores como se muestra más abajo.



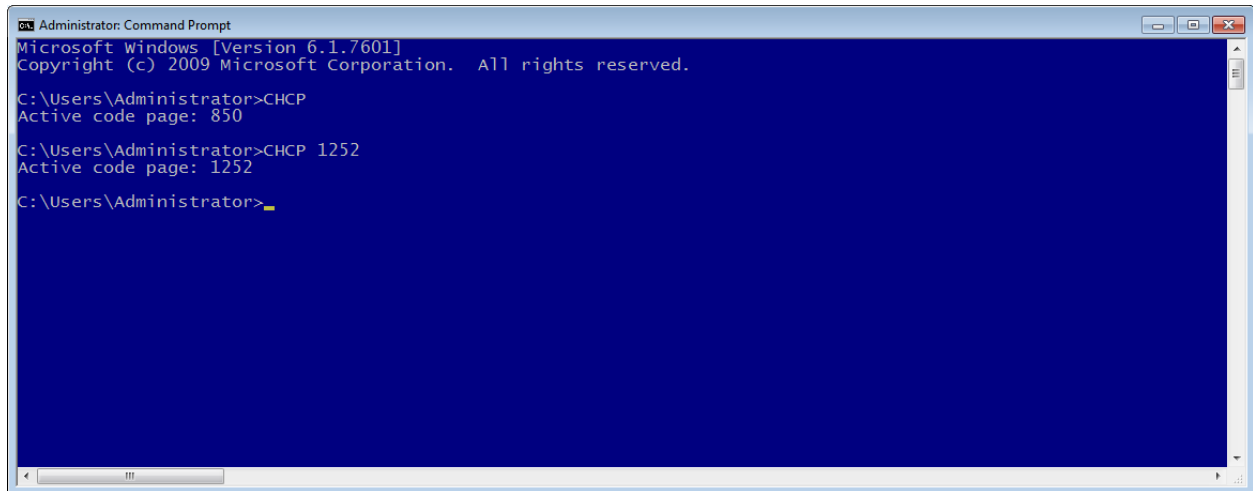




- Pulsa OK.

7. Cambia la «Página de Código» a Windows-1252.

Para que se muestren correctamente las tildes y ciertos caracteres como la letra Ñ y similares deberemos ejecutar el comando CHCP 1252 (CHange CodePage). Esto cambia la CP850 por la CP1252 que soporta el conjunto de caracteres ISO 8859, necesarios por los motivos indicados anteriormente. Esta configuración no se queda guardada, por lo que deberemos ejecutarla cada vez que iniciemos el terminal.



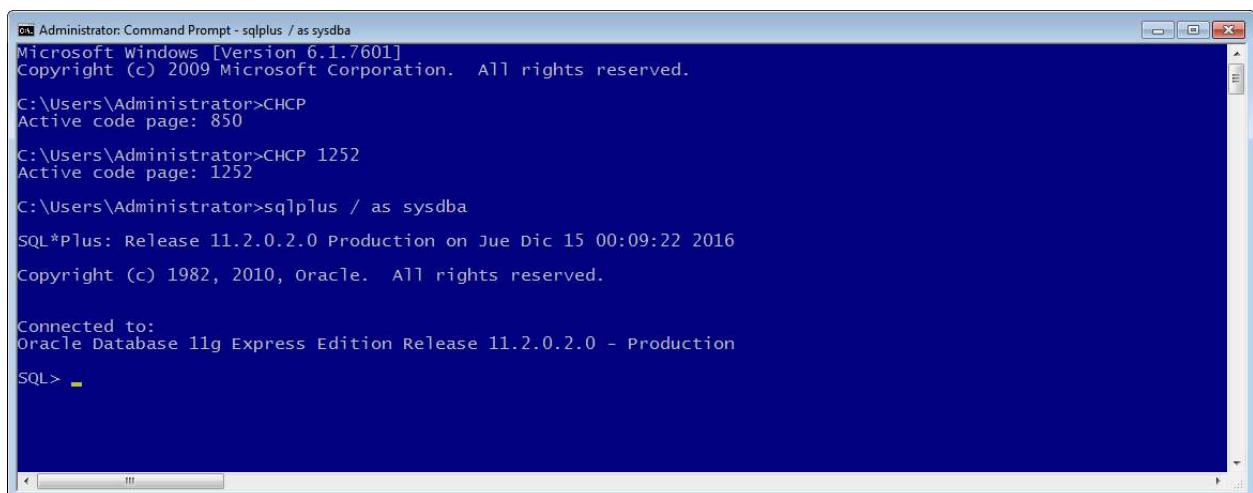
```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>CHCP
Active code page: 850

C:\Users\Administrator>CHCP 1252
Active code page: 1252

C:\Users\Administrator>
```

8. Ya podemos lanzar el cliente SQL*Plus como aparece en la imagen. Lo haremos como SYSDBA (permisos de DataBase Administrator).



```
Administrator: Command Prompt - sqlplus / as sysdba
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>CHCP
Active code page: 850

C:\Users\Administrator>CHCP 1252
Active code page: 1252

C:\Users\Administrator>sqlplus / as sysdba

SQL*Plus: Release 11.2.0.2.0 Production on Jue Dic 15 00:09:22 2016
Copyright (c) 1982, 2010, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - Production

SQL>
```

Para comprobar que SQL*Plus y la base de datos Oracle están funcionando bien, realizaremos lo siguiente. Con Oracle se instala un esquema (usuario) de ejemplo llamado HR (Human Resources). Por defecto está bloqueado. Los desbloqueamos. Le concedemos roles CONNECT y RESOURCE y asignamos una contraseña.

```
sqlplus / as sysdba
...
SQL> alter user hr account unlock;
User altered.
SQL> grant connect,resource to hr identified by "HR";
Grant succeeded.
```

9. Ahora conectamos con usuario/contraseña anterior y vemos las tablas que tiene dicho esquema (usuario).

```
Administrator: Command Prompt - SQLPLUS / AS SYSDBA
SQL> GRANT CONNECT,RESOURCE TO HR IDENTIFIED BY "HR";
Grant succeeded.
SQL> CONN HR/HR
Connected.
SQL> SELECT TABLE_NAME FROM USER_TABLES;

TABLE_NAME
-----
REGIONS
LOCATIONS
DEPARTMENTS
JOBS
EMPLOYEES
JOB_HISTORY
COUNTRIES

7 rows selected.
SQL> _
```

10. Insertamos algunos datos con tildes y caracteres tales como la letra Ñ.

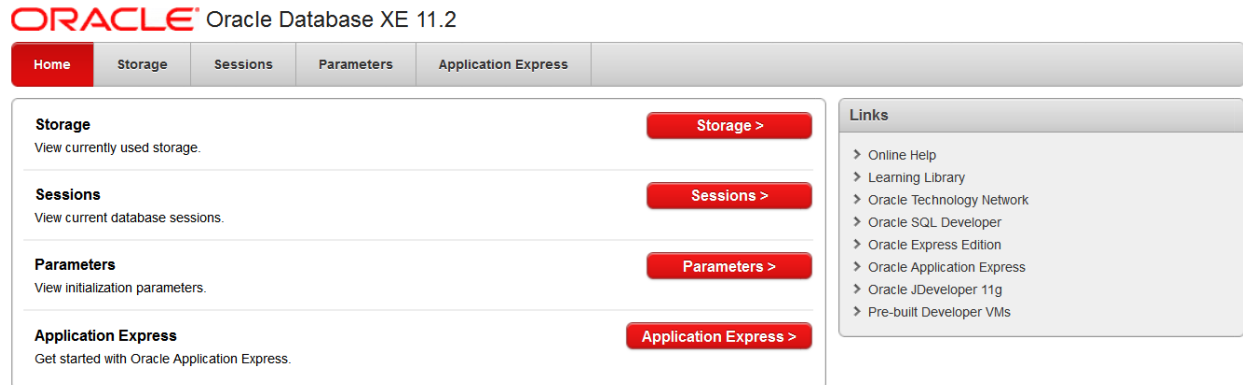
```
Administrator: Command Prompt - SQLPLUS / AS SYSDBA
SQL> SELECT * FROM COUNTRIES;

CO COUNTRY_NAME                                REGION_ID
--
AR Argentina                                    2
AU Australia                                    3
C
:
:
US United States of America                    2
ZM Zambia                                      4
ZW Zimbabwe                                    4

25 rows selected.

SQL> INSERT INTO COUNTRIES VALUES ('ES', 'España (áéíóúÁÉÍÓÚÑÑ)', 1);
1 row created.
SQL> COMMIT;
Commit complete.
SQL>
```

11. Comprobamos ahora la interfaz web APEX. Pulsamos en Application Express. Luego introducimos la clave del usuario SYSTEM que establecimos durante la instalación.



Advertencia: En las contraseñas se distingue entre mayúsculas y minúsculas.

The screenshot shows the Oracle Database XE 11.2 Login page. The page has a header with the word "Login". Below the header, there are two input fields: "Username" and "Password". The "Username" field contains the text "system". The "Password" field contains a series of dots. To the right of the "Password" field is a "Login" button. Below the input fields, there is a line of text: "Login as a database user which has been granted the DBA database role (for example, SYSTEM)."

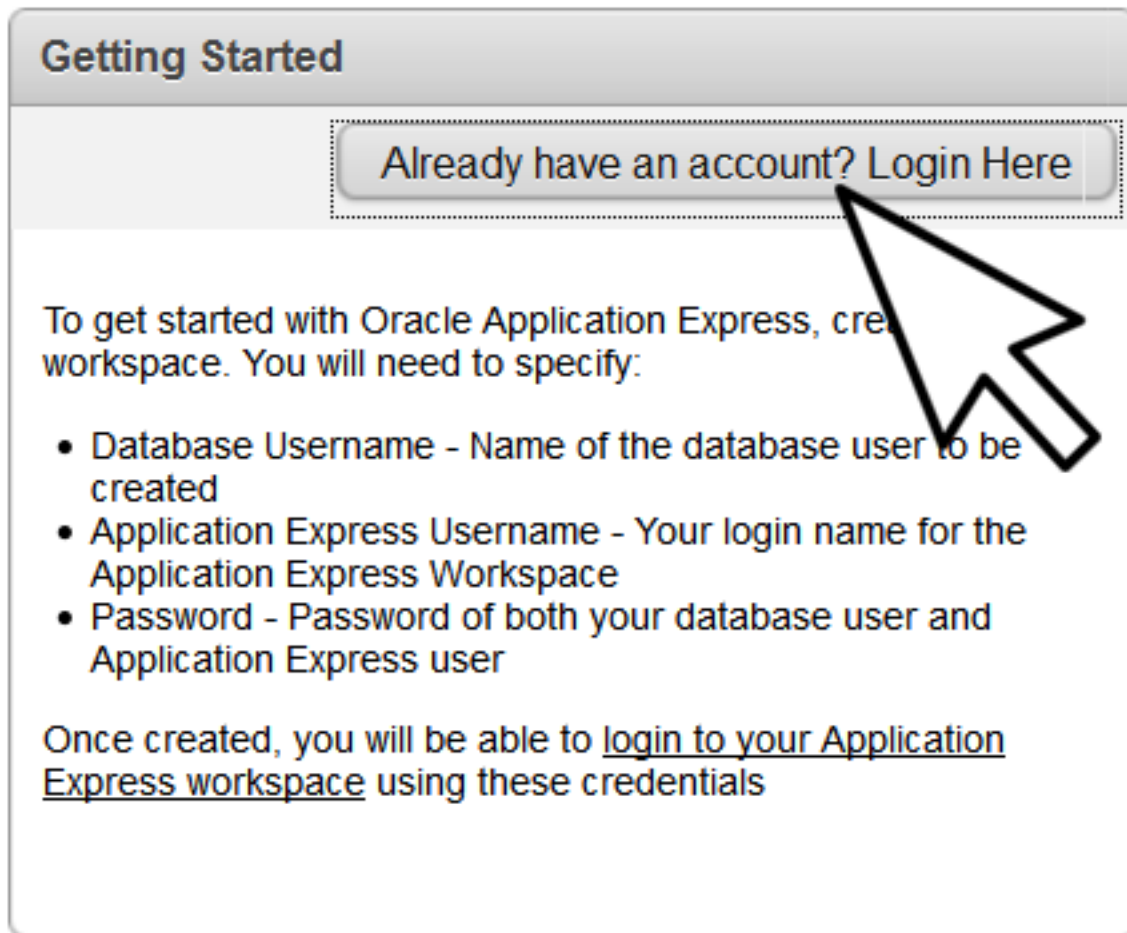
12. Una vez dentro nos creamos un espacio de trabajo (Workspace) como aparece en la siguiente imagen. Luego pulsaremos «Create Workspace».

The screenshot shows the Oracle Database XE 11.2 Application Express interface. The browser address bar shows the URL: 127.0.0.1:8080/apex/f?p=4950:7:1624897185305252::NO::. The page title is 'ORACLE® Oracle Database XE 11.2'. The navigation bar includes 'Home', 'Storage', 'Sessions', 'Parameters', and 'Application Express'. The 'Application Express' tab is selected. Below the navigation bar, there is a breadcrumb 'Home > Oracle Application Express'. The main content area is titled 'Create Application Express Workspace'. It contains a 'Database User' section with radio buttons for 'Create New' and 'Use Existing'. The 'Use Existing' option is selected. Below this, there are four input fields: 'Database Username' (containing 'HR'), 'Application Express Username' (containing 'YO'), 'Password' (masked with dots), and 'Confirm Password' (masked with dots). There are 'Cancel' and 'Create Workspace' buttons. To the right of the dialog box is a 'Getting Started' section with a link 'Already have an account? Login Here' and a list of instructions for creating a workspace.

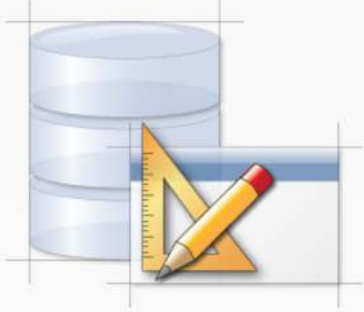
Como usuario y contraseña de APEX, por motivos de comodidad, utilizaremos siempre los siguientes:

- Username: YO
- Password: YO

13. Ya podemos trabajar con el esquema HR en APEX. Para ello pulsamos en «Already have an account? Login Here»



ORACLE Application Express



Enter Application Express workspace and credentials.

Workspace

HR

Username

YO

Password

••

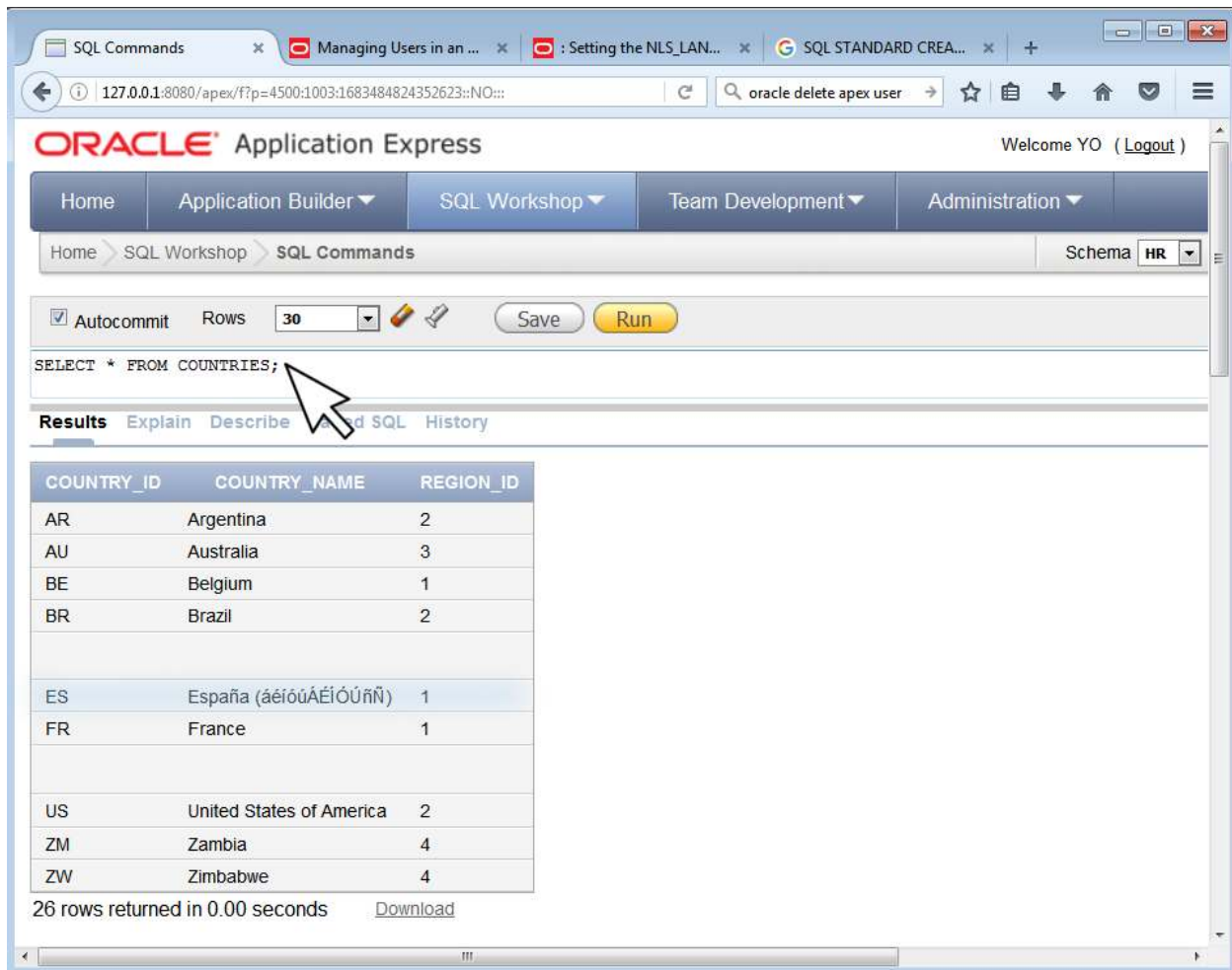
Login

[Click here to learn how to get started](#)

Oracle Application Express is a rapid Web application development tool that lets you share data and create custom applications. Using only a Web browser and limited programming experience, you can develop and deploy powerful applications that are both fast and secure.

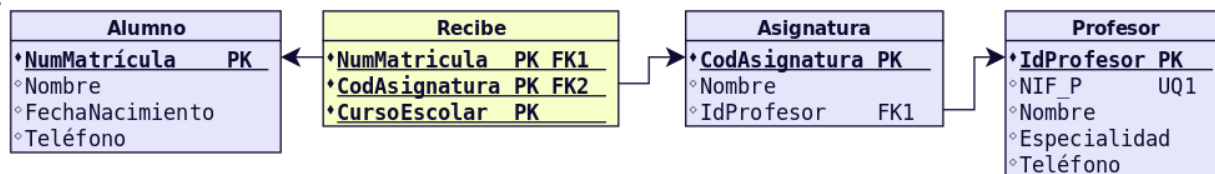
Language: English, Português (Brasil), 

14. Después pulsamos sobre «SQL WorkShop» y «SQL Commands». Realizamos la consulta *SELECT * FROM COUNTRIES;* para comprobar que aparece el registro introducido anteriormente con las tildes correctas.

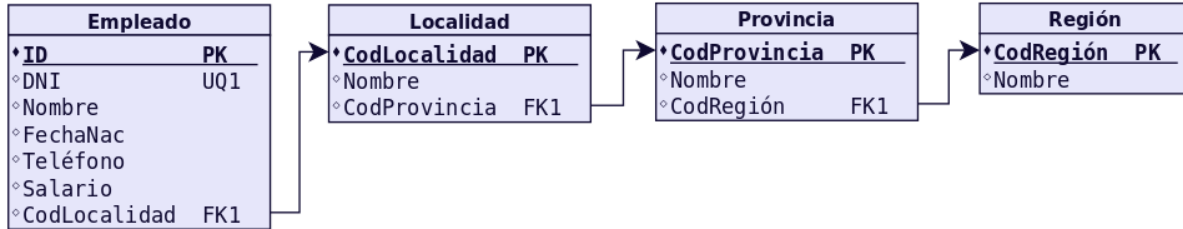


3.4.2 Cuestiones (II)

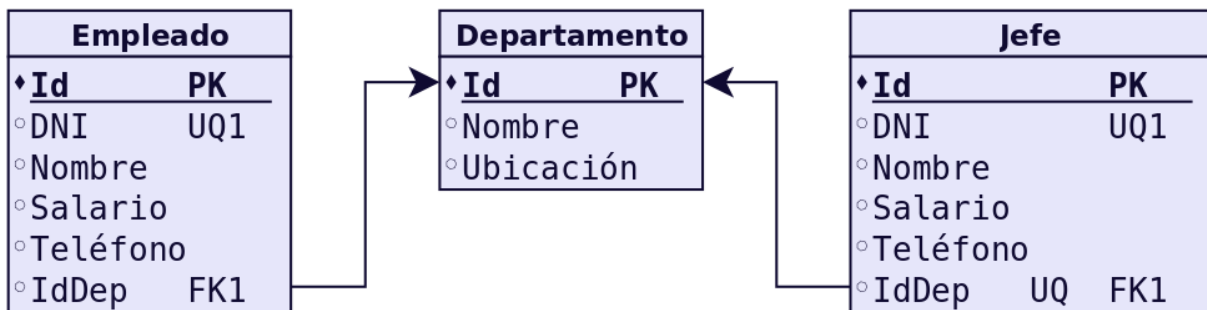
1. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E01, contraseña «E01».



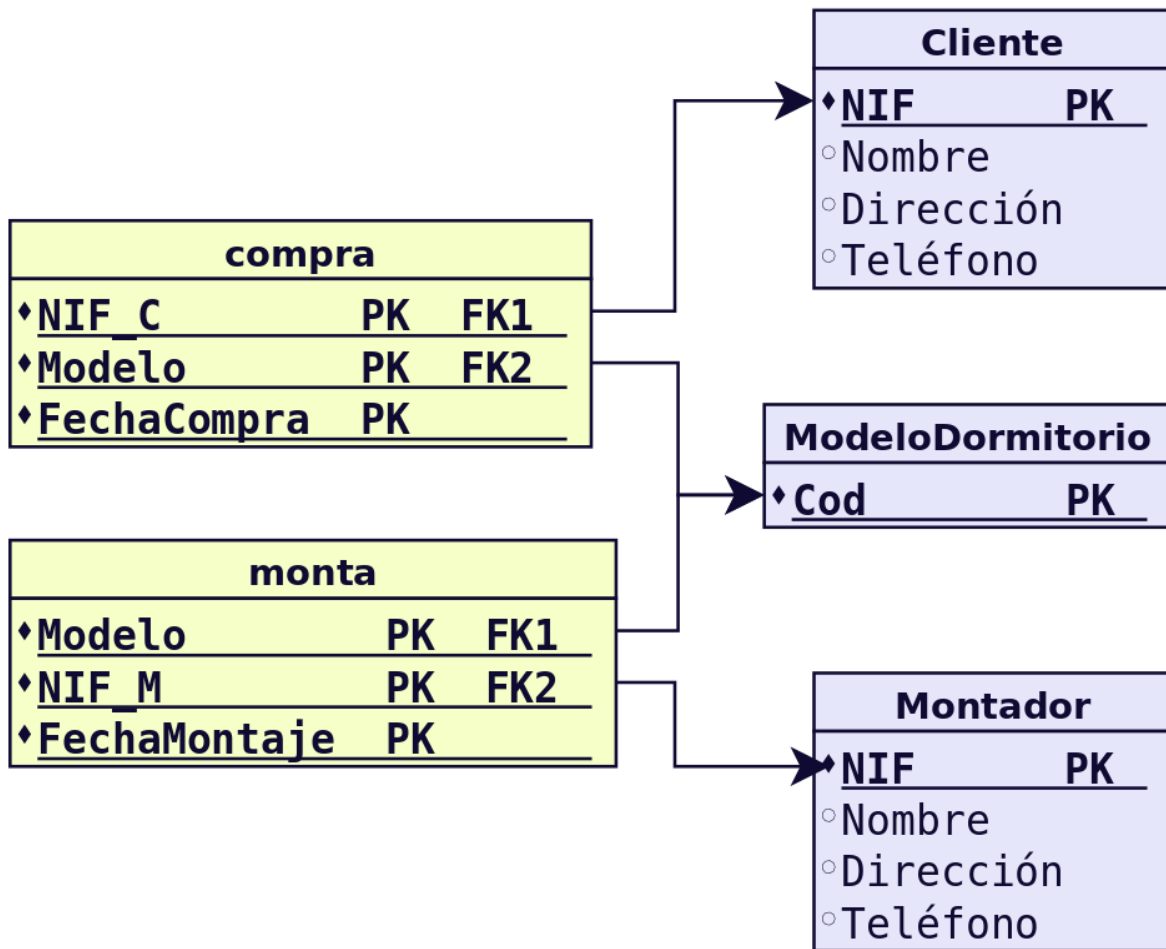
2. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E02, contraseña «E02».



3. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E03, contraseña «E03».

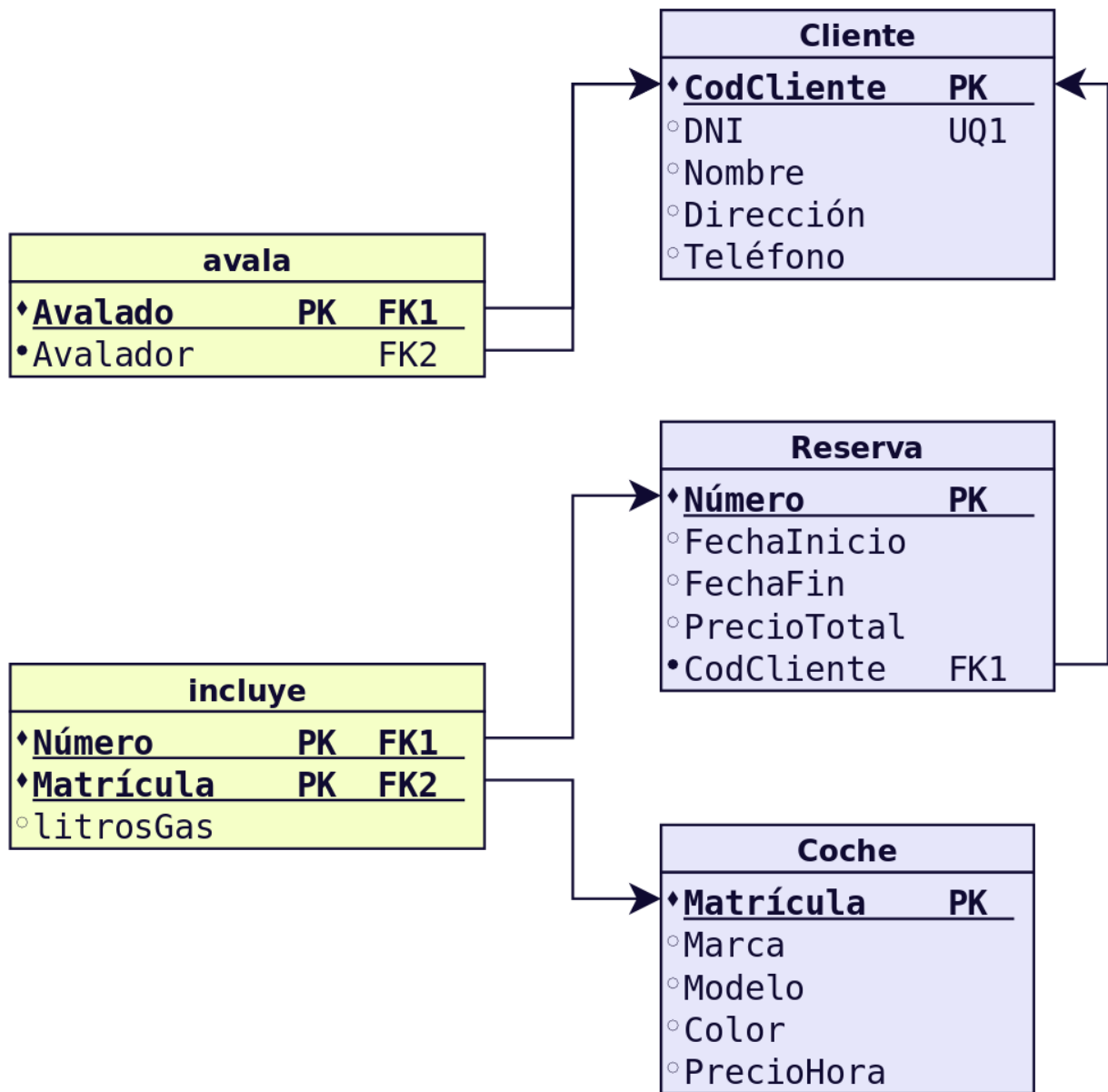


4. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E04, contraseña «E04».



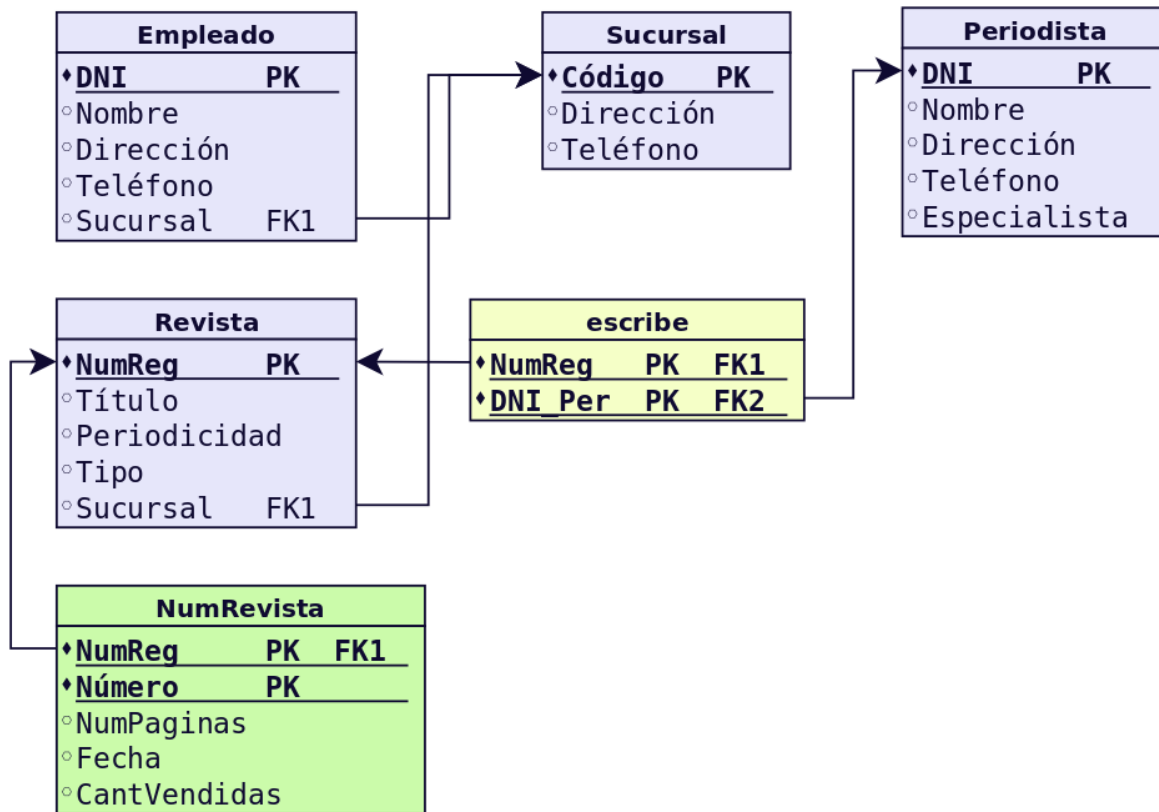
5. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño poniendo nombres a las restricciones. Esquema E05, contraseña «E05».

No pueden ser nulos los siguientes campos: Nombre de Cliente, Marca y Modelo de Coche. Crear una secuencia para el Número de Reserva.



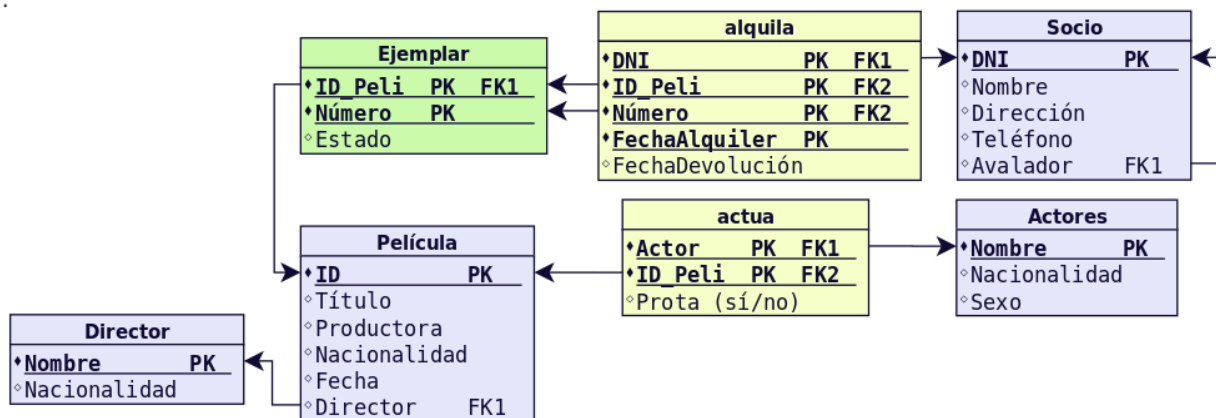
6. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño poniendo nombres a las restricciones. Esquema E06, contraseña «E06».

No pueden ser nulos los siguientes campos: Nombre de Empleado, Nombre de Periodista, Título de Revista. La Periodicidad toma uno de los siguientes valores: Semanal, Quincenal, Mensual, Trimestral o Anual, siendo el valor por defecto Mensual. NumPaginas debe ser mayor que 0.



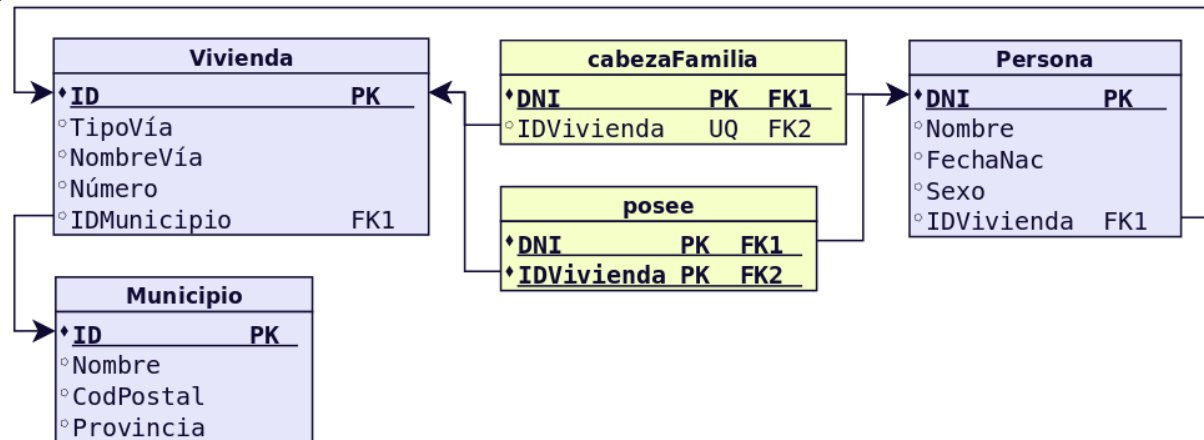
7. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño poniendo nombres a las restricciones. Esquema E07, contraseña «E07».

No pueden ser nulos los siguientes campos: Nombre de Socio, Título de Película. Sexo toma los valores H o M. Por defecto si no se indica nada un actor o actriz no es Protagonista (este campo toma valores S o N). FechaDevolución debe ser mayor que FechaAlquiler.



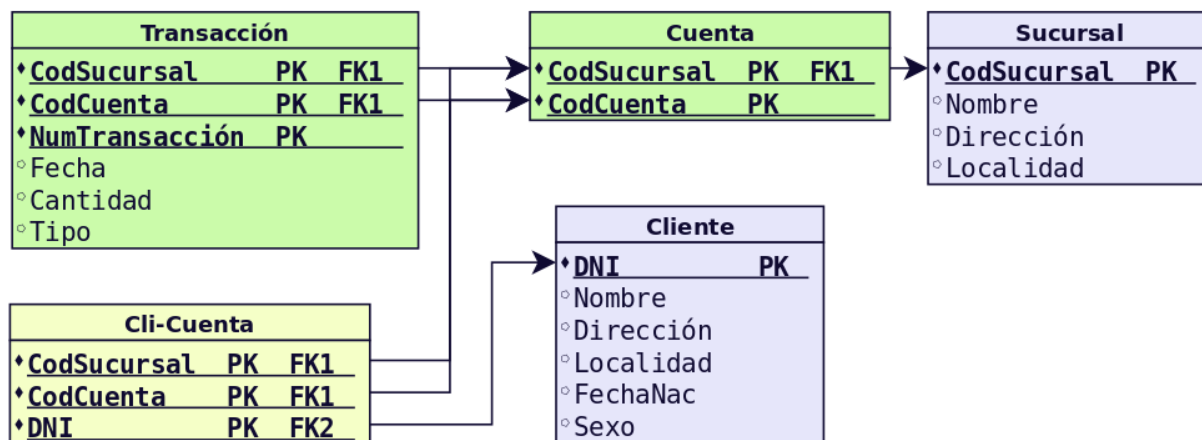
8. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño poniendo nombres a las restricciones. Esquema E08, contraseña «E08».

No pueden ser nulos los siguientes campos: Nombre de Persona, NombreVía, Número de Vivienda, Nombre de Municipio. Sexo toma los valores H o M. Por defecto si no se indica nada el TipoVia es Calle.



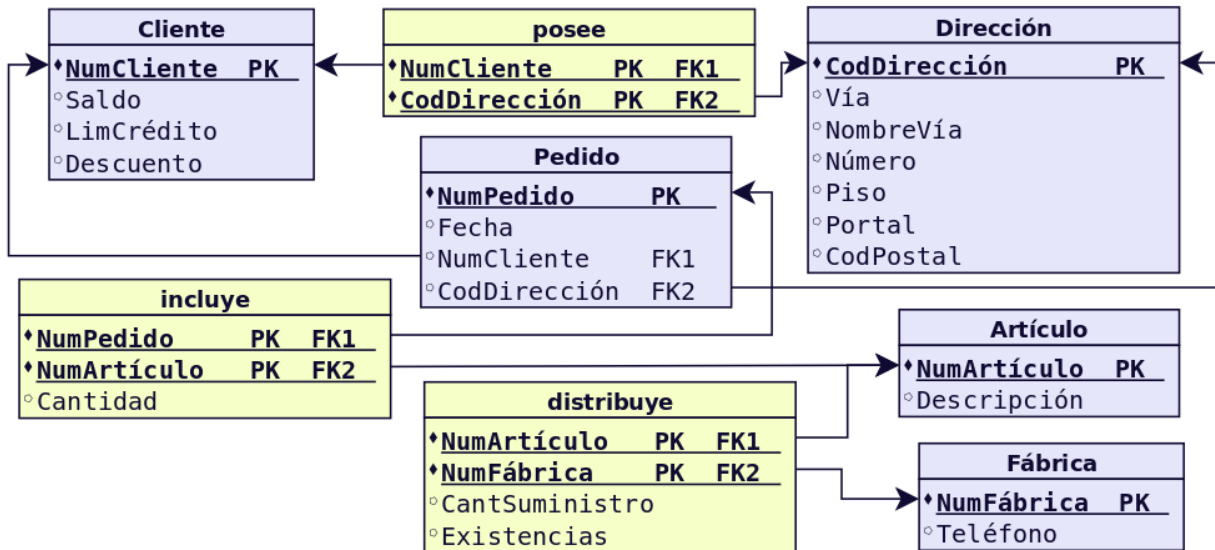
9. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E09, contraseña «E09».

Crea las tablas sin restricciones y añádelas después con el comando ALTER TABLE. Crea índices para los siguientes campos: Nombre de Sucursal, Nombre de Cliente. También para Localidad de Cliente, Localidad de Sucursal. Crea una secuencia que inicie en 1 para CodSucursal.



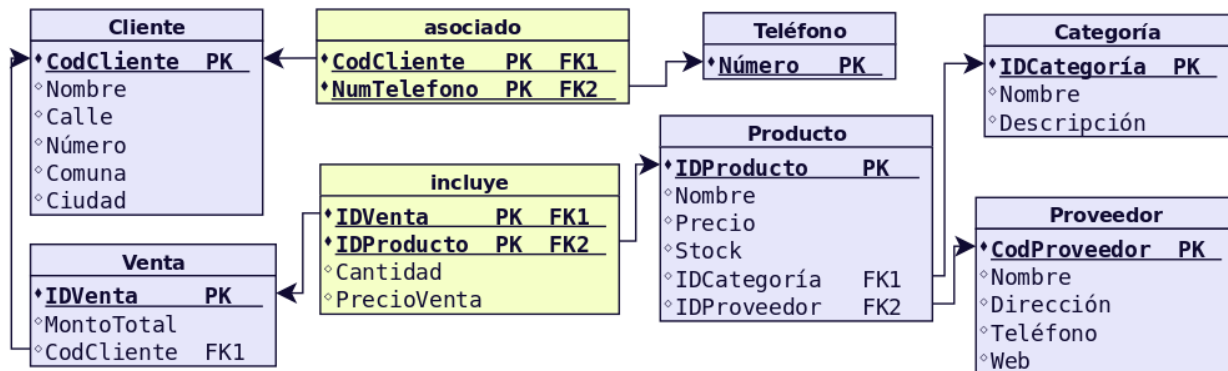
10. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E10, contraseña «E10».

Crea las tablas sin restricciones y añádelas después con el comando ALTER TABLE. Crea secuencias que inicien en 1 para: NumCliente, NumArtículo, NumFabrica y NumPedido.



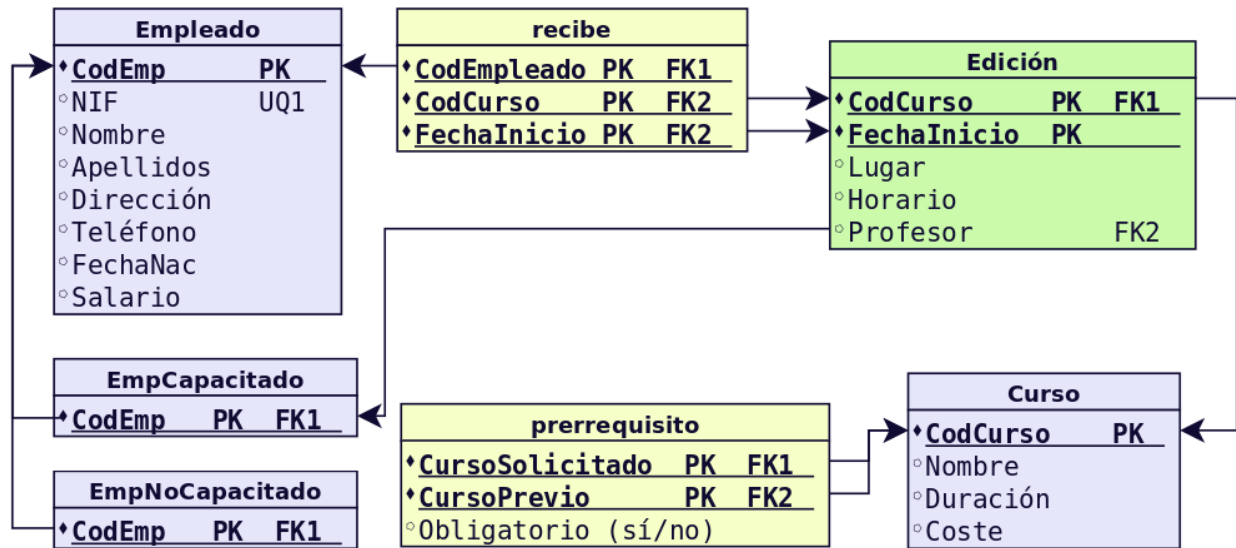
11. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E11, contraseña «E11».

Crea las tablas sin restricciones y añádelas después con el comando ALTER TABLE. Crea índices para los siguientes campos: Nombre de Cliente, Nombre de Categoría, Nombre de Proveedor, Ciudad de Cliente. Crea una secuencia para IDVenta.



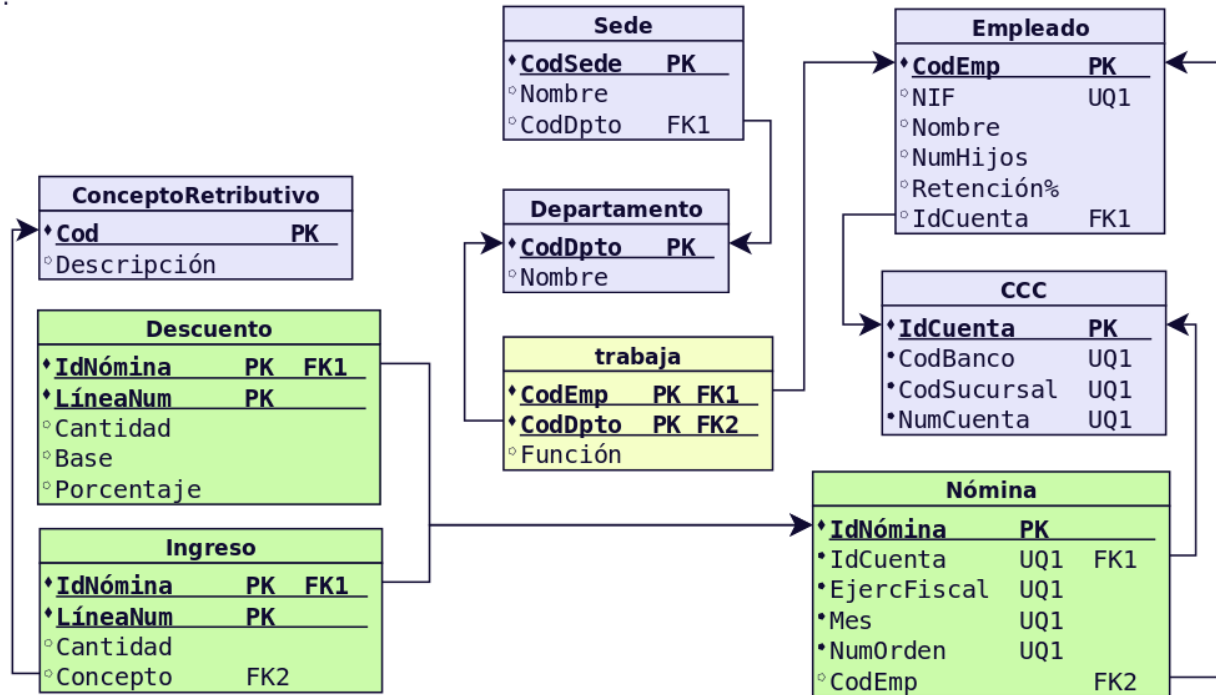
12. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E12, contraseña «E12».

Crea las tablas sin restricciones y añádelas después con el comando ALTER TABLE. Crea índices para los siguientes campos: Apellidos de Empleado, Lugar de Edición, Horario de Edición. Para el indicar si un prerequisite es obligatorio o no utilizamos 'S' o 'N'.



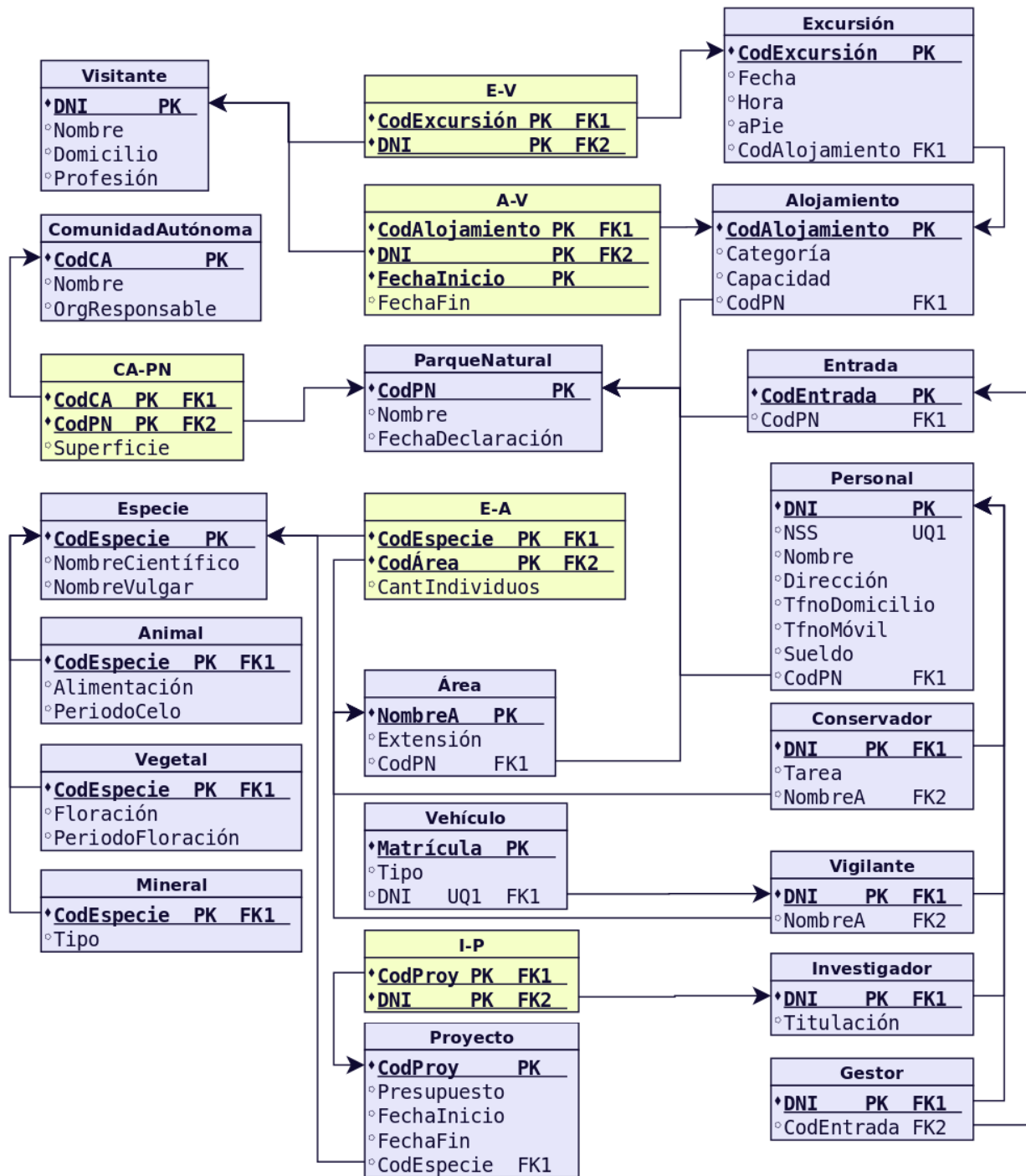
13. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E13, contraseña «E13».

Crea las tablas sin restricciones y añádelas después con el comando ALTER TABLE. Crea una secuencia para ID-Cuenta, otra para IDNomina y otra para LineaNum.



14. Realiza el diseño físico para el siguiente modelo relacional. Asigna el tipo de datos que consideres más adecuado. Realiza el diseño sin poner nombres a las restricciones. Esquema E14, contraseña «E14».

Crea las tablas sin restricciones y añádelas después con el comando ALTER TABLE.



3.4.3 Cuestiones (III)

1. Nombra los distintos tipos de instrucciones DDL que puede haber, distinguiendo el tipo de objeto que se puede crear, borrar o modificar.

2. Realiza un esquema resumen de las cláusulas SQL utilizadas en la modificación de tablas.

3. Desde SQLPlus crea un esquema (usuario) llamado TIENDAS con contraseña TIENDAS. Concédete los roles CONNECT y RESOURCE. Accede con dicho usuario/contraseña.

4. Crear las siguientes tablas de acuerdo con las restricciones que se mencionan:

Tabla 1: TIENDAS

Columna	Tipo de dato
NIF	VARCHAR2(10)
NOMBRE	VARCHAR2(20)
DIRECCION	VARCHAR2(20)
POBLACION	VARCHAR2(20)
PROVINCIA	VARCHAR2(20)
CODPOSTAL	NUMBER(5)

Crear tabla sin restricciones. Después añadir las siguientes restricciones:

- La clave primaria es NIF.
- PROVINCIA ha de almacenarse en mayúscula.
- Cambia la longitud de NOMBRE a 30 caracteres y no nulo.

Tabla 2: FABRICANTES

Columna	Tipo de dato
COD_FABRICANTE	NUMBER(3)
NOMBRE	VARCHAR2(15)
PAIS	VARCHAR2(15)

Restricciones:

- La clave primaria es COD_FABRICANTE.
- Las columnas NOMBRE y PAIS han de almacenarse en mayúscula.

Tabla 3: ARTICULOS

Columna	Tipo de dato	
ARTICULO	VARCHAR2(20)	
COD_FABRICANTE	NUMBER(3)	
PESO	NUMBER(3)	
CATEGORIA	VARCHAR2(10)	
PRECIO_VENTA	NUMBER(6)	2)
PRECIO_COSTO	NUMBER(6)	2)
EXISTENCIAS	NUMBER(5)	

Restricciones:

- La clave primaria está formada por las columnas: ARTICULO, COD_FABRICANTE, PESO y CATEGORIA.
- COD_FABRICANTE es clave ajena que referencia a la tabla FABRICANTES.

- PRECIO_VENTA, PRECIO_COSTO y PESO han de ser > 0 .
- CATEGORIA ha de ser 'Primera', 'Segunda' o 'Tercera'.

Tabla 4: VENTAS

Columna	Tipo de dato
NIF	VARCHAR2(10)
ARTICULO	VARCHAR2(20)
COD_FABRICANTE	NUMBER(3)
PESO	NUMBER(3)
CATEGORIA	VARCHAR2(10)
FECHA_VENTA	DATE
UNIDADES_VENDIDAS	NUMBER(4)

Restricciones:

- La clave primaria está formada por las columnas: NIF, ARTICULO, COD_FABRICANTE, PESO, CATEGORIA y FECHA_VENTA.
- NIF es clave ajena que referencia a la tabla TIENDAS.
- ARTICULO, COD_FABRICANTE, PESO y CATEGORIA es clave ajena que referencia a la tablas ARTICULOS.
- UNIDADES_VENDIDAS han de ser > 0 .
- CATEGORIA ha de ser 'Primera', 'Segunda' o 'Tercera'.

Tabla 5: PEDIDOS

Columna	Tipo de dato
NIF	VARCHAR2(10)
ARTICULO	VARCHAR2(20)
COD_FABRICANTE	NUMBER(3)
PESO	NUMBER(3)
CATEGORIA	VARCHAR2(10)
FECHA_PEDIDO	DATE
UNIDADES_PEDIDAS	NUMBER(4)
EXISTENCIAS	NUMBER(5)

Restricciones:

- La clave primaria está formada por las columnas: NIF, ARTICULO, COD_FABRICANTE, PESO, CATEGORIA y FECHA_PEDIDO.
- NIF es clave ajena que referencia a la tabla TIENDAS.
- ARTICULO, COD_FABRICANTE, PESO y CATEGORIA es clave ajena que referencia a la tablas ARTICULOS.
- UNIDADES_PEDIDAS han de ser > 0 .
- CATEGORIA ha de ser 'Primera', 'Segunda' o 'Tercera'.

5. Añadir una restricción a la tabla TIENDAS para que el NOMBRE de la tienda sea de tipo título (InitCap).

6. Visualizar las CONSTRAINTS definidas para las tablas anteriores.
7. Modificar las columnas de las tablas PEDIDOS y VENTAS para que las UNIDADES_VENDIDAS y las UNIDADES_PEDIDAS puedan almacenar cantidades numéricas de 6 dígitos.
8. Impedir que se den de alta más tiendas en la provincia de “TOLEDO”.
9. Añadir a las tablas PEDIDOS y VENTAS una nueva columna para que almacenen el PVP del artículo.
10. Crear una vista que se llame CONSERJES que contenga el nombre del centro y el nombre de sus conserjes.
11. Crear un sinónimo llamado CONSER asociado a la vista creada antes.
12. Añadir a la tabla PROFESORES una columna llamada COD_ASIG con dos posiciones numéricas.
13. Crear la tabla TASIG con las siguientes columnas: COD_ASIG numérico, 2 posiciones y NOM_ASIG cadena de 20 caracteres.
14. Añadir la restricción de clave primaria a la columna COD_ASIG de la tabla TASIG.
15. Añadir la restricción de clave ajena a la columna COD_ASIG de la tabla PROFESORES.
16. Cambiar de nombre la tabla PROFESORES y llamarla PROFES.
17. Borrar la tabla TASIG.
18. Devolver la tabla PROFESORES a su situación inicial.

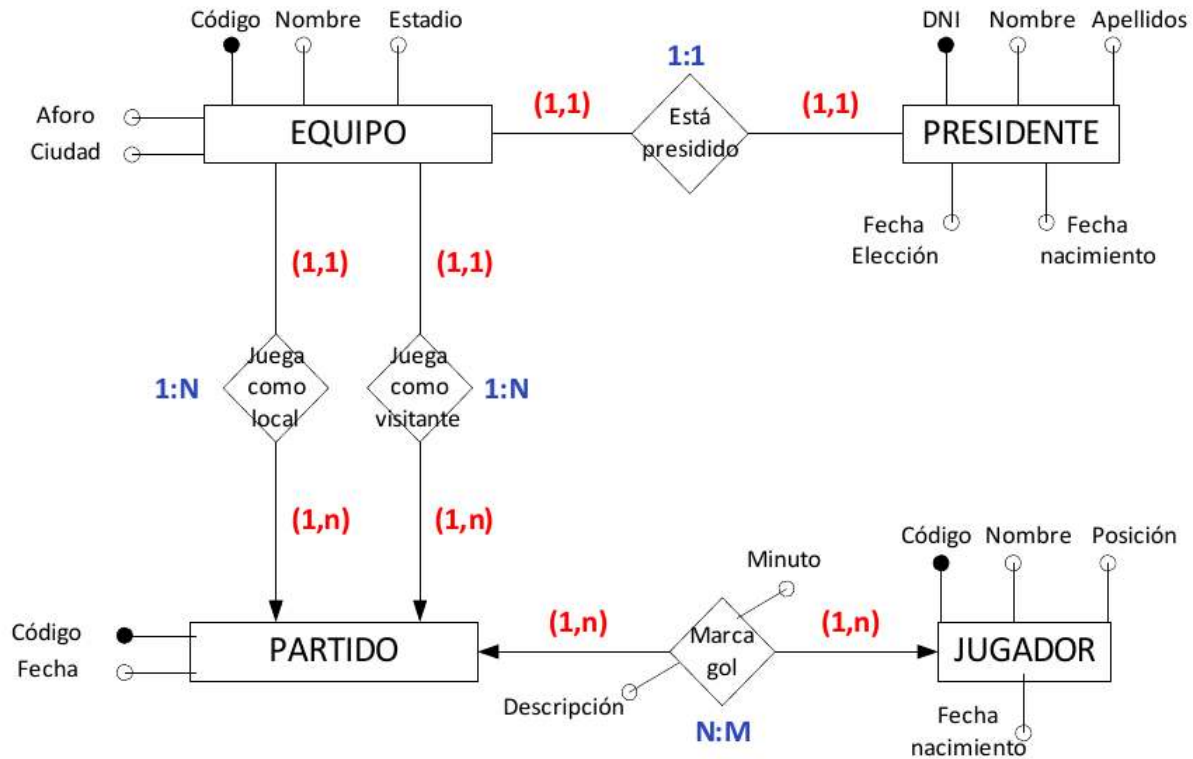
3.4.4 Prácticas

Práctica 1

PLANTEAMIENTO

OBJETIVO: Realizar el diseño físico de bases de datos utilizando asistentes, herramientas gráficas y el lenguaje de definición de datos.

ENUNCIADO: Dado el siguiente esquema E/R:



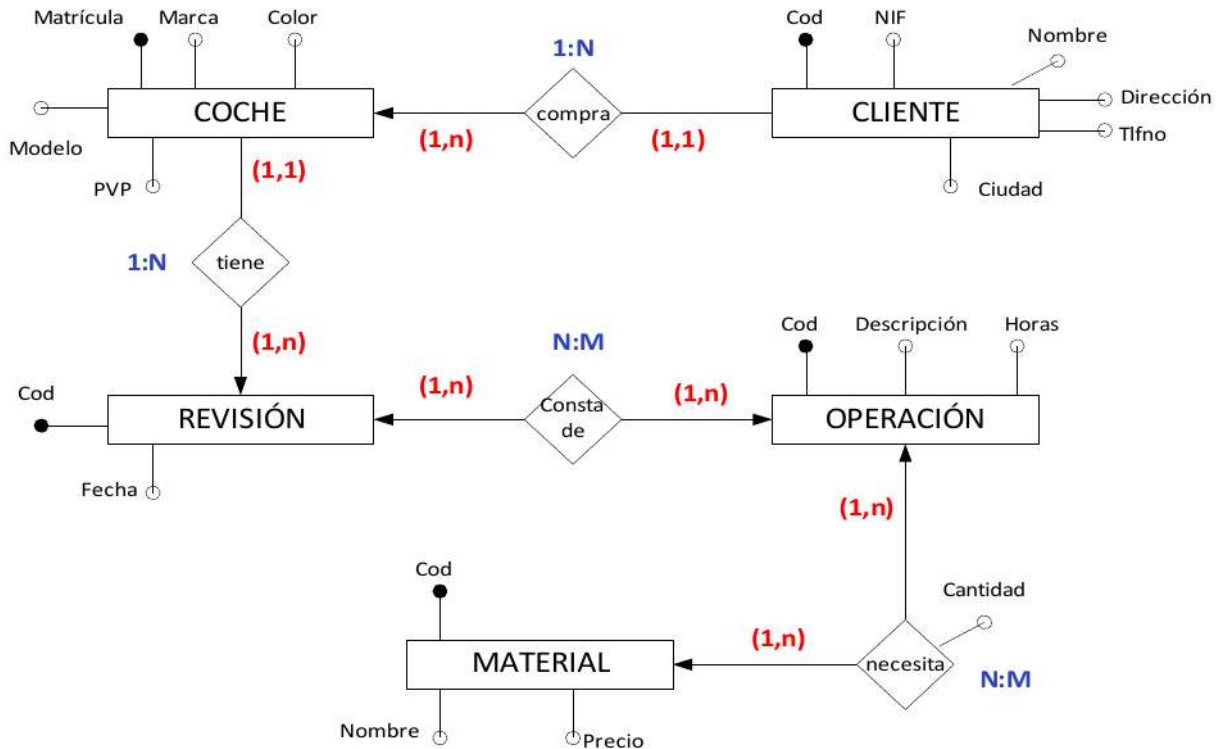
- Obtén el esquema relacional correspondiente.
- Comprueba que está en 3FN.
- Crea las tablas en ORACLE procurando que las columnas tengan el tipo y tamaño adecuado y con las siguientes restricciones:
 - Todas las claves primarias, ajenas y candidatas.
 - No hay partidos en verano (desde el 21/06 al 21/09)
 - Los partidos no pueden durar más de 100 minutos incluyendo el descuento.
 - La posición de un jugador puede ser Portero, Defensa, Centrocampista o Delantero.
 - Los jugadores han de tener como mínimo 16 años en el momento en que se dan de alta en la base de datos.
 - Si no se sabe el aforo de un estadio, se guardará un 0 por defecto.
 - La fecha de un partido es un campo obligatorio.
 - Un partido se juega entre un EquipoLocal y EquipoVisitante (no pueden ser el mismo equipo).
- Una vez creadas las tablas:
 - Añade una columna NumTítulos a la tabla Equipos.
 - Elimina la columna Ciudad.
 - Añade la restricción: Todos los equipos se han fundado después del año 1890.
 - Añade la restricción: La hora de comienzo de los partidos estará entre las 12:00 y las 22:00 horas.

Práctica 2

PLANTEAMIENTO

OBJETIVOS: Realizar el diseño físico de bases de datos utilizando asistentes, herramientas gráficas y el lenguaje de definición de datos.

ENUNCIADO: Dado el siguiente esquema E/R:



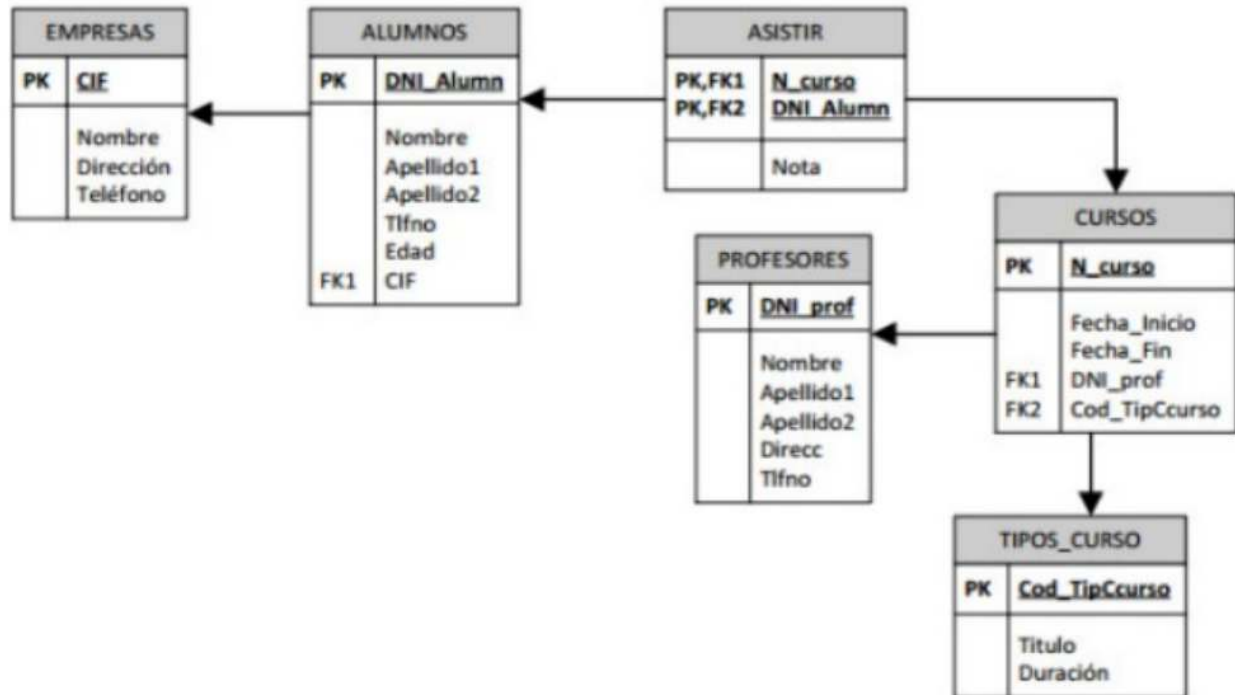
- Obtén el esquema relacional correspondiente.
- Comprueba que está en 3FN.
- Crea las tablas en ORACLE procurando que las columnas tengan el tipo y tamaño adecuado y con las siguientes restricciones:
 - El Color de los coches es verde, rojo o azul.
 - La matrícula está formada por cuatro números y tres letras.
 - Los DNI terminan en letra.
 - Las Horas de mano de obra de una operación nunca pasan de 10.
 - Señala todas las claves primarias, ajenas y candidatas.
 - La cantidad de Piezas por Operación por defecto es 1.
 - La marca y modelo del coche no pueden dejarse en blanco.
 - Los teléfonos empiezan por 6 o por 9.
 - El precio de un coche está entre 10000 y 40000.

Práctica 3

PLANTEAMIENTO

OBJETIVO: Realizar el diseño físico de bases de datos utilizando asistentes, herramientas gráficas y el lenguaje de definición de datos.

ENUNCIADO: Dado el siguiente esquema relacional crear en Oracle el modelo físico, asignando nombre a las restricciones.



CONSULTA DE BASES DE DATOS. LENGUAJE DE MANIPULACIÓN DE DATOS

4.1 INTRODUCCIÓN

A lo largo de esta unidad nos centraremos en la cláusula **SELECT**. Sin duda es el comando más versátil del lenguaje SQL. El comando **SELECT** permite:

- Obtener datos de ciertas columnas de una tabla (proyección).
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios (selección).
- Mezclar datos de tablas diferentes (asociación, join).

4.2 CONSULTAS

Para realizar consultas a una base de datos relacional hacemos uso de la sentencia **SELECT**. La sintaxis básica del comando **SELECT** es la siguiente:

```
SELECT * | {[ DISTINCT ] columna | expresión [[AS] alias ], ...}  
FROM nombre_tabla;
```

Donde:

- *. El asterisco significa que se seleccionan todas las columnas.
- *DISTINCT*. Hace que no se muestren los valores duplicados.
- *columna*. Es el nombre de una columna de la tabla que se desea mostrar.
- *expresión*. Una expresión válida SQL.
- *alias*. Es un nombre que se le da a la cabecera de la columna en el resultado de esta instrucción.

Ejemplos:

```
-- Selección de todos los registros de la tabla CLIENTES
SELECT * FROM CLIENTES;

-- Selección de algunos campos de la tabla CLIENTES
SELECT nombre, apellido1, apellido2 FROM CLIENTES;
```

4.3 CÁLCULOS

4.3.1 Cálculos Aritméticos

Los operadores + (suma), - (resta), * (multiplicación) y / (división), se pueden utilizar para hacer cálculos en las consultas. Cuando se utilizan como expresión en una consulta SELECT, no modifican los datos originales.

Ejemplo:

```
-- Consulta con 3 columnas
SELECT nombre, precio, precio*1.16 FROM ARTICULOS;

-- Ponemos un alias a la tercera columna.
-- Las comillas dobles en el alias hacen que se respeten mayúsculas y minúsculas,
-- de otro modo siempre aparece en mayúsculas
SELECT nombre, precio, precio*1.16 AS "Precio + IVA" FROM ARTICULOS;
```

La prioridad de esos operadores es: tienen más prioridad la multiplicación y división, después la suma y la resta. En caso de igualdad de prioridad, se realiza primero la operación que esté más a la izquierda. Como es lógico se puede evitar cumplir esa prioridad usando paréntesis; el interior de los paréntesis es lo que se ejecuta primero. Cuando una expresión aritmética se calcula sobre valores NULL, el resultado de la expresión es siempre NULL.

4.3.2 Concatenación

El operador || es el de la concatenación. Sirve para unir textos.

Ejemplo:

```
SELECT tipo, modelo, tipo || '-' || modelo "Clave Pieza" FROM PIEZAS;
```

El resultado de esa consulta tendría esta estructura:

TIPO	MODELO	Clave Pieza
AR	6	AR-6
AR	7	AR-7
AR	8	AR-8
AR	9	AR-9
AR	12	AR-12
AR	15	AR-15
AR	20	AR-20
AR	21	AR-21
BI	10	BI-10
BI	20	BI-20
BI	22	BI-22
BI	24	BI-24

4.3.3 Condiciones

Se puede realizar consultas que restrinjan los datos de salida de las tablas. Para ello se utiliza la cláusula **WHERE**. Esta cláusula permite colocar una condición que han de cumplir todos los registros que queremos que se muestren. Las filas que no la cumplan no aparecerán en la ejecución de la consulta.

Nota: Con el comando **SELECT** indicamos las columnas que queremos que aparezcan en nuestra consulta. Con el comando **WHERE** indicamos las filas que queremos que aparezcan en nuestra consulta (serán las que cumplan las condiciones que especifiquemos detrás del **WHERE**).

Ejemplo:

```
-- Tipo y modelo de las piezas cuyo precio es mayor que 3
SELECT tipo, modelo
FROM PIEZAS
WHERE precio > 3;
```

Operadores de comparación

Los operadores de comparación que se pueden utilizar en la cláusula **WHERE** son:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto

Se pueden utilizar tanto para comparar números como para comparar textos y fechas. En el caso de los textos, las comparaciones se hacen en orden alfabético. Sólo que es un orden alfabético estricto. Es decir el orden de los caracteres en la tabla de códigos. Así la letra Ñ y las vocales acentuadas nunca quedan bien ordenadas ya que figuran con códigos más altos. Las mayúsculas figuran antes que las minúsculas (la letra “Z” es menor que la “a”).

Operadores lógicos

Son:

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR, son verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

Ejemplos:

```
-- Personas entre 25 y 50 años
SELECT nombre, apellidos
FROM PERSONAS
WHERE edad >= 25 AND edad <= 50;

-- Personas de más de 60 y menos de 20
SELECT nombre, apellidos
FROM PERSONAS
WHERE edad > 60 OR edad < 20;
```

BETWEEN

El operador BETWEEN nos permite obtener datos que se encuentren entre dos valores determinados (incluyendo los dos extremos).

Ejemplo:

```
-- Selección de las piezas cuyo precio está entre 3 y 8
-- (ambos valores incluidos)
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE precio BETWEEN 3 AND 8;
```

El operador NOT BETWEEN nos permite obtener los valores que son menores (estrictamente) que el más pequeño y mayores (estrictamente) que el más grande. Es decir, no incluye los extremos.

Ejemplo:

```
-- Selección de las piezas cuyo precio sea menor que 3 o mayor que 8
-- (los de precio 3 y precio 8 no estarán incluidos)
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE precio NOT BETWEEN 3 AND 8;
```

IN

El operador IN nos permite obtener registros cuyos valores estén en una lista:

Ejemplo:

```
-- Selección de las piezas cuyo precio sea igual a 3, 5 u 8
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE precio IN ( 3,5,8 );

-- Selección de las piezas cuyo precio no sea igual a 3, 5 u 8
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE precio NOT IN ( 3,5,8 );
```

LIKE

El operador LIKE se usa sobre todo con textos, permite obtener registros cuyo valor en un campo cumpla una condición textual. LIKE utiliza una cadena que puede contener estos símbolos:

Símbolo	Significado
%	Una serie cualquiera de caracteres
_	Un carácter cualquiera

Ejemplos:

```
-- Selección el nombre de las personas que empiezan por A
SELECT nombre
FROM PERSONAS
WHERE nombre LIKE 'A%';

-- Selección el nombre y los apellidos de las personas
cuyo primer apellido sea Jiménez, Giménez, Ximénez
```

(continué en la próxima página)

(proviene de la página anterior)

```
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE apellido1 LIKE '_iménez';
```

Si queremos que en la cadena de caracteres se busquen los caracteres “%” o “_” le antepone el símbolo escape:

Ejemplo:

```
-- Seleccionamos el tipo, el modelo y el precio de las piezas
-- cuyo porcentaje de descuento sea 3%
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE descuento LIKE '3\%' ESCAPE '\';
```

IS NULL

La cláusula IS NULL devuelve “verdadero” si una expresión contiene un nulo, y “Falso” en caso contrario. La cláusula IS NOT NULL devuelve “verdadero” si una expresión NO contiene un nulo, y “Falso” en caso contrario.

Ejemplos:

```
-- Devuelve el nombre y los apellidos de las personas que NO tienen teléfono
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE telefono IS NULL;

-- Devuelve el nombre y los apellidos de las personas que SÍ tienen teléfono
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE telefono IS NOT NULL;
```

Precedencia de operadores

A veces las expresiones que se producen en los SELECT son muy extensas y es difícil saber que parte de la expresión se evalúa primero, por ello se indica la siguiente tabla de precedencia:

Orden de precedencia	Operador
1	*(Multiplicar) / (dividir)
2	+ (Suma) - (Resta)
3	(Concatenación)
4	Comparaciones (>, <, !=, ...)
5	IS [NOT] NULL, [NOT]LIKE, IN
6	NOT
7	AND
8	OR

4.4 SUBCONSULTAS

Se trata de una técnica que permite utilizar el resultado de una tabla SELECT en otra consulta SELECT. Permite solucionar problemas en los que el mismo dato aparece dos veces. La sintaxis es:

```
SELECT lista_expresiones
FROM tablas
WHERE expresión OPERADOR
      ( SELECT lista_expresiones
        FROM tablas );
```

Se puede colocar el SELECT dentro de las cláusulas WHERE, HAVING o FROM. El operador puede ser >, <, >=, <=, !=, = o IN.

Ejemplo:

```
-- Obtiene los empleados cuyas pagas sean inferiores a lo que gana Martina.
SELECT nombre_empleado, paga
FROM EMPLEADOS
WHERE paga < ( SELECT paga
               FROM EMPLEADOS
               WHERE nombre_empleado='Martina');
```

Lógicamente el resultado de la subconsulta debe incluir el campo que estamos analizando.

Se pueden realizar esas subconsultas las veces que haga falta:

```
SELECT nombre_empleado, paga
FROM EMPLEADOS
WHERE paga < ( SELECT paga
               FROM EMPLEADOS
               WHERE nombre_empleado='Martina')
AND   paga > ( SELECT paga
               FROM EMPLEADOS
               WHERE nombre_empleado='Luis');
```

La última consulta obtiene los empleados cuyas pagas estén entre lo que gana Luis y lo que gana Martina. Una subconsulta que utilice los valores >, <, >=, ... tiene que devolver un único valor, de otro modo ocurre un error. Pero a veces se utilizan consultas del tipo: mostrar el sueldo y nombre de los empleados cuyo sueldo supera al de cualquier empleado del departamento de ventas.

La subconsulta necesaria para ese resultado mostraría los sueldos del departamento de ventas. Pero no podremos utilizar un operador de comparación directamente ya que compararíamos un valor con muchos valores. La solución a esto es utilizar instrucciones especiales entre el operador y la consulta. Esas instrucciones son:

Instrucción	Significado
ANY	Compara con cualquier registro de la subconsulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta
ALL	Compara con todos los registros de la consulta. La instrucción resulta cierta si es cierta toda comparación con los registros de la subconsulta
IN	No usa comparador, ya que sirve para comprobar si un valor se encuentra en el resultado de la subconsulta
NOT IN	Comprueba si un valor no se encuentra en una subconsulta

Ejemplo:

```
-- Obtiene el empleado que más cobra.
SELECT nombre, sueldo
FROM EMPLEADOS
WHERE sueldo >= ALL ( SELECT sueldo
                     FROM EMPLEADOS );
```

Ejemplo:

```
-- Obtiene los nombres de los empleados cuyos DNI están en la tabla de directivos.
-- Es decir, obtendrá el nombre de los empleados que son directivos.
SELECT nombre, sueldo
FROM EMPLEADOS
WHERE DNI IN ( SELECT DNI
              FROM DIRECTIVOS );
```

4.5 ORDENACIÓN

El orden inicial de los registros obtenidos por un SELECT guarda una relación con al orden en el que fueron introducidos. Para ordenar en base a criterios más interesantes, se utiliza la cláusula ORDER BY. En esa cláusula se coloca una lista de campos que indica la forma de ordenar. Se ordena primero por el primer campo de la lista, si hay coincidencias por el segundo, si ahí también las hay por el tercero, y así sucesivamente. Se puede colocar las palabras ASC O DESC (por defecto se toma ASC). Esas palabras significan en ascendente (de la A a la Z, de los números pequeños a los grandes) o en descendente (de la Z a la a, de los números grandes a los pequeños) respectivamente.

Sintaxis completa de SELECT:

```
SELECT * | {[DISTINCT] columna | expresión [[AS] alias], ... }
FROM nombre_tabla
[WHERE condición]
[ORDER BY columna1[{ASC|DESC}]] [, columna2[{ASC|DESC}]] ...;
```

Ejemplo:

```
-- Devuelve el nombre y los apellidos
-- de las personas que tienen teléfono, ordenados por
-- apellido1, luego por apellido2 y finalmente por nombre
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE telefono IS NOT NULL
ORDER BY apellido1, apellido2, nombre;
```

4.6 FUNCIONES

Oracle incorpora una serie de instrucciones que permiten realizar cálculos avanzados, o bien facilitar la escritura de ciertas expresiones. Todas las funciones reciben datos para poder operar (parámetros) y devuelven un resultado (que depende de los parámetros enviados a la función. Los argumentos se pasan entre paréntesis:

```
NOMBRE_FUNCIÓN [ ( parámetro1 [, parámetros2] ... ) ];
```

Si una función no precisa parámetros (como SYSDATE) no hace falta colocar los paréntesis.

Las funciones pueden ser de dos tipos:

- Funciones que operan con una sola fila
- Funciones que operan con varias filas.

En este apartado, solo veremos las primeras. Más adelante se estudiarán las que operan sobre varias filas.

Nota: Oracle proporciona una tabla llamada **DUAL** con la que se permiten hacer pruebas. Esa tabla tiene un solo campo (llamado DUMMY) y una sola fila de modo que es posible hacer pruebas.

Por ejemplo la consulta:

```
SELECT SQRT(5) FROM DUAL;
```

Muestra una tabla con el contenido de ese cálculo (la raíz cuadrada de 5). DUAL es una tabla interesante para hacer pruebas.

4.6.1 Funciones de caracteres

Para convertir el texto a mayúsculas o minúsculas:

Función	Descripción
LOWER (texto)	Convierte el texto a minúsculas (funciona con los caracteres españoles)
UPPER (texto)	Convierte el texto a mayúsculas
INITCAP (texto)	Coloca la primera letra de cada palabra en mayúsculas

En la siguiente tabla mostramos las llamadas funciones de transformación:

Función	Descripción
RTRIM (texto)	Elimina los espacios a la derecha del texto
LTRIM (texto)	Elimina los espacios a la izquierda que posea el texto
TRIM (texto)	Elimina los espacios en blanco a la izquierda y la derecha del texto y los espacios dobles del interior.
TRIM (caracteres FROM texto)	Elimina del texto los caracteres indicados. Por ejemplo TRIM ('h' FROM nombre) elimina las haches de la columna <i>nombre</i> que estén a la izquierda y a la derecha
SUBSTR (texto,n[,m])	Obtiene los <i>m</i> siguientes caracteres del texto a partir de la posición <i>n</i> (si <i>m</i> no se indica se cogen desde <i>n</i> hasta el final).
LENGTH (texto)	Obtiene el tamaño del texto
INSTR (texto, textoBuscado [,posInicial [, nAparición]])	Obtiene la posición en la que se encuentra el texto buscado en el texto inicial. Se puede empezar a buscar a partir de una posición inicial concreta e incluso indicar el número de aparición del texto buscado. Ejemplo, si buscamos la letra <i>a</i> y ponemos 2 en <i>nAparición</i> , devuelve la posición de la segunda letra <i>a</i> del texto). Si no lo encuentra devuelve 0
REPLACE (texto, textoABuscar, [textoReemplazo])	Buscar el texto a buscar en un determinado texto y lo cambia por el indicado como texto de reemplazo. Si no se indica texto de reemplazo, entonces esta función elimina el texto a buscar
LPAD (texto, anchuraMáxima, [caracterDeRelleno]) RPAD (texto, anchuraMáxima, [caracterDeRelleno])	Rellena el texto a la izquierda (LPAD) o a la derecha (RPAD) con el carácter indicado para ocupar la anchura indicada. Si el texto es más grande que la anchura indicada, el texto se recorta. Si no se indica carácter de relleno se rellenará el espacio marcado con espacios en blanco.
REVERSE (texto)	Invierte el texto (le da la vuelta)

4.6.2 Funciones numéricas

Funciones para redondear el número de decimales o redondear a números enteros:

Función	Descripción
ROUND (n,decimales)	Redondea el número al siguiente número con el número de decimales indicado más cercano. ROUND (8.239,2) devuelve 8.24
TRUNC (n,decimales)	Los decimales del número se cortan para que sólo aparezca el número de decimales indicado

En el siguiente cuadro mostramos la sintaxis SQL de funciones matemáticas habituales:

Función	Descripción
MOD (n1,n2)	Devuelve el resto resultado de dividir n1 entre n2
POWER (valor,exponente)	Eleva el valor al exponente indicado
SQRT (n)	Calcula la raíz cuadrada de n
SIGN (n)	Devuelve 1 si n es positivo, cero si vale cero y -1 si es negativo
ABS (n)	Calcula el valor absoluto de n
EXP (n)	Calcula eⁿ , es decir el exponente en base e del número n
LN (n)	Logaritmo neperiano de n
LOG (n)	Logaritmo en base 10 de n
SIN (n)	Calcula el seno de n (n tiene que estar en radianes)
COS (n)	Calcula el coseno de n (n tiene que estar en radianes)
TAN (n)	Calcula la tangente de n (n tiene que estar en radianes)
ACOS (n)	Devuelve en radianes el arco coseno de n
ASIN (n)	Devuelve en radianes el arco seno de n
ATAN (n)	Devuelve en radianes el arco tangente de n
SINH (n)	Devuelve el seno hiperbólico de n
COSH (n)	Devuelve el coseno hiperbólico de n
TANH (n)	Devuelve la tangente hiperbólica de n

4.6.3 Funciones de fecha

Las fechas se utilizan muchísimo en todas las bases de datos. Oracle proporciona dos tipos de datos para manejar fechas, los tipos DATE y TIMESTAMP. En el primer caso se almacena una fecha concreta (que incluso puede contener la hora), en el segundo caso se almacena un instante de tiempo más concreto que puede incluir incluso fracciones de segundo. Hay que tener en cuenta que a los valores de tipo fecha se les pueden sumar números y se entendería que esta

suma es de días. Si tiene decimales entonces se suman días, horas, minutos y segundos. La diferencia entre dos fechas también obtiene un número de días.

Funciones para obtener la fecha y hora actual

Función	Descripción
SYSDATE	Obtiene la fecha y hora actuales
SYSTIMESTAMP	Obtiene la fecha y hora actuales en formato TIMESTAMP

Funciones para calcular fechas:

Función	Descripción
ADD_MONTHS (fecha,n)	Añade a la fecha el número de meses indicado por <i>n</i>
MONTHS_BETWEEN (fecha1, fecha2)	Obtiene la diferencia en meses entre las dos fechas (puede ser decimal)
NEXT_DAY (fecha,día)	Indica cual es el día que corresponde a añadir a la fecha el día indicado. El día puede ser el texto ' <i>Lunes</i> ', ' <i>Martes</i> ', ' <i>Miércoles</i> ',... (si la configuración está en español) o el número de día de la semana (1=lunes, 2=martes,...)
LAST_DAY (fecha)	Obtiene el último día del mes al que pertenece la fecha. Devuelve un valor DATE
EXTRACT (valor FROM fecha)	Extrae un valor de una fecha concreta. El valor puede ser day (día), month (mes), year (año), etc.
GREATEST (fecha1, fecha2,..)	Devuelve la fecha más moderna la lista
LEAST (fecha1, fecha2,..)	Devuelve la fecha más antigua la lista
ROUND (fecha [, 'formato']	Redondea la fecha al valor de aplicar el formato a la fecha. El formato puede ser: ' YEAR ' Hace que la fecha refleje el año completo ' MONTH ' Hace que la fecha refleje el mes completo más cercano a la fecha ' HH24 ' Redondea la hora a las 00:00 más cercanas ' DAY ' Redondea al día más cercano
TRUNC (fecha [formato])	Igual que el anterior pero trunca la fecha en lugar de redondearla.

4.6.4 Funciones de conversión

Oracle es capaz de convertir datos automáticamente a fin de que la expresión final tenga sentido. En ese sentido son fáciles las conversiones de texto a número y viceversa.

Ejemplos:

```
-- El resultado es 8
SELECT 5+'3'
FROM DUAL;

-- El resultado es 53
SELECT 5||'3'
FROM DUAL;
```

Pero en determinadas ocasiones queremos realizar conversiones explícitas. Para hacerlo utilizaremos las funciones que se detallan a continuación.

Función de conversión TO_CHAR

Obtiene un texto a partir de un número o una fecha. En especial se utiliza con fechas (ya que de número a texto se suele utilizar de forma implícita). En el caso de las fechas se indica el formato de conversión, que es una cadena que puede incluir estos símbolos (en una cadena de texto):

Símbolo	Significado
YY	Año en formato de dos cifras
YYYY	Año en formato de cuatro cifras
MM	Mes en formato de dos cifras
MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes
DY	Día de la semana en tres letras
DAY	Día completo de la semana
D	Día de la semana (del 1 al 7)
DD	Día del mes en formato de dos cifras (del 1 al 31)
DDD	Día del año
Q	Semestre
WW	Semana del año
AM	Indicador AM
PM	Indicador PM
HH12	Hora de 1 a 12
HH24	Hora de 0 a 23
MI	Minutos (0 a 59)
SS	Segundos (0 a 59)
SSSS	Segundos desde medianoche
/ . , ; ' ,	Posición de los separadores, donde se pongan estos símbolos aparecerán en el resultado

Ejemplo:

```
-- Si esta consulta se ejecuta el 20 de Febrero de 2014 a las 14:15 horas,
-- devuelve: 20/FEBRERO/2014, JUEVES 14:15:03
SELECT TO_CHAR(SYSDATE, 'DD/MONTH/YYYY, DAY HH:MI:SS')
FROM DUAL;
```

Para convertir números a textos se usa esta función cuando se desean características especiales. En este caso en el formato se pueden utilizar estos símbolos:

Función de conversión TO_NUMBER

Convierte textos en números. Se indica el formato de la conversión.

Símbolo	Significado
9	Posición del número
0	Posición del número (muestra ceros)
\$	Formato dólar
L	Símbolo local de la moneda
S	Hace que aparezca el símbolo del signo
D	Posición del símbolo decimal (en español, la coma)
G	Posición del separador de grupo (en español el punto)

Función de conversión TO_DATE

Convierte textos en fechas. Como segundo parámetro se utilizan los códigos de formato de fechas comentados anteriormente.

4.6.5 Función DECODE

Se evalúa una expresión y se colocan a continuación pares valor,resultado de forma que si se la expresión equivale al valor, se obtiene el resultado indicado. Se puede indicar un último parámetro con el resultado a efectuar en caso de no encontrar ninguno de los valores indicados. Sintaxis:

```
DECODE (
    expresión, valor1, resultado1
            [, valor2, resultado2] ...
            [, valorPorDefecto]
);
```

Ejemplo:

```
SELECT
    DECODE (cotización, 1, salario*0.85,
                2, salario*0.93,
                3, salario*0.96,
                salario)
FROM EMPLEADOS;
```

Este ejemplo es idéntico al mostrado con una expresión CASE.

4.6.6 Expresión CASE

Es una instrucción incorporada a la versión 9 de Oracle que permite establecer condiciones de salida (al estilo if-then-else de muchos lenguajes).

```
CASE expresión
    WHEN valor1 THEN resultado1
    [WHEN valor2 THEN resultado2] ...
```

(continué en la próxima página)

(proviene de la página anterior)

```
[ELSE resultado_por_defecto]
END;
```

El funcionamiento es el siguiente:

1. Se evalúa la expresión indicada.
2. Se comprueba si esa expresión es igual al valor del primer WHEN, de ser así se devuelve el primer resultado (cualquier valor excepto nulo).
3. Si la expresión no es igual al valor 1, entonces se comprueba si es igual al segundo. De ser así se escribe el resultado 2. De no ser así se continua con el siguiente WHEN.
4. El resultado indicado en la zona ELSE sólo se escribe si la expresión no vale ningún valor de los indicados.

Ejemplo:

```
SELECT
CASE cotización
  WHEN 1 THEN salario*0.85
  WHEN 2 THEN salario*0.93
  WHEN 3 THEN salario*0.96
  ELSE salario
END
FROM EMPLEADOS;
```

4.7 AGRUPACIONES

Es muy común utilizar consultas en las que se desee agrupar los datos a fin de realizar cálculos en vertical, es decir calculados a partir de datos de distintos registros. Para ello se utiliza la cláusula GROUP BY que permite indicar en base a qué registros se realiza la agrupación. Con GROUP BY la instrucción SELECT queda de esta forma:

```
SELECT lista_expresiones
FROM lista_tablas
[ WHERE condiciones ]
[ GROUP BY grupos ]
[ HAVING condiciones_de_grupos ]
[ ORDER BY columnas ];
```

En el apartado GROUP BY, se indican las columnas por las que se agrupa. La función de este apartado es crear un único registro por cada valor distinto en las columnas del grupo. Vamos a ver un ejemplo de como funciona GROUP BY. Supongamos que tenemos la siguiente tabla EXISTENCIAS:

TI	MODELO	N_ALMACEN	CANTIDAD
AR	6	1	2500
AR	6	2	5600
AR	6	3	2430
AR	9	1	250
AR	9	2	4000
AR	9	3	678
AR	15	1	5667
AR	20	3	43
BI	10	2	340
BI	10	3	23
BI	38	1	1100
BI	38	2	540
BI	38	3	

Si por ejemplo agrupamos en base a las columnas tipo y modelo, la sintaxis sería la siguiente:

```
SELECT tipo, modelo
FROM EXISTENCIAS
GROUP BY tipo, modelo;
```

Al ejecutarla, en la tabla de existencias, se creará un único registro por cada tipo y modelo distintos, generando la siguiente salida:

TI	MODELO
AR	6
AR	9
AR	15
AR	20
BI	10
BI	38

Es decir es un resumen de los datos anteriores. Pero observamos que los datos n_almacen y cantidad no están disponibles directamente ya que son distintos en los registros del mismo grupo.

Así si los hubiésemos seleccionado también en la consulta habríamos ejecutado una consulta ERRÓNEA. Es decir al ejecutar:

```
SELECT tipo, modelo, cantidad
FROM EXISTENCIAS
GROUP BY tipo, modelo;
```

Habríamos obtenido un mensaje de error.

```
ERROR en línea 1:
ORA-00979: no es una expresión GROUP BY
```

Es decir esta consulta es errónea, porque GROUP BY sólo se pueden utilizar desde funciones.

4.7.1 Funciones de cálculo con grupo (o funciones colectivas)

Lo interesante de la creación de grupos es la posibilidad de cálculo que ofrece. Para ello se utilizan las funciones que permiten trabajar con los registros de un grupo. Estas son:

Función	Significado
COUNT (*)	Cuenta los elementos de un grupo. Se utiliza el asterisco para no tener que indicar un nombre de columna concreto, el resultado es el mismo para cualquier columna
SUM (expresión)	Suma los valores de la expresión
AVG (expresión)	Calcula la media aritmética sobre la expresión indicada
MIN (expresión)	Mínimo valor que toma la expresión indicada
MAX (expresión)	Máximo valor que toma la expresión indicada
STDDEV (expresión)	Calcula la desviación estándar
VARIANCE (expresión)	Calcula la varianza

Las funciones anteriores se aplicarán sobre todos los elementos del grupo. Así, por ejemplo, podemos calcular la suma de las cantidades para cada tipo y modelo de la tabla EXISTENCIAS. (Es como si lo hiciéramos manualmente con las antiguas fichas sobre papel: primero las separaríamos en grupos poniendo juntas las que tienen el mismo tipo y modelo y luego para cada grupo sumaríamos las cantidades). La sintaxis SQL de dicha consulta quedaría:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
GROUP BY tipo, modelo;
```

Y se obtiene el siguiente resultado, en el que se suman las cantidades para cada grupo.

TI	MODELO	SUM(CANTIDAD)
AR	6	10530
AR	9	4928
AR	15	5667
AR	20	43
BI	10	363
BI	38	1740

4.7.2 Condiciones HAVING

A veces se desea restringir el resultado de una función agrupada y no aplicarla a todos los grupos. Por ejemplo, imaginemos que queremos realizar la consulta anterior, es decir queremos calcular la suma de las cantidades para cada tipo y modelo de la tabla EXISTENCIAS, pero queremos que se muestren solo los registros en los que la suma de las cantidades calculadas sean mayor que 500. si planteáramos la consulta del modo siguiente:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
WHERE SUM(cantidad) > 500
GROUP BY tipo, modelo;
```

Habríamos ejecutado una consulta ERRÓNEA.

```
ERROR en línea 3:
ORA-00934: función de grupo no permitida aquí
```

La razón es que Oracle calcula primero el WHERE y luego los grupos; por lo que esa condición no la puede realizar al no estar establecidos los grupos. Es decir, no puede saber que grupos tienen una suma de cantidades mayor que 500 cuando todavía no ha aplicado los grupos. Por ello se utiliza la cláusula HAVING, cuya ejecución se efectúa una vez realizados los grupos. Se usaría de esta forma:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
GROUP BY tipo, modelo
HAVING SUM(cantidad) > 500;
```

Ahora bien, esto no implica que con la cláusula GROUP BY no podamos emplear un WHERE. Esta expresión puede usarse para imponer condiciones sobre las filas de la tabla antes de agrupar. Por ejemplo, la siguiente expresión es correcta:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
WHERE tipo='AR'
GROUP BY tipo, modelo
HAVING SUM(cantidad) > 500;
```

De la tabla EXISTENCIAS tomará solo aquellas filas cuyo tipo sea AR, luego agrupará según tipo y modelo y dejará sólo aquellos grupos en los que SUM(cantidad)>500 y por último mostrará tipo, modelo y la suma de las cantidades para aquellos grupos que cumplan dicha condición. En definitiva, el orden de ejecución de la consulta marca lo que se puede utilizar con WHERE y lo que se puede utilizar con HAVING.

Pasos en la ejecución de una consulta SELECT el gestor de bases de datos sigue el siguiente orden:

1. Se aplica la cláusula FROM, de manera que determina sobre que tablas se va a ejecutar la consulta.
2. Se seleccionan las filas deseadas utilizando WHERE. (Solo quedan las filas que cumplen las condiciones especificadas en el WHERE).
3. Se establecen los grupos indicados en la cláusula GROUP BY.
4. Se calculan los valores de las funciones de totales o colectivas que se especifiquen en el HAVING (COUNT, SUM, AVG,...)
5. Se filtran los registros que cumplen la cláusula HAVING
6. Se aplica la cláusula SELECT que indica las columnas que mostraremos en la consulta.
7. El resultado se ordena en base al apartado ORDER BY.

4.8 OBTENER DATOS DE MÚLTIPLES TABLAS

Es más que habitual necesitar en una consulta datos que se encuentran distribuidos en varias tablas. Las bases de datos relacionales se basan en que los datos se distribuyen en tablas que se pueden relacionar mediante un campo. Ese campo es el que permite integrar los datos de las tablas. A continuación veremos como se pueden realizar las

consultas entre varias tablas. Para ello partiremos del siguiente ejemplo: Supongamos que disponemos de una tabla de EMPLEADOS cuya clave principal es el DNI y otra tabla de TAREAS que se refiere a las tareas realizadas por los empleados. Suponemos que cada empleado realizará múltiples tareas, pero que cada tarea es realizada por un único empleado. Si el diseño está bien hecho, en la tabla de TAREAS aparecerá el DNI del empleado (como clave foránea) para saber qué empleado realizó la tarea.

4.8.1 Producto cruzado o cartesiano de tablas

En el ejemplo anterior si quiere obtener una lista de los datos de las tareas y los empleados, se podría hacer de esta forma:

```
SELECT cod_tarea, descripción_tarea, dni_empleado, nombre_empleado
FROM TAREAS, EMPLEADOS;
```

Aunque la sintaxis es correcta ya que, efectivamente, en el apartado FROM se pueden indicar varias tareas separadas por comas, al ejecutarla produce un producto cruzado de las tablas. Es decir, aparecerán todos los registros de las tareas relacionados con todos los registros de empleados (y no para cada empleado sus tareas específicas). El producto cartesiano pocas veces es útil para realizar consultas. Nosotros necesitamos discriminar ese producto para que sólo aparezcan los registros de las tareas relacionadas con sus empleados correspondientes. A eso se le llama asociar tablas (join) y se ve en el siguiente apartado.

4.8.2 Asociando tablas

La forma de realizar correctamente la consulta anterior (asociado las tareas con los empleados que la realizaron) sería:

```
SELECT cod_tarea, descripción_tarea, dni_empleado, nombre_empleado
FROM TAREAS, EMPLEADOS
WHERE TAREAS.dni_empleado = EMPLEADOS.dni_empleado;
```

Nótese que se utiliza la notación tabla.columna para evitar la ambigüedad, ya que el mismo nombre de campo se puede repetir en ambas tablas. Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un alias de tabla:

```
SELECT T.cod_tarea, T.descripcion_tarea, E.dni_empleado, E.nombre_empleado
FROM TAREAS T, EMPLEADOS E
WHERE T.dni_empleado = E.dni_empleado;
```

A la sintaxis WHERE se le pueden añadir condiciones sin más que encadenarlas con el operador AND.

Ejemplo:

```
SELECT T.cod_tarea, T.descripcion_tarea
FROM TAREAS T, EMPLEADOS E
WHERE T.dni_empleado = E.dni_empleado AND E.nombre_empleado = 'Javier';
```

Finalmente indicar que se pueden enlazar más de dos tablas a través de sus claves principales y foráneas. Por cada relación necesaria entre tablas, aparecerá una condición (igualando la clave principal y la foránea correspondiente) en el WHERE.

Ejemplo:

```
SELECT T.cod_tarea, T.descripcion_tarea, E.nombre_empleado, U.nombre_utensilio
FROM TAREAS T, EMPLEADOS E, UTENSILIOS U
WHERE T.dni_empleado = E.dni_empleado AND T.cod_tarea = U.cod_tarea;
```

4.8.3 Relaciones sin igualdad

A las relaciones descritas anteriormente se las llama relaciones en igualdad (equijoins), ya que las tablas se relacionan a través de campos que contienen valores iguales en dos tablas. A veces esto no ocurre, en las tablas:

EMPLEADOS		
Empleado	Sueldo	
Antonio	18000	
Marta	21000	
Sonia	15000	

CATEGORÍAS		
categoría	Sueldo mínimo	Sueldo máximo
D	6000	11999
C	12000	17999
B	18000	20999
A	20999	80000

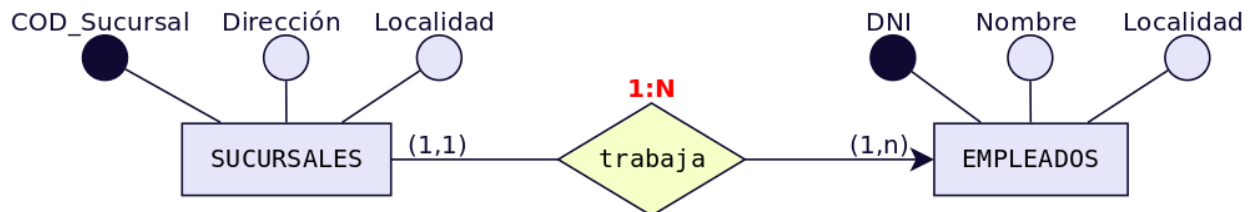
En el ejemplo anterior podríamos averiguar la categoría a la que pertenece cada empleado, pero estas tablas poseen una relación que ya no es de igualdad.

La forma sería:

```
SELECT E.empleado, E.sueldo, C.categoria
FROM EMPLEADOS E, CATEGORÍAS C
WHERE E.sueldo BETWEEN C.sueldo_mínimo AND C.sueldo_máximo;
```

4.8.4 Combinación de tablas (JOIN)

Existe otra forma más moderna e intuitiva de trabajar con varias tablas. Para ello se utiliza la cláusula JOIN. Supongamos que tenemos una base de datos de una entidad bancaria. Disponemos de una tabla con sus empleados y otra tabla con sus sucursales. En una sucursal trabajan varios empleados. Los empleados viven en una localidad y trabajan en una sucursal situada en la misma localidad o en otra localidad. El esquema E-R es el siguiente:



Los datos de las tablas son:

Tabla 1: EMPLEADOS

DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL
11111111A	ANA	ALMERÍA	0001
22222222B	BERNARDO	GRANADA	0001
33333333C	CARLOS	GRANADA	■
44444444D	DAVID	JEREZ	0003

Tabla 2: SUCURSALES

COD_SUCURSAL	DIRECCIÓN	LOCALIDAD
0001	C/ ANCHA, 1	ALMERÍA
0002	C/ NUEVA, 1	GRANADA
0003	C/ CORTÉS, 33	CÁDIZ

Se observa que Ana vive en Almería y trabaja en la sucursal 0001 situada en Almería. Bernardo vive en Granada pero trabaja en la sucursal 0001 de Almería. Carlos es un empleado del que no disponemos el dato acerca de la sucursal en la que trabaja. David es un empleado que vive en Jerez de la Frontera y trabaja en la sucursal 0003 en Cádiz. Existe otra sucursal 0002 en Granada donde no aparece registrado ningún empleado.

Existen diversas formas de combinar (JOIN) las tablas según la información que deseemos obtener. Los tipos de JOIN se clasifican en:

- INNER JOIN (o simplemente JOIN): Combinación interna.
 - JOIN
 - SELF JOIN
 - NATURAL JOIN
- OUTER JOIN: Combinación externa.
 - LEFT OUTER JOIN (o simplemente LEFT JOIN)
 - RIGHT OUTER JOIN (o simplemente RIGHT JOIN)
 - FULL OUTER JOIN (o simplemente FULL JOIN)
- CROSS JOIN: Combinación cruzada.

Pasamos a continuación a explicar cada uno de ellos.

INNER JOIN

También se conoce como EQUI JOIN o combinación de igualdad. Esta combinación devuelve todas las filas de ambas tablas donde hay una coincidencia. Este tipo de unión se puede utilizar en la situación en la que sólo debemos seleccionar las filas que tienen valores comunes en las columnas que se especifican en la cláusula ON.

JOIN

En lugar de INNER JOIN es más frecuente encontrarlo escrito como JOIN simplemente:

Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
       TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 JOIN TABLA2
ON TABLA1.columnaX = TABLA2.columnaY;
```

Por ejemplo, para ver los empleados con sucursal asignada:

```
SELECT E.*, S.LOCALIDAD
FROM EMPLEADOS E JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL	LOCALIDAD
22222222B	BERNARDO	GRANADA	0001	ALMERÍA
11111111A	ANA	ALMERÍA	0001	ALMERÍA
44444444D	DAVID	JEREZ	0003	CÁDIZ

En esta consulta, utilizamos la combinación interna basada en la columna «COD_SUCURSAL» que es común en las tablas «EMPLEADOS» y «SUCURSALES». Esta consulta dará todas las filas de ambas tablas que tienen valores comunes en la columna «COD_SUCURSAL»

SELF JOIN

En algún momento podemos necesitar unir una tabla consigo misma. Este tipo de combinación se denomina SELF JOIN. En este JOIN, necesitamos abrir dos copias de una misma tabla en la memoria. Dado que el nombre de tabla es el mismo para ambas instancias, usamos los alias de tabla para hacer copias idénticas de la misma tabla que se abran en diferentes ubicaciones de memoria.

Nota: Observa que no existe la clausula SELF JOIN, solo JOIN.

Sintaxis:

```
SELECT ALIAS1.columna1, ALIAS1.columna2, ..., ALIAS2.columna1, ...
FROM TABLA ALIAS1 JOIN TABLA ALIAS2
ON ALIAS1.columnaX = ALIAS2.columnaY;
```

Ejemplo:

```
SELECT E1.NOMBRE, E2.NOMBRE, E1.LOCALIDAD
FROM EMPLEADOS E1 JOIN EMPLEADOS E2 ON E1.LOCALIDAD = E2.LOCALIDAD;
```

NOMBRE	NOMBRE	LOCALIDAD
ANA	ANA	ALMERÍA
CARLOS	BERNARDO	GRANADA
BERNARDO	BERNARDO	GRANADA
CARLOS	CARLOS	GRANADA
BERNARDO	CARLOS	GRANADA
DAVID	DAVID	JEREZ

Esto muestra las combinaciones de los empleados que viven en la misma localidad. En este caso no es de mucha utilidad, pero el SELF JOIN puede ser muy útil en relaciones reflexivas.

NATURAL JOIN

NATURAL JOIN establece una relación de igualdad entre las tablas a través de los campos que tengan el mismo nombre en ambas tablas. Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
       TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 NATURAL JOIN TABLA2;
```

En este caso no existe clausula ON puesto que se realiza la combinación teniendo en cuenta las columnas del mismo nombre. Por ejemplo:

```
SELECT *
FROM EMPLEADOS E NATURAL JOIN SUCURSALES S;
```

LOCALIDAD	COD_SUCURSAL	DNI	NOMBRE	DIRECCIÓN
ALMERÍA	0001	11111111A	ANA	C/ ANCHA, 1

En el resultado de la consulta nos aparece la combinación donde la (LOCALIDAD, COD_SUCURSAL) de EMPLEADOS es igual a (LOCALIDAD, COD_SUCURSAL) de SUCURSALES. Es decir estamos mostrando todos los empleados que tienen asignada una sucursal y dicha sucursal está en la localidad donde vive el empleado. El NATURAL JOIN elimina columnas duplicadas, por eso no aparecen los campos LOCALIDAD ni SUCURSAL duplicados. Este tipo de consulta no permite indicar estos campos en la cláusula SELECT. Por ejemplo: SELECT E.LOCALIDAD o SELECT E.COD_SUCURSAL sería incorrecto.

OUTER JOIN

La combinación externa o OUTER JOIN es muy útil cuando deseamos averiguar que campos están a NULL en un lado de la combinación. En nuestro ejemplo, podemos ver qué empleados no tienen sucursal asignada; también podemos ver que sucursales no tienen empleados asignados.

LEFT JOIN

También conocido como LEFT OUTER JOIN, nos permite obtener todas las filas de la primera tabla asociadas a filas de la segunda tabla. Si no existe correspondencia en la segunda tabla, dichos valores aparecen como NULL. Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
       TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 LEFT JOIN TABLA2
ON TABLA1.columnaX = TABLA2.columnaY;
```

Por ejemplo:

```
SELECT E.*, S.LOCALIDAD
FROM EMPLEADOS E LEFT JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL	LOCALIDAD
11111111A	ANA	ALMERÍA	0001	ALMERÍA
22222222B	BERNARDO	GRANADA	0001	ALMERÍA
33333333C	CARLOS	GRANADA	■	■
44444444D	DAVID	JEREZ	0003	CÁDIZ

Todos los empleados tienen una sucursal asignada salvo el empleado Carlos.

RIGHT JOIN

También conocido como RIGHT OUTER JOIN, nos permite obtener todas las filas de la segunda tabla asociadas a filas de la primera tabla. Si no existe correspondencia en la primera tabla, dichos valores aparecen como NULL. Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
       TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 RIGHT JOIN TABLA2
ON TABLA1.columnaX = TABLA2.columnaY;
```

Por ejemplo:

```
SELECT E.DNI, E.NOMBRE, S.*
FROM EMPLEADOS E RIGHT JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

DNI	NOMBRE	COD_SUCURSAL	DIRECCIÓN	LOCALIDAD
11111111A	ANA	0001	C/ ANCHA, 1	ALMERÍA
22222222B	BERNARDO	0001	C/ ANCHA, 1	ALMERÍA
44444444D	DAVID	0003	C/ CORTÉS, 33	CÁDIZ
■	■	0002	C/ NUEVA, 1	GRANADA

Todas las sucursales tienen algún empleado asignado salvo la sucursal 0002.

FULL JOIN

También conocido como FULL OUTER JOIN, nos permite obtener todas las filas de la primera tabla asociadas a filas de la segunda tabla. Si no existe correspondencia en alguna de las tablas, dichos valores aparecen como NULL.

Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
       TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 FULL JOIN TABLA2
ON TABLA1.columnaX = TABLA2.columnaY;
```

Por ejemplo:

```
SELECT E.DNI, E.NOMBRE, S.COD_SUCURSAL, S.LOCALIDAD
FROM EMPLEADOS E FULL JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

DNI	NOMBRE	COD_SUCURSAL	LOCALIDAD	
22222222B	BERNARDO	0001	ALMERÍA	
11111111A	ANA	0001	ALMERÍA	
■	■	0002	GRANADA	
44444444D	DAVID	0003	CÁDIZ	
33333333C	CARLOS	■	■	

Como puede observarse fácilmente, vemos que en la sucursal 0002 no hay ningún empleado asignado y que el empleado Carlos no tiene asignada ninguna sucursal.

CROSS JOIN

El CROSS JOIN o combinación cruzada produce el mismo resultado del producto cartesiano, es decir nos da todas las combinaciones posibles. Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
       TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 CROSS JOIN TABLA2;
```

Por ejemplo:

```
SELECT E.DNI, E.NOMBRE, E.LOCALIDAD, S.COD_SUCURSAL, S.LOCALIDAD
FROM EMPLEADOS E CROSS JOIN SUCURSALES S;
```

DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL	LOCALIDAD
11111111A	ANA	ALMERÍA	0001	ALMERÍA
11111111A	ANA	ALMERÍA	0002	GRANADA
11111111A	ANA	ALMERÍA	0003	CÁDIZ
22222222B	BERNARDO	GRANADA	0001	ALMERÍA
22222222B	BERNARDO	GRANADA	0002	GRANADA
22222222B	BERNARDO	GRANADA	0003	CÁDIZ
33333333C	CARLOS	GRANADA	0001	ALMERÍA
33333333C	CARLOS	GRANADA	0002	GRANADA
33333333C	CARLOS	GRANADA	0003	CÁDIZ
44444444D	DAVID	JEREZ	0001	ALMERÍA
44444444D	DAVID	JEREZ	0002	GRANADA
44444444D	DAVID	JEREZ	0003	CÁDIZ

En el ejemplo que estamos viendo nos mostraría 12 filas (4x3: 4 filas de empleados x 3 filas de sucursales). El primer cliente se combina con todas las sucursales. El segundo cliente igual. Y así sucesivamente. Esta combinación asocia todas las filas de la tabla izquierda con cada fila de la tabla derecha. Este tipo de unión es necesario cuando necesitamos seleccionar todas las posibles combinaciones de filas y columnas de ambas tablas. Este tipo de unión no es generalmente preferido ya que toma mucho tiempo y da un resultado enorme que no es a menudo útil

4.9 COMBINACIONES ESPECIALES

4.9.1 Uniones

La palabra UNION permite añadir el resultado de un SELECT a otro SELECT. Para ello ambas instrucciones tienen que utilizar el mismo número y tipo de columnas.

Ejemplo:

```
-- Tipos y modelos de piezas que se encuentren el almacén 1, en el 2 o en ambos.

SELECT tipo,modelo FROM existencias
WHERE n_almacen = 1

UNION

SELECT tipo,modelo FROM existencias
WHERE n_almacen = 2;
```

Nota: Observa que sólo se pone punto y coma al final.

Es decir, UNION crea una sola tabla con registros que estén presentes en cualquiera de las consultas. Si están repetidas sólo aparecen una vez, para mostrar los duplicados se utiliza UNION ALL en lugar de la palabra UNION.

4.9.2 Intersecciones

De la misma forma, la palabra INTERSECT permite unir dos consultas SELECT de modo que el resultado serán las filas que estén presentes en ambas consultas. Ejemplo: tipos y modelos de piezas que se encuentren en los almacenes 1 y 2 (en ambos).

```
-- Tipos y modelos de piezas que se encuentren en los almacenes 1 y 2 (en ambos).

SELECT tipo,modelo FROM existencias
WHERE n_almacen = 1

INTERSECT

SELECT tipo,modelo FROM existencias
WHERE n_almacen = 2;
```

Nota: Observa que sólo se pone punto y coma al final.

4.9.3 Diferencia

Con MINUS también se combinan dos consultas SELECT de forma que aparecerán los registros del primer SELECT que no estén presentes en el segundo. Ejemplo:

```
-- Tipos y modelos de piezas que se encuentren el almacén 1 y no en el 2.

SELECT tipo,modelo FROM existencias
```

(continué en la próxima página)

(proviene de la página anterior)

```
WHERE n_almacen = 1

MINUS

SELECT tipo,modelo FROM existencias
WHERE n_almacen = 2;
```

Nota: Observa que sólo se pone punto y coma al final.

Nota: La palabra *MINUS* es utiliza en el SGBD Oracle. El estándar SQL establece la palabra *EXCEPT*.

Se podrían hacer varias combinaciones anidadas (una unión a cuyo resultado se restará de otro `SELECT` por ejemplo), en ese caso es conveniente utilizar paréntesis para indicar qué combinación se hace primero:

```
(
SELECT ...

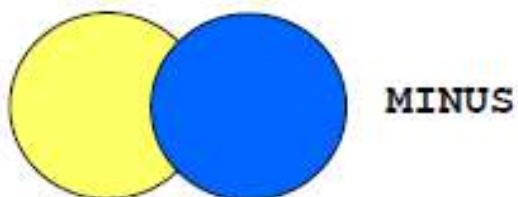
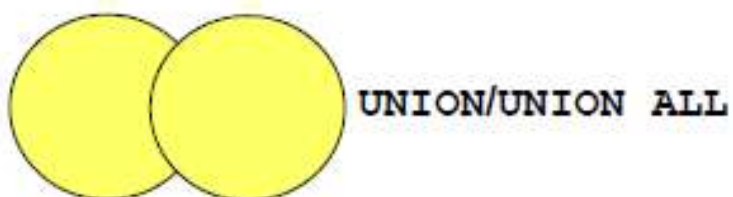
UNION

SELECT ...
)

MINUS

SELECT ... ;

-- Primero se hace la unión y luego la diferencia
```



4.10 CONSULTA DE VISTAS

A efectos de uso, la consulta de una vista es idéntica a la consulta de una tabla. Supongamos que tenemos la siguiente vista:

```
CREATE VIEW RESUMEN
-- a continuación indicamos los alias
(id_localidad, localidad, poblacion,
 n_provincia, provincia, superficie,
 id_comunidad, comunidad)
AS
SELECT L.IdLocalidad, L.Nombre, L.Poblacion,
       P.IdProvincia, P.Nombre, P.Superficie,
       C.IdComunidad, C.Nombre
FROM LOCALIDADES L JOIN PROVINCIAS P ON L.IdProvincia = P.IdProvincia
                  JOIN COMUNIDADES C ON P.IdComunidad = C.IdComunidad;
```

Podemos consultar ahora sobre la vista como si de una tabla se tratase

```
SELECT DISTINCT (provincia, comunidad) FROM resumen;
```

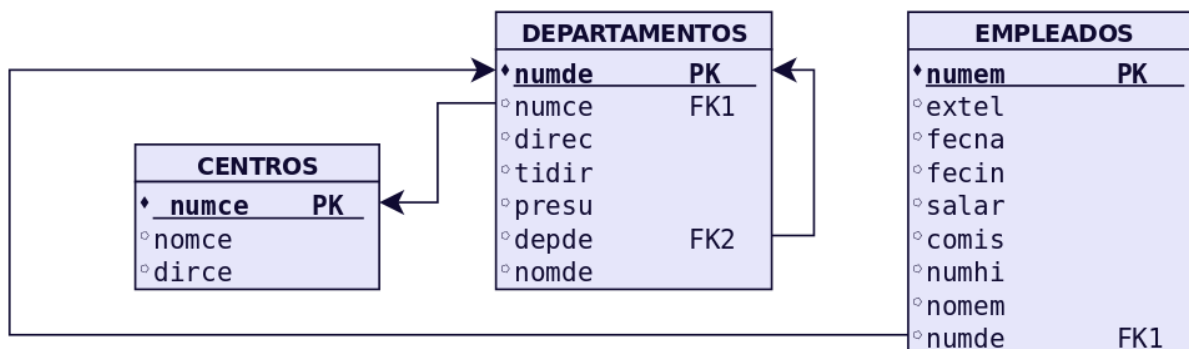
4.11 ACTIVIDADES PROPUESTAS

IMPORTANTE

Todos los scripts que aparecen a continuación deben ejecutarse en SQL*Plus.

4.11.1 Prácticas

Dado el siguiente modelo relacional:



Práctica 1: Creación de BD e inserción de Datos.

1. Obtener el posible diagrama E/R a partir del modelo relacional anterior.
2. Escribir las sentencias SQL correspondientes para crear las tablas en ORACLE, teniendo en cuenta las siguientes restricciones:

Tabla 3: CENTROS

Campo	Nulo	Tipo de datos	Observaciones
NUMCE	NOT NULL	NUMBER(4)	Número de centro
NOMCE		VARCHAR2(25)	Nombre de centro
DIRCE		VARCHAR2(25)	Dirección del centro

Tabla 4: DEPARTAMENTOS

Campo	Nulo	Tipo de datos	Observaciones
NUMDE	NOT NULL	NUMBER(3)	Número de departamento
NUMCE		NUMBER(4)	Número de centro
DIREC		NUMBER(3)	Director
TIDIR		CHAR(1)	Tipo de director (en Propiedad, en Funciones)
PRESU		NUMBER(3,1)	Presupuesto en miles de €
DEPDE		NUMBER(3)	Departamento del que depende
NOMDE		VARCHAR2(20)	Nombre de departamento

Tabla 5: EMPLEADOS

Campo	Nulo	Tipo de datos	Observaciones
NUMEM	NOT NULL	NUMBER(3)	Número de empleado
EXTEL		NUMBER(3)	Extensión telefónica
FECNA		DATE	Fecha de nacimiento
FECIN		DATE	Fecha de incorporación
SALAR		NUMBER(5)	Salario
COMIS		NUMBER(3)	Comisión
NUMHI		NUMBER(1)	Número de hijos
NOMEM		VARCHAR2(10)	Nombre de empleado
NUMDE		NUMBER(3)	Número de departamento

3. Inserta los siguientes datos en la tabla DEPARTAMENTOS.

Tabla 6: EMPLEADOS

NUMDE	NUMCE	DIREC	TIDIR	PRESU	DEPDE	NOMDE
100	10	260	P	72	NULL	DIRECCIÓN GENERAL
110	20	180	P	90	100	DIRECC.COMERCIAL
111	20	180	F	66	110	SECTOR INDUSTRIAL
112	20	270	P	54	110	SECTOR SERVICIOS
120	10	150	F	18	100	ORGANIZACIÓN
121	10	150	P	12	120	PERSONAL
122	10	350	P	36	120	PROCESO DE DATOS
130	10	310	P	12	100	FINANZAS

4. ¿Qué ocurre al insertar el primer registro? ¿Por qué? Plantea la solución.
5. Inserta los siguientes datos en la tabla CENTROS

Tabla 7: EMPLEADOS

NUMCE	NOMCE	DIRCE
10	SEDE CENTRAL	C/ ATOCHA, 820, MADRID
20	RELACIÓN CON CLIENTES	C/ ATOCHA, 405, MADRID

6. Inserta los siguientes datos en la tabla EMPLEADOS.

Tabla 8: EMPLEADOS

NUMEM	EXTEL	FECNA	FECIN	SALAR	COMIS	NUMHI	NOMEM	NUMDE
110	350	10/11/1970	15/02/1985	1800	NULL	3	CESAR	121
120	840	09/06/1968	01/10/1988	1900	110	1	MARIO	112
130	810	09/09/1965	01/02/1981	1500	110	2	LUCIANO	112
150	340	10/08/1972	15/01/1997	2600	NULL	0	JULIO	121
160	740	09/07/1980	11/11/2005	1800	110	2	AUREO	111
180	508	18/10/1974	18/03/1996	2800	50	2	MARCOS	110
190	350	12/05/1972	11/02/1992	1750	NULL	4	JULIANA	121
210	200	28/09/1970	22/01/1999	1910	NULL	2	PILAR	100
240	760	26/02/1967	24/02/1989	1700	100	3	LAVINIA	111
250	250	27/10/1976	01/03/1997	2700	NULL	0	ADRIANA	100
260	220	03/12/1973	12/07/2001	720	NULL	6	ANTONIO	100
270	800	21/05/1975	10/09/2003	1910	80	3	OCTAVIO	112
280	410	10/01/1978	08/10/2010	1500	NULL	5	DOROTEA	130
285	620	25/10/1979	15/02/2011	1910	NULL	0	OTILIA	122
290	910	30/11/1967	14/02/1988	1790	NULL	3	GLORIA	120
310	480	21/11/1976	15/01/2001	1950	NULL	0	AUGUSTO	130
320	620	25/12/1977	05/02/2003	2400	NULL	2	CORNELIO	122
330	850	19/08/1958	01/03/1980	1700	90	0	AMELIA	112
350	610	13/04/1979	10/09/1999	2700	NULL	1	AURELIO	122
360	750	29/10/1978	10/10/1998	1800	100	2	DORINDA	111
370	360	22/06/1977	20/01/2000	1860	NULL	1	FABIOLA	121
380	880	30/03/1978	01/01/1999	1100	NULL	0	MICAELA	112
390	500	19/02/1976	08/10/2010	1290	NULL	1	CARMEN	110
400	780	18/08/1979	01/11/2011	1150	NULL	0	LUCRECIA	111
410	660	14/07/1968	13/10/1989	1010	NULL	0	AZUCENA	122
420	450	22/10/1966	19/11/1988	2400	NULL	0	CLAUDIA	130
430	650	26/10/1967	19/11/1988	1260	NULL	1	VALERIANA	122
440	760	26/09/1966	28/02/1986	1260	100	0	LIVIA	111
450	880	21/10/1966	28/02/1986	1260	100	0	SABINA	112
480	760	04/04/1965	28/02/1986	1260	100	1	DIANA	111
490	880	06/06/1964	01/01/1988	1090	100	0	HORACIO	112
500	750	08/10/1965	01/01/1987	1200	100	0	HONORIA	111
510	550	04/05/1966	01/11/1986	1200	NULL	1	ROMULO	110
550	780	10/01/1970	21/01/1998	600	120	0	SANCHO	111

Práctica 2: Consultas Sencillas

1. Hallar, por orden alfabético, los nombres de los departamentos cuyo director lo es en funciones y no en propiedad.

NOMDE ----- ORGANIZACIÓN SECTOR INDUSTRIAL

2. Obtener un listín telefónico de los empleados del departamento 121 incluyendo nombre de empleado, número de empleado y extensión telefónica. Por orden alfabético.

NOMEM	NUMEM	EXTEL
CESAR	110	350
FABIOLA	370	360
JULIANA	190	350
JULIO	150	340

3. Obtener por orden creciente una relación de todos los números de extensiones telefónicas de los empleados, junto con el nombre de estos, para aquellos que trabajen en el departamento 110. Mostrar la consulta tal y como aparece en la imagen.

Nombre	Extensión Telefónica
CARMEN	500
MARCOS	508
ROMULO	550

4. Hallar la comisión, nombre y salario de los empleados que tienen tres hijos, clasificados por comisión, y dentro de comisión por orden alfabético.

COMIS	NOMEM	SALAR
80	OCTAVIO	1910
100	LAVINIA	1700
	CESAR	1800
	GLORIA	1790

5. Hallar la comisión, nombre y salario de los empleados que tienen tres hijos, clasificados por comisión, y dentro de comisión por orden alfabético, para aquellos empleados que tienen comisión.

COMIS	NOMEM	SALAR
80	OCTAVIO	1910
100	LAVINIA	1700

6. Obtener salario y nombre de los empleados sin hijos y cuyo salario es mayor que 1200 y menor que 1500 €. Se obtendrán por orden decreciente de salario y por orden alfabético dentro de salario.

SALAR	NOMEM
1260	LIVIA
1260	SABINA

7. Obtener los números de los departamentos donde trabajan empleados cuyo salario sea inferior a 1500 €

NUMDE
100
110
111
112
122

8. Obtener las distintas comisiones que hay en el departamento 110.

COMIS

(continué en la próxima página)

(proviene de la página anterior)

50

Práctica 3: Consultas con Predicados Básicos

1. Obtener una relación por orden alfabético de los departamentos cuyo presupuesto es inferior a 30.000 €. El nombre de los departamentos vendrá precedido de las palabras “DEPARTAMENTO DE “. Nota: El presupuesto de los departamentos viene expresado en miles de €.

NOMBRE

DEPARTAMENTO DE FINANZAS
DEPARTAMENTO DE ORGANIZACIÓN
DEPARTAMENTO DE PERSONAL

2. Muestra el número y el nombre de cada departamento separados por un guión y en un mismo campo llamado “Número-Nombre”, además del tipo de director mostrado como “Tipo de Director”, para aquellos departamentos con presupuesto inferior a 30.000 €.

Número-Nombre T

120-ORGANIZACIÓN F
121-PERSONAL P
130-FINANZAS P

3. Suponiendo que en los próximos dos años el coste de vida va a aumentar un 8 % anual y que se suben los salarios solo un 2 % anual, hallar para los empleados con más de 4 hijos su nombre y su sueldo anual, actual y para cada uno de los próximos dos años, clasificados por orden alfabético. Muestra la consulta tal y como aparece en la captura.

Nombre	Salario 2014	Salario 2015	Salario 2016
ANTONIO	8640	8812,8	8989,056
DOROTEA	18000	18360	18727,2

4. Hallar, por orden alfabético, los nombres de los empleados tales que si se les da una gratificación de 120 € por hijo, el total de esta gratificación supera el 20 % de su salario.

NOMEM

ANTONIO
DOROTEA
GLORIA
JULIANA
LAVINIA

5. Para los empleados del departamento 112 hallar el nombre y el salario total (salario más comisión), por orden de salario total decreciente, y por orden alfabético dentro de salario total.

NOMBRE	SALARIO TOTAL
MICHAELA	
MARIO	2010
OCTAVIO	1990

(continué en la próxima página)

(proviene de la página anterior)

AMELIA	1790
LUCIANO	1610
SABINA	1360
HORACIO	1190

6. Vemos que para Micaela no se muestra nada en Salario Total, esto es debido a que su comisión es Nula (Lo que no significa que sea 0—> significa que no se ha introducido ningún valor). Esto impide hacer el cálculo de la suma. Muestra entonces la misma consulta anterior pero sólo para aquellos empleados cuya comisión no sea nula.

NOMBRE	SALARIO TOTAL
MARIO	2010
OCTAVIO	1990
AMELIA	1790
LUCIANO	1610
SABINA	1360
HORACIO	1190

7. Repite la consulta anterior para mostrarla como sigue:

NOMBRE	SALARIO TOTAL
MARIO	2010 €
OCTAVIO	1990 €
AMELIA	1790 €
LUCIANO	1610 €
SABINA	1360 €
HORACIO	1190 €

8. En una campaña de ayuda familiar se ha decidido dar a los empleados una paga extra de 60 € por hijo, a partir del cuarto inclusive. Obtener por orden alfabético para estos empleados: nombre y salario total que van a cobrar incluyendo esta paga extra. Mostrarlo como en la imagen.

NOMBRE	SALARIO TOTAL
ANTONIO	900 €
DOROTEA	1620 €
JULIANA	1810 €

9. Introducción a SELECT subordinado. Imaginemos la misma consulta anterior, pero en la que se nos pide mostrar los mismos campos pero para aquellos empleados cuyo número de hijos iguale o supere a los de Juliana. Es decir, Juliana tiene 4 hijos pero no lo sabemos. Lo que sabemos es el nombre. En este caso haremos otro SELECT cuyo resultado de la búsqueda sea el número de hijos de Juliana.

NOMBRE	SALARIO TOTAL
ANTONIO	900 €
DOROTEA	1620 €
JULIANA	1810 €

10. Obtener por orden alfabético los nombres de los empleados cuyos sueldos igualan o superan al de CLAUDIA en más del 15 %.

NOMEM

MARCOS

11. Obtener los nombres de los departamentos que no dependen funcionalmente de otro.

NOMDE

DIRECCIÓN GENERAL

Práctica 4: Consultas con Predicados Cuantificados. ALL, SOME o ANY.

1. Obtener por orden alfabético los nombres de los empleados cuyo salario supera al máximo salario de los empleados del departamento 122.

NOMEM

MARCOS

2. La misma consulta pero para el departamento 150. Explica por qué obtenemos la relación de todos los empleados por orden alfabético.

NOMEM

ADRIANA
AMELIA
ANTONIO
AUGUSTO
AURELIO
AUREO
AZUCENA
CARMEN
CESAR
CLAUDIA
CORNELIO

NOMEM

DIANA
DORINDA
DOROTEA
FABIOLA
GLORIA
HONORIA
HORACIO
JULIANA
JULIO
LAVINIA
LIVIA

NOMEM

LUCIANO
LUCRECIA
MARCOS
MARIO

(continué en la próxima página)

(proviene de la página anterior)

MICAELA
OCTAVIO
OTILIA
PILAR
ROMULO
SABINA
SANCHO
NOMEM

VALERIANA

3. Obtener por orden alfabético los nombres de los empleados cuyo salario supera en dos veces y media o más al mínimo salario de los empleados del departamento 122.

NOMEM

ADRIANA
AURELIO
JULIO
MARCOS

4. Obtener los nombres y salarios de los empleados cuyo salario coincide con la comisión multiplicada por 10 de algún otro o la suya propia.

NOMEM	SALAR
-----	-----
MICAELA	1100
ROMULO	1200
HONORIA	1200

5. Obtener por orden alfabético los nombres y salarios de los empleados cuyo salario es superior a la comisión máxima existente multiplicada por 20.

NOMEM	SALAR
-----	-----
ADRIANA	2700
AURELIO	2700
JULIO	2600
MARCOS	2800

6. Obtener por orden alfabético los nombres y salarios de los empleados cuyo salario es inferior a veinte veces la comisión más baja existente.

NOMEM	SALAR
-----	-----
ANTONIO	720
SANCHO	600

Práctica 5: Consultas con Predicados BETWEEN

1. Obtener por orden alfabético los nombres de los empleados cuyo salario está entre 1500 € y 1600 €.-

NOMEM

(continué en la próxima página)

(proviene de la página anterior)

DOROTEA
LUCIANO

2. Obtener por orden alfabético los nombres y salarios de los empleados con comisión, cuyo salario dividido por su número de hijos cumpla una, o ambas, de las dos condiciones siguientes:

- Que sea inferior de 720 €
- Que sea superior a 50 veces su comisión.

NOMEM	SALAR
-----	-----
AMELIA	1700
HONORIA	1200
HORACIO	1090
LAVINIA	1700
LIVIA	1260
OCTAVIO	1910
SABINA	1260
SANCHO	600

Práctica 6: Consultas con Predicados LIKE

1. Obtener por orden alfa el nombre y el salario de aquellos empleados que comienzan por la letra “A” y muestra la consulta como aparece en la captura.

Nombre	Salario
-----	-----
ADRIANA	2700 €
AMELIA	1700 €
ANTONIO	720 €
AUGUSTO	1950 €
AURELIO	2700 €
AUREO	1800 €
AZUCENA	1010 €

2. Obtener por orden alfabético los nombres de los empleados que tengan 8 letras.

NOMEM

CORNELIO
LUCRECIA

3. Obtener por orden alfabético los nombres y el presupuesto de los departamentos que incluyen la palabra “SECTOR”. La consulta la deberás mostrar como la imagen.

Departamento	Presupuesto
-----	-----
DEPARTAMENTO DE SECTOR INDUSTRIAL	66.000 €
DEPARTAMENTO DE SECTOR SERVICIOS	54.000 €

Práctica 7: Consultas con Predicados IN

1. Obtener por orden alfabético los nombres de los empleados cuya extensión telefónica es 250 o 750.

NOMEM

ADRIANA
DORINDA
HONORIA

2. Obtener por orden alfabético los nombres de los empleados que trabajan en el mismo departamento que PILAR o DOROTEA.

NOMEM

ADRIANA
ANTONIO
AUGUSTO
CLAUDIA
DOROTEA
PILAR

3. Obtener por orden alfabético los nombres de los departamentos cuyo director es el mismo que el del departamento: DIRECC.COMERCIAL o el del departamento: PERSONAL Mostrar la consulta como imagen.

Nombres Departamentos	Identificador de su director
SECTOR INDUSTRIAL	180
DIRECC.COMERCIAL	180
PERSONAL	150
ORGANIZACIÓN	150

Práctica 8: Consultas con Predicados EXISTS

1. Obtener los nombres de los centros de trabajo si hay alguno que esté en la calle ATOCHA.

NOMCE

SEDE CENTRAL
RELACIÓN CON CLIENTES

2. Obtener los nombres y el salario de los empleados del departamento 100 si en él hay alguno que gane más de 1300 €.

NOMEM	SALAR
PILAR	1910
ADRIANA	2700
ANTONIO	720

3. Obtener los nombres y el salario de los empleados del departamento 100 si en él hay alguno que gane más de 2750 €.

-- no rows selected

4. Obtener los nombres y el salario de los empleados del departamento 100 si en él hay alguno que gane más de 3000 €.

-- no rows selected

Práctica 9: Más Consultas con Predicados

1. Obtener por orden alfabético los nombres y comisiones de los empleados del departamento 110 si en él hay algún empleado que tenga comisión.

NOMEM	COMIS
CARMEN	
MARCOS	50
ROMULO	

2. Obtener los nombres de los departamentos que no sean ni de DIRECCION ni de SECTORES.

NOMDE
ORGANIZACIÓN
PERSONAL
PROCESO DE DATOS
FINANZAS

3. Obtener por orden alfabético los nombres y salarios de los empleados que o bien no tienen hijos y ganan más de 1.500 €, o bien tienen hijos y ganan menos de 1.000 €.

NOMBRE	SALARIO
ADRIANA	2700 €
AMELIA	1700 €
ANTONIO	720 €
AUGUSTO	1950 €
CLAUDIA	2400 €
JULIO	2600 €
OTILIA	1910 €

4. Hallar por orden de número de empleado el nombre y salario total (salario más comisión) de los empleados cuyo salario total supera al salario mínimo en 1800 € mensuales. Muestra la consulta como aparece en la captura de pantalla.

NÚMERO EMPLEADO	NOMBRE	SALARIO TOTAL
nº 180	MARCOS	2850 €

5. Obtener, por orden alfabético, los nombres y salarios de los empleados del departamento 111 que tienen comisión si hay alguno de ellos cuya comisión supere al 15 % de su salario.

NOMEM	SALAR
AUREO	1800
DIANA	1260
DORINDA	1800
HONORIA	1200
LAVINIA	1700
LIVIA	1260
SANCHO	600

6. Hallar los nombres de departamentos, el tipo de director y su presupuesto, para aquellos departamentos que tienen directores en funciones, o bien en propiedad y su presupuesto anual excede a 30.000 € o no dependen de ningún otro.

Nombre de Departamento	T	Presupuesto
DIRECCIÓN GENERAL	P	72
DIRECC.COMERCIAL	P	90
SECTOR INDUSTRIAL	F	66
SECTOR SERVICIOS	P	54
ORGANIZACIÓN	F	18
PROCESO DE DATOS	P	36

7. Realizamos la misma consulta anterior pero mostrándola del modo siguiente:

Nombre de Departamento	T	Presupuesto
DIRECCIÓN GENERAL	P	72.000 €
DIRECC.COMERCIAL	P	90.000 €
SECTOR INDUSTRIAL	F	66.000 €
SECTOR SERVICIOS	P	54.000 €
ORGANIZACIÓN	F	18.000 €
PROCESO DE DATOS	P	36.000 €

Práctica 10: Consultas con Fechas

Nota: En muchos casos, el resultado dependerá de la fecha en la que realizamos la consulta.

1. Obtener por orden alfabético, los nombres y fechas de nacimiento de los empleados que cumplen años en el mes de noviembre.

NOMEM	NACIMIENTO
AUGUSTO	21/11/1976
CESAR	10/11/1970
GLORIA	30/11/1967

2. Obtener los nombres de los empleados que cumplen años en el día de hoy.

-- no rows selected

3. Obtener los nombres y fecha exacta de nacimiento de los empleados cuya fecha de nacimiento es anterior al año 1950.

-- no rows selected

4. Obtener los nombres y fecha exacta de incorporación de los empleados cuya fecha de incorporación a la empresa es anterior al año 1970.

-- no rows selected

5. Obtener los nombres, fecha de nacimiento y fecha de incorporación de los empleados cuya edad a la fecha de incorporación era inferior a 30 años.

NOMEM	FECNA	FECIN
CESAR	10/11/70	15/02/85

(continúe en la próxima página)

(proviene de la página anterior)

MARIO	09/06/68	01/10/88
LUCIANO	09/09/65	01/02/81
JULIO	10/08/72	15/01/97
AUREO	09/07/80	11/11/05
MARCOS	18/10/74	18/03/96
JULIANA	12/05/72	11/02/92
PILAR	28/09/70	22/01/99
LAVINIA	26/02/67	24/02/89
ADRIANA	27/10/76	01/03/97
ANTONIO	03/12/73	12/07/01
NOMEM	FECNA	FECIN
OCTAVIO	21/05/75	10/09/03
GLORIA	30/11/67	14/02/88
AUGUSTO	21/11/76	15/01/01
CORNELIO	25/12/77	05/02/03
AMELIA	19/08/58	01/03/80
AURELIO	13/04/79	10/09/99
DORINDA	29/10/78	10/10/98
FABIOLA	22/06/77	20/01/00
MICAELA	30/03/78	01/01/99
AZUCENA	14/07/68	13/10/89
CLAUDIA	22/10/66	19/11/88
NOMEM	FECNA	FECIN
VALERIANA	26/10/67	19/11/88
LIVIA	26/09/66	28/02/86
SABINA	21/10/66	28/02/86
DIANA	04/04/65	28/02/86
HORACIO	06/06/64	01/01/88
HONORIA	08/10/65	01/01/87
ROMULO	04/05/66	01/11/86
SANCHO	10/01/70	21/01/98

6. Obtener los empleados cuyo nacimiento fue en Lunes.

NOMEM	Día de nacimiento
PILAR	lunes
ANTONIO	lunes
LIVIA	lunes

7. Obtener los empleados cuyo día de la semana para el nacimiento y la incorporación fue Viernes.

NOMEM	Viernes para nac. e incorp.
AURELIO	viernes
SABINA	viernes

8. Obtener los empleados cuyo día de la semana para el nacimiento y la incorporación coinciden. Es decir nacieron y se incorporaron un Lunes, o nacieron y se incorporaron un Martes, etc

NOMEM	Mismo día de nac. e incorp.
-------	-----------------------------

(continué en la próxima página)

(proviene de la página anterior)

OCTAVIO	miércoles
AURELIO	viernes
CLAUDIA	sábado
SABINA	viernes

9. Obtener los empleados y su mes de incorporación siempre que esté entre los meses de Enero y Junio (ambos inclusive).

NOMEM	Mes incorporación
CESAR	FEBRERO
LUCIANO	FEBRERO
JULIO	ENERO
MARCOS	MARZO
JULIANA	FEBRERO
PILAR	ENERO
LAVINIA	FEBRERO
ADRIANA	MARZO
OTILIA	FEBRERO
GLORIA	FEBRERO
AUGUSTO	ENERO
NOMEM	Mes incorporación
CORNELIO	FEBRERO
AMELIA	MARZO
FABIOLA	ENERO
MICAELA	ENERO
LIVIA	FEBRERO
SABINA	FEBRERO
DIANA	FEBRERO
HORACIO	ENERO
HONORIA	ENERO
SANCHO	ENERO

10. Obtener los empleados y su mes de incorporación siempre que esté entre los meses de Enero y Junio (ambos inclusive) y el mes de nacimiento coincida en dicho mes.

NOMEM	Mes incorporación y nac.
LAVINIA	FEBRERO
SANCHO	ENERO

Práctica 11: Consultas con funciones colectivas

1. Hallar el salario medio, mínimo y máximo de los empleados de la empresa.

Salario medio	Salario mínimo	Salario máximo
1670	600	2800

2. Obtener por orden alfabético los salarios y nombres de los empleados tales que su salario más un 40 % supera al máximo salario.

SALAR	NOMEM
2700	ADRIANA
2700	AURELIO
2400	CLAUDIA
2400	CORNELIO
2600	JULIO
2800	MARCOS

3. Hallar la edad en años cumplidos del empleado más viejo del departamento 110. .. note:

La edad que obtengamos dependerá de la fecha en la que realicemos la consulta.

Edad
50

4. Hallar la edad en años cumplidos y el nombre del empleado más viejo del departamento 110.

Nota: La edad que obtengamos dependerá de la fecha en la que realicemos la consulta.

NOMEM	Edad
ROMULO	50

5. Hallar el número de empleados del departamento 112, cuántas comisiones distintas hay en ese departamento y la suma de las comisiones.

COUNT (NUMEM)	COUNT (DISTINCTCOMIS)	SUM (COMIS)
6	4	590

Práctica 12: Agrupamiento de filas. GROUP BY

1. Hallar cuántos empleados hay en cada departamento.

NUMDE	COUNT (NUMEM)
100	3
121	4
120	1
112	7
110	3
130	3
111	8
122	5

2. Hallar para cada departamento el salario medio, el mínimo y el máximo.

NUMDE	Salario medio	Salario mínimo	Salario máximo
100	1776,67	720	2700
121	2002,5	1750	2600

(continué en la próxima página)

(proviene de la página anterior)

120	1790	1790	1790
112	1494,29	1090	1910
110	1763,33	1200	2800
130	1950	1500	2400
111	1346,25	600	1800
122	1856	1010	2700

3. Hallar el salario medio y la edad media en años para cada grupo de empleados con igual comisión. ... note:

La edad dependerá de la fecha en la que realicemos la consulta.

COMIS	SALARIO MEDIO	EDAD MEDIA
	1750	43
100	1367,14	49
120	600	47
90	1700	59
110	1733,33	46
50	2800	42
80	1910	42

4. Repite la consulta anterior expresando la edad en años cumplidos. (Aunque en este caso se obtiene lo mismo, la edad media podría variar de una consulta a otra dependiendo del momento en el que se realice la consulta).

COMIS	SALARIO MEDIO	EDAD MEDIA
	1750	43
100	1367,14	49
120	600	47
90	1700	58
110	1733,33	45
50	2800	42
80	1910	41

5. Hallar el salario medio y la edad media en años cumplidos para cada grupo de empleados del mismo departamento y con igual comisión.

NUMDE	COMIS	SALARIO MEDIO	EDAD MEDIA
100		1776,67	43
110	50	2800	42
110		1245	46
111	100	1444	48
111	110	1800	36
111	120	600	47
111		1150	37
112	80	1910	41
112	90	1700	58
112	100	1175	51
112	110	1700	50
NUMDE	COMIS	SALARIO MEDIO	EDAD MEDIA
112		1100	39
120		1790	49
121		2002,5	43

(continué en la próxima página)

(proviene de la página anterior)

122	1856	42
130	1950	43

6. Para los departamentos en los que hay algún empleado cuyo salario sea mayor que 2.500 € al mes, hallar el número de empleados y la suma de sus salarios.

NUMDE	COUNT (NUMEM)	SUM (SALAR)
100	3	5330
121	4	8010
110	3	5290
122	5	9280

Práctica 13: Agrupamiento de filas. CLÁUSULA HAVING

1. Hallar el número de empleados que usan la misma extensión telefónica. Solamente se desea mostrar aquellos grupos que tienen más de 1 empleado.

EXTEL	COUNT (NUMEM)
620	2
880	3
350	2
750	2
760	3
780	2

2. Para cada centro, hallar los presupuestos medios de los departamentos.

NUMCE	Presupuesto medio
20	70
10	30

3. Para cada centro, hallar los presupuestos medios de los departamentos clasificados según estén dirigidos en propiedad o en funciones.

NUMCE	T	Presupuesto medio
10	P	33
10	F	18
20	F	66
20	P	72

4. Para los departamentos cuyo salario medio supera al de la empresa, hallar cuántas extensiones telefónicas tienen.

NUMDE	n° EXTENSIONES TELEFÓNICAS
100	3
120	1
121	3
110	3
130	3
122	4

5. Hallar el máximo valor de la suma de los salarios de los departamentos.

NUMDE	SUM(SALAR)
111	10770

Práctica 14: Consultas sobre varias tablas

1. Para cada departamento con presupuesto inferior a 35.000 €, hallar le nombre del Centro donde está ubicado y el máximo salario de sus empleados (si dicho máximo excede de 1.500 €). Clasificar alfabéticamente por nombre de departamento.

NOMDE	NOMCE	MAX (SALAR)
FINANZAS	SEDE CENTRAL	2400
ORGANIZACIÓN	SEDE CENTRAL	1790
PERSONAL	SEDE CENTRAL	2600

2. Hallar por orden alfabético los nombres de los departamentos que dependen de los que tienen un presupuesto inferior a 30.000 €. También queremos conocer el nombre del departamento del que dependen y su presupuesto.

Departamento	Dpt. del que depende	PRESU
PERSONAL	ORGANIZACIÓN	18
PROCESO DE DATOS	ORGANIZACIÓN	18

3. Obtener los nombres y los salarios medios de los departamentos cuyo salario medio supera al salario medio de la empresa.

NOMDE	SALARIO MEDIO
ORGANIZACIÓN	1790
DIRECC. COMERCIAL	1763,33
FINANZAS	1950
PERSONAL	2002,5
DIRECCIÓN GENERAL	1776,67
PROCESO DE DATOS	1856

4. Para los departamentos cuyo director lo sea en funciones, hallar el número de empleados y la suma de sus salarios, comisiones y número de hijos.

NOMDE	COUNT (NUMEM)	SUM (SALAR)	SUM (COMIS)	SUM (NUMHI)
ORGANIZACIÓN	1	1790		3
SECTOR INDUSTRIAL	8	10770	730	8

5. Para los departamentos cuyo presupuesto anual supera los 35.000 €, hallar cuantos empleados hay por cada extensión telefónica.

NOMDE	EXTTEL	COUNT (NUMEM)
DIRECCIÓN GENERAL		200
DIRECCIÓN GENERAL		250
SECTOR INDUSTRIAL		760
SECTOR INDUSTRIAL		750
SECTOR INDUSTRIAL		780

(continué en la próxima página)

(proviene de la página anterior)

SECTOR SERVICIOS	810	1
PROCESO DE DATOS	620	2
DIRECC.COMERCIAL	500	1
PROCESO DE DATOS	660	1
DIRECC.COMERCIAL	508	1
SECTOR SERVICIOS	850	1
NOMDE EXTEL COUNT (NUMEM)		

PROCESO DE DATOS	610	1
SECTOR SERVICIOS	880	3
DIRECC.COMERCIAL	550	1
DIRECCIÓN GENERAL	220	1
SECTOR SERVICIOS	800	1
PROCESO DE DATOS	650	1
SECTOR SERVICIOS	840	1
SECTOR INDUSTRIAL	740	1

6. Hallar por orden alfabético los nombres de los empleados y su número de hijos para aquellos que son directores en funciones.

NOMEM	NUMHI

JULIO	0
MARCOS	2

7. Hallar si hay algún departamento (suponemos que sería de reciente creación) que aún no tenga empleados asignados ni director en propiedad.

-- no rows selected

8. Añadir un nuevo departamento de nombre NUEVO y con director en funciones.

-- no se muestra salida por ser una inserción.

9. Añadir un nuevo empleado de nombre NORBERTO y sin departamento asignado. Inventar el resto de datos.

-- no se muestra salida por ser una inserción.

10. Muestra los departamentos que no tienen empleados.

-- no rows selected

11. Muestra los nombres de departamentos que no tienen empleados haciendo uso la combinación externa LEFT JOIN. Muestra una segunda columna con los nombres de empleados para asegurarnos que realmente esta a NULL.

NOMDE

NUEVO

12. Muestra los nombres de departamentos que no tienen empleados haciendo uso la combinación externa RIGHT JOIN. Muestra una segunda columna con los nombres de empleados para asegurarnos que realmente esta a NULL.

NOMDE

NUEVO

13. Muestra los nombres de empleados que no tienen departamento haciendo uso la combinación externa LEFT JOIN. Muestra una segunda columna con los nombres de departamentos para asegurarnos que realmente esta a NULL.

NOMEM

NORBERTO

14. Muestra los nombres de empleados que no tienen departamento haciendo uso la combinación externa RIGHT JOIN. Muestra una segunda columna con los nombres de empleados para asegurarnos que realmente esta a NULL.

NOMEM

NORBERTO

15. Muestra los departamentos que no tienen empleados y los empleados que no tiene departamento haciendo uso la combinación externa FULL JOIN.

NOMDE	NOMEM
-----	-----
NUEVO	NORBERTO

16. Muestra los empleados y sus respectivos departamentos haciendo uso de la combinación interna INNER JOIN. ¿Aparecen el departamento NUEVO y el empleado NORBERTO? ¿Por qué?

NOMDE	NOMEM
-----	-----
PERSONAL	CESAR
SECTOR SERVICIOS	MARIO
SECTOR SERVICIOS	LUCIANO
PERSONAL	JULIO
SECTOR INDUSTRIAL	AUREO
DIRECC .COMERCIAL	MARCOS
PERSONAL	JULIANA
DIRECCIÓN GENERAL	PILAR
SECTOR INDUSTRIAL	LAVINIA
DIRECCIÓN GENERAL	ADRIANA
DIRECCIÓN GENERAL	ANTONIO
NOMDE	NOMEM
-----	-----
SECTOR SERVICIOS	OCTAVIO
FINANZAS	DOROTEA
PROCESO DE DATOS	OTILIA
ORGANIZACIÓN	GLORIA
FINANZAS	AUGUSTO
PROCESO DE DATOS	CORNELIO
SECTOR SERVICIOS	AMELIA
PROCESO DE DATOS	AURELIO
SECTOR INDUSTRIAL	DORINDA
PERSONAL	FABIOLA

(continué en la próxima página)

(proviene de la página anterior)

SECTOR SERVICIOS	MICAELA
NOMDE	NOMEM
DIRECC.COMERCIAL	CARMEN
SECTOR INDUSTRIAL	LUCRECIA
PROCESO DE DATOS	AZUCENA
FINANZAS	CLAUDIA
PROCESO DE DATOS	VALERIANA
SECTOR INDUSTRIAL	LIVIA
SECTOR SERVICIOS	SABINA
SECTOR INDUSTRIAL	DIANA
SECTOR SERVICIOS	HORACIO
SECTOR INDUSTRIAL	HONORIA
DIRECC.COMERCIAL	ROMULO
NOMDE	NOMEM
SECTOR INDUSTRIAL	SANCHO

17. Realiza la misma consulta anterior donde se cumpla la condición que NUMDE está a NULL. ¿Aparece algún resultado? ¿Por qué?

-- no rows selected

18. Muestra los empleados y sus respectivos departamentos haciendo uso de la combinación interna NATURAL JOIN.

NOMEM	NOMDE
CESAR	PERSONAL
MARIO	SECTOR SERVICIOS
LUCIANO	SECTOR SERVICIOS
JULIO	PERSONAL
AUREO	SECTOR INDUSTRIAL
MARCOS	DIRECC.COMERCIAL
JULIANA	PERSONAL
PILAR	DIRECCIÓN GENERAL
LAVINIA	SECTOR INDUSTRIAL
ADRIANA	DIRECCIÓN GENERAL
ANTONIO	DIRECCIÓN GENERAL
NOMEM	NOMDE
OCTAVIO	SECTOR SERVICIOS
DOROTEA	FINANZAS
OTILIA	PROCESO DE DATOS
GLORIA	ORGANIZACIÓN
AUGUSTO	FINANZAS
CORNELIO	PROCESO DE DATOS
AMELIA	SECTOR SERVICIOS
AURELIO	PROCESO DE DATOS
DORINDA	SECTOR INDUSTRIAL
FABIOLA	PERSONAL
MICAELA	SECTOR SERVICIOS

(continué en la próxima página)

(proviene de la página anterior)

NOMEM	NOMDE
CARMEN	DIRECC.COMERCIAL
LUCRECIA	SECTOR INDUSTRIAL
AZUCENA	PROCESO DE DATOS
CLAUDIA	FINANZAS
VALERIANA	PROCESO DE DATOS
LIVIA	SECTOR INDUSTRIAL
SABINA	SECTOR SERVICIOS
DIANA	SECTOR INDUSTRIAL
HORACIO	SECTOR SERVICIOS
HONORIA	SECTOR INDUSTRIAL
ROMULO	DIRECC.COMERCIAL
NOMEM	NOMDE
SANCHO	SECTOR INDUSTRIAL

19. Muestra la combinación de las 3 tablas CENTROS, DEPARTAMENTOS y EMPLEADOS haciendo uso de NATURAL JOIN.

NUMDE	NUMCE	NOMCE	...
121	10	SEDE CENTRAL	...
121	10	SEDE CENTRAL	...
121	10	SEDE CENTRAL	...
100	10	SEDE CENTRAL	...
100	10	SEDE CENTRAL	...
100	10	SEDE CENTRAL	...
130	10	SEDE CENTRAL	...
122	10	SEDE CENTRAL	...
120	10	SEDE CENTRAL	...
130	10	SEDE CENTRAL	...
...	...	...	...

20. Borra los registros dados de alta para el departamento NUEVO y el empleado introducida en el apartado anterior.

-- no se muestra salida por ser una eliminación.

--INTRODUCCIÓN A VISTAS

--1° HACEMOS UNA CONSULTA DONDE MUESTRE
 --PARA CADA EMPLEADO SU NÚMERO DE EMPLEADO,
 --NOMBRE, NUMHI Y NOMBRE DEL DEPARTAMENTO
 --EN EL QUE TRABAJA

```
SELECT numem, NOMEM, numhi, NOMDE
FROM EMPLEADOS E, DEPARTAMENTOS D
WHERE E.numde=D.numde;
```

--2° CREAMOS UNA VISTA LLAMADA EJEMPLO1
 --CON LA CONSULTA ANTERIOR

```
CREATE VIEW EJEMPLO1 AS
SELECT numem, NOMEM, numhi, NOMDE
FROM EMPLEADOS E, DEPARTAMENTOS D
WHERE E.numde=D.numde;
```

(continué en la próxima página)

(proviene de la página anterior)

```
--OBTENER EL NOMBRE DE CADA EMPLEADO
--Y EL NÚMERO DE HIJOS QUE TIENE Y CREAR
--UNA VISTA LLAMADA EJEMPLO2

SELECT NOMEM, NUMHI
FROM EMPLEADOS;

CREATE VIEW EJEMPLO2 AS
SELECT NOMEM, NUMHI FROM EMPLEADOS;

--HACEMOS LA MISMA VISTA ANTERIOR
--CON OTRO NOMBRE, PARA MOSTRAR TAMBIÉN
--EL NUMEM

CREATE VIEW EJEMPLO3 AS
SELECT NUMEM,NOMEM, NUMHI FROM EMPLEADOS;
```

Práctica 15: Vistas

1. Crear una vista con todos los empleados del departamento 111 en donde figuren solo el número de empleado, su nombre, su salario y la comisión. La llamarás VISTA1.
2. Crear una vista que obtenga el máximo valor de la suma de los salarios de los departamentos. Se llamará VISTA2.
3. Utilizar la vista anterior para obtener el departamento con más gasto en salario.

NUMDE
111

4. Utilizar la VISTA1 para obtener por orden alfabético los nombres de los empleados del departamento 111 que tienen comisión.

NOMEM
AUREO
DIANA
DORINDA
HONORIA
LAVINIA
LIVIA
SANCHO

5. Insertar la siguiente fila en la VISTA1: (999,"RODOLFO",999,999). ¿Qué consecuencias tiene?
6. Borra la fila anterior.
7. Crear una VISTA3 en la que aparezcan los centros con sus departamentos.
8. Utilizar la VISTA3 para mostrar el nombre de cada centro y el total de los presupuestos de sus departamentos.

NOMCE	Total presupuestos
RELACIÓN CON CLIENTES	210
SEDE CENTRAL	150

9. Insertar la siguiente fila en la VISTA3: (30,"SUCURSAL ÉCIJA",200,120,"F",20,110,"CONTABILIDAD"). ¿Qué ocurre?

10. Borra la fila anterior

Práctica 16: Repaso

1. Selecciona, por orden alfabético decreciente, el nombre de los empleados junto con su salario aumentado un 1 %, para aquellos empleados del departamento 100 que en la fecha de su contratación tenían más de 20 años.

NOMEM	SALAR+0.01*SALAR
PILAR	1929,1
ANTONIO	727,2
ADRIANA	2727

2. Para cada Centro selecciona el presupuesto medio de los departamentos que tienen su sede en él.

NUMCE	AVG (PRESU)
20	70
10	30

3. Selecciona el nombre de los empleados junto con su edad actual para aquellos empleados que trabajan en el departamento de PERSONAL.

NOMEM	EDAD
CESAR	46
JULIO	44
JULIANA	44
FABIOLA	39

4. Selecciona la dirección del centro donde están ubicados los departamentos que tiene empleados con más de tres hijos. Deberás mostrar también el nombre de dichos departamentos.

DIRCE	NOMDE
C/ ATOCHA, 820, MADRID	DIRECCIÓN GENERAL
C/ ATOCHA, 820, MADRID	PERSONAL
C/ ATOCHA, 820, MADRID	FINANZAS

5. Selecciona la dirección del centro donde están ubicados los departamentos si existe alguno que tiene empleados con más de tres hijos. Deberás mostrar también el nombre de dichos departamentos.

DIRCE	NOMDE
C/ ATOCHA, 405, MADRID	DIRECC.COMERCIAL
C/ ATOCHA, 820, MADRID	DIRECCIÓN GENERAL
C/ ATOCHA, 405, MADRID	SECTOR SERVICIOS
C/ ATOCHA, 820, MADRID	ORGANIZACIÓN

(continué en la próxima página)

(proviene de la página anterior)

C/ ATOCHA, 820, MADRID	PROCESO DE DATOS
C/ ATOCHA, 820, MADRID	PERSONAL
C/ ATOCHA, 820, MADRID	FINANZAS
C/ ATOCHA, 405, MADRID	SECTOR INDUSTRIAL

6. Cuenta el número de empleados que tienen el mismo número de hijos. Deberás mostrar también el número de hijos que corresponde en cada caso.

NUMHI	NºEmpleados
1	7
6	1
2	6
4	1
5	1
3	4
0	14

7. Crea una vista llamada “Sin comisión” donde muestres el nombre, la edad y el salario de los empleados que no tienen comisión. El salario deberá aparecer en la consulta seguido de “€” y el nombre del campo en el que aparezca la edad será “Edad actual”.

NOMEM	EDADACTUAL	SALARIO
CESAR	46	1800 €
JULIO	44	2600 €
JULIANA	44	1750 €
PILAR	46	1910 €
ADRIANA	40	2700 €
ANTONIO	43	720 €
DOROTEA	39	1500 €
OTILIA	37	1910 €
GLORIA	49	1790 €
AUGUSTO	40	1950 €
CORNELIO	39	2400 €
NOMEM	EDADACTUAL	SALARIO
AURELIO	37	2700 €
FABIOLA	39	1860 €
MICAELA	39	1100 €
CARMEN	41	1290 €
LUCRECIA	37	1150 €
AZUCENA	48	1010 €
CLAUDIA	50	2400 €
VALERIANA	49	1260 €
ROMULO	50	1200 €

8. Utiliza la vista anterior para calcular el salario medio de los empleados que no tienen comisión.

MEDIASALARIOS
1750

9. Selecciona el nombre de los departamentos en los que trabajan empleados cuyo salario máximo no supere los 2000 €.

NOMDE	MAX (SALAR)
ORGANIZACIÓN	1790
SECTOR SERVICIOS	1910
SECTOR INDUSTRIAL	1800

10. Crea una vista con el nombre “Jubilación” donde muestres el nombre de cada empleado, el nombre del departamento en el que trabajan, su edad y su salario para aquellos cuya edad sea, al menos, de 60 años.

11. Utiliza la vista anterior para mostrar el nombre de los empleados que tienen justo 60 años.

no rows selected

12. Muestra la dirección de los centros, el nombre de los empleados que trabajan en él, el nombre del departamento concreto en el que trabajan y quien es el director de dicho departamento para aquellos empleados cuyo nombre comience por la letra “J”.

DIRCE	NOMEM	NOMDE	DIREC
C/ ATOCHA, 820, MADRID	JULIANA	PERSONAL	150
C/ ATOCHA, 820, MADRID	JULIO	PERSONAL	150

MODIFICACIÓN DE BASES DE DATOS. LENGUAJE DE MANIPULACIÓN DE DATOS

5.1 INTRODUCCIÓN

En esta unidad veremos 3 sentencias SQL DML para la modificación de datos. Además aprenderemos el funcionamiento de las transacciones y realizaremos una introducción al lenguaje **PL/SQL**.

5.2 LENGUAJE DE MANIPULACIÓN DE DATOS: DML

Una vez que se ha creado de forma conveniente las tablas, el siguiente paso consiste en insertar datos en ellas, es decir, añadir tuplas. Durante la vida de la base de datos será necesario, además, borrar determinadas tuplas o modificar los valores que contienen. Los comandos de SQL que se van a estudiar en este apartado son **INSERT**, **UPDATE** y **DELETE**. Estos comandos pertenecen al **DML**.

5.2.1 Inserción de datos

El comando **INSERT** de SQL permite introducir datos en una tabla o en una vista de la base de datos. La sintaxis del comando es la siguiente:

```
INSERT INTO {nombre_tabla | nombre_vista} [(columna1 [, columna2]...)]  
VALUES (valor1 [, valor2] ... );
```

Indicando la tabla se añaden los datos que se especifiquen tras el apartado **VALUES** en un nuevo registro. Los valores deben corresponderse con el orden de las columnas. Si no es así se puede indicar tras el nombre de la tabla y entre paréntesis.

Ejemplos:

Supongamos que tenemos el siguiente diseño físico de una tabla:

```
CREATE TABLE EMPLEADOS (  
  COD          NUMBER(2) PRIMARY KEY,  
  NOMBRE       VARCHAR2(50) NOT NULL,  
  LOCALIDAD    VARCHAR2(50) DEFAULT 'Écija',  
  FECHANAC     DATE  
);
```

La forma más habitual de introducir datos es la siguiente:

```
INSERT INTO EMPLEADOS VALUES (1, 'Pepe', 'Osuna', '01/01/1970');  
INSERT INTO EMPLEADOS VALUES (2, 'Juan', DEFAULT, NULL);  
INSERT INTO EMPLEADOS VALUES (3, 'Sara', NULL, NULL);
```

Es obligatorio introducir valores para los campos COD y NOMBRE. Dichos campos no pueden tener valor NULL. Podemos insertar sólo el valor de ciertos campos. En este caso hay que indicar los campos a insertar y el orden en el que los introducimos:

```
INSERT INTO EMPLEADOS (NOMBRE, COD) VALUES ('Ana', 5);
```

Inserción de datos obtenidos de una consulta

También es posible insertar datos en una tabla que hayan sido obtenidos de una consulta realizada a otra tabla/vista u otras tablas/vistas. Su forma es:

```
INSERT INTO tabla  
SELECT ...
```

Debe respetarse lo dicho anteriormente respecto a los campos. La consulta SELECT debe devolver la misma cantidad y tipo de campos que los definidos en la tabla.

Por ejemplo, suponiendo que disponemos de una tabla SOLICITANTES con el siguiente diseño:

```
CREATE TABLE SOLICITANTES (  
  NUM          NUMBER(2) PRIMARY KEY,  
  NOMBRE       VARCHAR2(50),  
  CIUDAD       VARCHAR2(50),  
  NACIMIENTO   DATE,  
  ESTUDIOS     VARCHAR2(50)  
);  
  
INSERT INTO EMPLEADOS  
SELECT NUM, NOMBRE, CIUDAD, NACIMIENTO  
FROM SOLICITANTES  
WHERE ESTUDIOS='CFGS ASIR';
```

También podemos indicar los campos a insertar, teniendo en cuenta que, en este caso los campos COD y NOMBRE de la tabla EMPLEADO no aceptan valores NULL, por tanto es obligatorio introducir valores para ellos:

```
INSERT INTO EMPLEADOS (FECHANAC, NOMBRE, COD)  
SELECT NACIMIENTO, NOMBRE, NUM  
FROM SOLICITANTES  
WHERE ESTUDIOS='CFGS ASIR';
```

5.2.2 Modificación de datos

Para la modificación de registros dentro de una tabla o vista se utiliza el comando UPDATE. La sintaxis del comando es la siguiente:

```
UPDATE {nombre_tabla | nombre_vista}
SET columna1=valor1 [, columna2=valor2] ...
[WHERE condición];
```

Se modifican las columnas indicadas en el apartado SET con los valores indicados. La cláusula WHERE permite especificar qué registros serán modificados.

Ejemplos:

```
-- Ponemos todos los nombres a mayúsculas
-- y todas las localidades a Estepa
UPDATE EMPLEADOS
SET NOMBRE=UPPER(NOMBRE), LOCALIDAD='Estepa';

-- Para los empleados que nacieron a partir de 1970
-- ponemos nombres con inicial mayúscula y localidades Marchena
UPDATE EMPLEADOS
SET NOMBRE=INITCAP(NOMBRE), LOCALIDAD='Marchena'
WHERE FECHANAC >= '01/01/1970';
```

Actualización de datos usando una subconsulta

También se admiten subconsultas. Por ejemplo:

```
UPDATE empleados
SET sueldo=sueldo*1.10
WHERE id_seccion = (SELECT id_seccion FROM secciones
                    WHERE nom_seccion='Producción');
```

Esta instrucción aumenta un 10 % el sueldo de los empleados que están dados de alta en la sección llamada Producción.

5.2.3 Eliminación de datos

Es más sencilla que el resto, elimina los registros de la tabla que cumplan la condición indicada. Se realiza mediante la instrucción DELETE:

```
DELETE [ FROM ] {nombre_tabla|nombre_vista}
[WHERE condición] ;
```

Ejemplos:

```
-- Borramos empleados de Estepa
DELETE EMPLEADOS
WHERE LOCALIDAD='Estepa';

-- Borramos empleados cuya fecha de nacimiento sea anterior a 1970
-- y localidad sea Osuna
DELETE EMPLEADOS
WHERE FECHANAC < '01/01/1970' AND LOCALIDAD = 'Osuna';

-- Borramos TODOS los empleados;
DELETE EMPLEADOS;
```

Hay que tener en cuenta que el borrado de un registro no puede provocar fallos de integridad y que la opción de integridad ON DELETE CASCADE (clave secundaria o foránea) hace que no sólo se borren los registros indicados sino todos los relacionados. En la práctica esto significa que no se pueden borrar registros cuya clave primaria sea

referenciada por alguna clave foránea en otra tabla, a no ser que dicha tabla secundaria tenga activada la cláusula ON DELETE CASCADE en su clave foránea, en cuyo caso se borraría el/los registro/s de la tabla principal y los registros de tabla secundaria cuya clave foránea coincide con la clave primaria eliminada en la tabla primera.

Eliminación de datos usando una subconsulta

Al igual que en el caso de las instrucciones INSERT o SELECT, DELETE dispone de cláusula WHERE y en dicha cláusula podemos utilizar subconsultas. Por ejemplo:

```
DELETE empleados
WHERE id_empleado IN (SELECT id_empleado FROM operarios);
```

En este caso se trata de una subconsulta creada con el operador IN, se eliminarán los empleados cuyo identificador esté dentro de la tabla operarios.

5.2.4 Ejecución de comandos DML sobre vistas.

Las instrucciones DML ejecutadas sobre las vistas permiten añadir o modificar los datos de las tablas relacionados con las filas de la vista. Ahora bien, no es posible ejecutar instrucciones DML sobre vistas que:

- Utilicen funciones de grupo (SUM, AVG,...)
- Usen GROUP BY o DISTINCT
- Posean columnas con cálculos (P. ej: PRECIO * 1.16)

Además no se pueden añadir datos a una vista si en las tablas referencias en la consulta SELECT hay campos NOT NULL que no aparecen en la consulta (es lógico ya que al añadir el dato se tendría que añadir el registro colocando el valor NULL en el campo).

Si tenemos la siguiente vista:

```
CREATE VIEW resumen (id_localidad, localidad, poblacion,
                    n_provincia, provincia, superficie,
                    id_comunidad, comunidad)
AS SELECT L.IdLocalidad, L.Nombre, L.Poblacion,
        P.IdProvincia, P.Nombre, P.Superficie,
        C.IdComunidad, C.Nombre
FROM LOCALIDADES L JOIN PROVINCIAS P ON L.IdProvincia=P.IdProvincia
JOIN COMUNIDADES C ON P.IdComunidad=C.IdComunidad;
```

Si realizamos la siguiente inserción

```
INSERT INTO resumen (id_localidad, localidad, poblacion)
VALUES (10000, 'Sevilla', 750000);
```

Se producirá un error, puesto que estamos insertando un registro dentro de la vista donde muchos de sus campos no tienen especificado un valor y por tanto serán insertados a NULL. El problema es que no puede insertarse un NULL en n_provincia ni id_comunidad puesto que son claves primarias de las tablas subyacentes PROVINCIAS y COMUNIDADES. La solución al problema anterior se soluciona creando un disparador (trigger) de sustitución, que veremos en el apartado de triggers.

5.3 GESTIÓN DE TRANSACCIONES

En términos teóricos, una transacción es un conjunto de tareas relacionadas que se realizan de forma satisfactoria o incorrecta como una unidad. En términos de procesamiento, las transacciones se confirman o se anulan. Para que una

transacción se confirme se debe garantizar la permanencia de los cambios efectuados en los datos. Los cambios deben conservarse aunque el sistema se bloquee o tengan lugar otros eventos imprevistos. Existen 4 propiedades necesarias, que son conocidas como **propiedades ACID**:

- atomicidad (Atomicity)
- coherencia (Consistency)
- aislamiento (Isolation)
- permanencia (Durability).

Estas propiedades garantizan un comportamiento predecible, reforzando la función de las transacciones como proposiciones de todo o nada.

- **Atomicidad:** Una transacción es una unidad de trabajo el cual se realiza en su totalidad o no se realiza en ningún caso. Las operaciones asociadas a una transacción comparten normalmente un objetivo común y son interdependientes. Si el sistema ejecutase únicamente una parte de las operaciones, podría poner en peligro el objetivo final de la transacción.
- **Coherencia:** Una transacción es una unidad de integridad porque mantiene la coherencia de los datos, transformando un estado coherente de datos en otro estado de datos igualmente coherente.
- **Aislamiento:** Una transacción es una unidad de aislamiento, permitiendo que transacciones concurrentes se comporten como si cada una fuera la única transacción que se ejecuta en el sistema. El aislamiento requiere que parezca que cada transacción sea la única que manipula el almacén de datos, aunque se puedan estar ejecutando otras transacciones al mismo tiempo. Una transacción nunca debe ver las fases intermedias de otra transacción.
- **Permanencia:** Una transacción también es una unidad de recuperación. Si una transacción se realiza satisfactoriamente, el sistema garantiza que sus actualizaciones se mantienen aunque el equipo falle inmediatamente después de la confirmación. El registro especializado permite que el procedimiento de reinicio del sistema complete las operaciones no finalizadas, garantizando la permanencia de la transacción.

En términos más prácticos, una transacción está formada por una serie de instrucciones DML. Una transacción comienza con la primera instrucción DML que se ejecute y finaliza con una operación COMMIT (si la transacción se confirma) o una operación ROLLBACK (si la operación se cancela). Hay que tener en cuenta que cualquier instrucción DDL o DCL da lugar a un COMMIT implícito, es decir todas las instrucciones DML ejecutadas hasta ese instante pasan a ser definitivas.

Para poder hacer uso de transacciones en SQL*Plus debemos tener desactivado el modo AUTOCOMMIT. Podemos ver su estado con la orden:

```
SHOW AUTOCOMMIT
```

Para desactivar dicho modo, usamos la orden:

```
SET AUTOCOMMIT OFF
```

5.3.1 COMMIT

La instrucción COMMIT hace que los cambios realizados por la transacción sean definitivos, irrevocables. Se dice que tenemos una **transacción confirmada**. Sólo se debe utilizar si estamos de acuerdo con los cambios, conviene asegurarse mucho antes de realizar el COMMIT ya que las instrucciones ejecutadas pueden afectar a miles de registros. Además el cierre correcto de la sesión da lugar a un COMMIT, aunque siempre conviene ejecutar explícitamente esta instrucción a fin de asegurarnos de lo que hacemos.

5.3.2 ROLLBACK

Esta instrucción regresa a la instrucción anterior al inicio de la transacción, normalmente el último COMMIT, la última instrucción DDL o DCL o al inicio de sesión. Anula definitivamente los cambios, por lo que conviene también asegurarse de esta operación. Un abandono de sesión incorrecto o un problema de comunicación o de caída del sistema dan lugar a un ROLLBACK implícito.

5.3.3 SAVEPOINT

Esta instrucción permite establecer un punto de ruptura. El problema de la combinación ROLLBACK/COMMIT es que un COMMIT acepta todo y un ROLLBACK anula todo. SAVEPOINT permite señalar un punto intermedio entre el inicio de la transacción y la situación actual. Su sintaxis es:

```
-- ... instrucciones DML ...

SAVEPOINT nombre

-- ... instrucciones DML ...
```

Para regresar a un punto de ruptura concreto se utiliza ROLLBACK TO SAVEPOINT seguido del nombre dado al punto de ruptura. También es posible hacer ROLLBACK TO nombre de punto de ruptura. Cuando se vuelve a un punto marcado, las instrucciones que siguieron a esa marca se anulan definitivamente.

Ejemplo de uso:

```
SET AUTOCOMMIT OFF;
```

```
CREATE TABLE T (FECHA DATE);

INSERT INTO T VALUES ('01/01/2017');
INSERT INTO T VALUES ('01/02/2017');

SAVEPOINT febrero;

INSERT INTO T VALUES ('01/03/2017');
INSERT INTO T VALUES ('01/04/2017');

SAVEPOINT abril;

INSERT INTO T VALUES ('01/05/2017');

ROLLBACK TO febrero;
-- También puede escribirse ROLLBACK TO SAVEPOINT febrero;
-- En este ejemplo sólo se guardan en la tabla
-- los 2 primeros registros o filas.
```

5.3.4 Estado de los datos durante la transacción

Si se inicia una transacción usando comandos DML hay que tener en cuenta que:

- Se puede volver a la instrucción anterior a la transacción cuando se desee.
- Las instrucciones de consulta SELECT realizadas por el usuario que inició la transacción muestran los datos ya modificados por las instrucciones DML.

- El resto de usuarios ven los datos tal cual estaban antes de la transacción, de hecho los registros afectados por la transacción aparecen bloqueados hasta que la transacción finalice. Esos usuarios no podrán modificar los valores de dichos registros.
- Tras la transacción todos los usuarios ven los datos tal cual quedan tras el fin de transacción. Los bloqueos son liberados y los puntos de ruptura borrados.

5.3.5 Concurrencia de varias transacciones (Bloqueos)

Cuando se realizan varias transacciones de forma simultánea, pueden darse diversas situaciones en el acceso concurrente a los datos, es decir, cuando se accede a un mismo dato en dos transacciones distintas. Estas situaciones son:

- **Lectura sucia** (Dirty Read). Una transacción lee datos que han sido escritos por otra transacción que aún no se ha confirmado.
- **Lectura no repetible** (Non-repeatable Read). Una transacción vuelve a leer los datos que ha leído anteriormente y descubre que otra transacción confirmada ha modificado o eliminado los datos.
- **Lectura fantasma** (Phantom Read). Una transacción vuelve a ejecutar una consulta que devuelve un conjunto de filas que satisface una condición de búsqueda y descubre que otra transacción confirmada ha insertado filas adicionales que satisfacen la condición.

Para una mejor gestión de estas situaciones debemos indicar el nivel de aislamiento que deseamos. De las cuatro propiedades de ACID de un SGBD, la **propiedad de aislamiento** es la más laxa. Un nivel de aislamiento bajo aumenta la capacidad de muchos usuarios para acceder a los mismos datos al mismo tiempo, pero también aumenta el número de efectos de concurrencia (como lecturas sucias). Un mayor nivel de aislamiento puede dar como resultado una pérdida de concurrencia y el aumento de las posibilidades de que una transacción bloquee a otra.

Podemos solicitar al SGBD cuatro niveles de aislamiento. De menor a mayor nivel de aislamiento, tenemos:

- **READ UNCOMMITTED** (Lectura no confirmada). Las sentencias SELECT son efectuadas sin realizar bloqueos, por tanto, todos los cambios hechos por una transacción pueden verlos las otras transacciones. Permite que sucedan las 3 situaciones indicadas previamente: lecturas fantasma, no repetibles y sucias.
- **READ COMMITTED** (Lectura confirmada). Los datos leídos por una transacción pueden ser modificados por otras transacciones. Se pueden dar lecturas fantasma y lecturas no repetibles.
- **REPEATABLE READ** (Lectura repetible). Consiste en que ningún registro leído con un SELECT se puede cambiar en otra transacción. Solo pueden darse lecturas fantasma.
- **SERIALIZABLE**. Las transacciones ocurren de forma totalmente aislada a otras transacciones. Se bloquean las transacciones de tal manera que ocurren unas detrás de otras, sin capacidad de concurrencia. El SGBD las ejecuta concurrentemente si puede asegurar que no hay conflicto con el acceso a los datos.

Nivel de aislamiento y Lecturas

Nivel de aislamiento	Lecturas sucias	Lecturas no repetibles	Lecturas fantasma
READ UNCOMMITTED	SÍ	SÍ	SÍ
READ COMMITTED	NO	SÍ	SÍ
REPEATABLE READ	NO	NO	SÍ
SERIALIZABLE	NO	NO	NO

Internamente el SGBD proporciona dicho nivel de aislamiento mediante **bloqueos** en los datos.

En Oracle, el nivel por defecto es **READ COMMITTED**. Además de éste, solo permite **SERIALIZABLE**. Se puede cambiar ejecutando el comando:

<code>SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;</code>
--

5.4 INTRODUCCIÓN A PL/SQL

Cuando se desea realizar una aplicación completa para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. **PL/SQL** es el lenguaje de programación que proporciona Oracle para extender el SQL estándar con otro tipo de instrucciones.

Casi todos los grandes Sistemas Gestores de Datos incorporan utilidades que permiten ampliar el lenguaje SQL para producir pequeñas utilidades que añaden al SQL mejoras de la programación estructurada (bucles, condiciones, funciones,...).

A diferencia de SQL, que es un lenguaje declarativo (indicamos qué deseamos obtener sin indicar cómo), PL/SQL es el lenguaje procedimental (indicamos cómo queremos obtener los resultados)

PL/SQL es implementado por el precompilador de Oracle que permite utilizar condiciones y bucles al estilo de lenguajes como Basic, Cobol, C++, Java, etc.

En otros sistemas gestores de bases de datos existen otros lenguajes procedimentales:

- Microsoft SQL Server utiliza **Transact SQL**
- PostgreSQL usa **PL/pgSQL**
- IBM Informix usa **SPL**
- Firebird usa **PSQL**
- MySQL usa **SQL/PSM**
- El estándar ANSI/ISO indica **SQL/PSM**

El código PL/SQL puede almacenarse:

- En la propia base de datos
- En archivos externos

Las funciones más destacadas que pueden realizarse con PL/SQL son las siguientes:

- Facilitar la realización de tareas administrativas sobre la base de datos (copia de valores antiguos, auditorías, control de usuarios,...)
- Validación y verificación avanzada de usuarios
- Consultas muy avanzadas
- Tareas imposibles de realizar con SQL

5.4.1 Conceptos básicos

- bloque PL/SQL Se trata de un trozo de código que puede ser interpretado por Oracle. Se encuentra inmerso dentro de las palabras BEGIN y END.
- programa PL/SQL Conjunto de bloques que realizan una determinada labor.
- procedimiento Programa PL/SQL almacenado en la base de datos y que puede ser ejecutado si se desea con solo saber su nombre (y teniendo permiso para su acceso).

- función Programa PL/SQL que a partir de unos datos de entrada obtiene un resultado (datos de salida). Una función puede ser utilizada en cualquier expresión desde cualquier otro programa PL/SQL e incluso desde una instrucción SQL.
- paquete Colección de procedimientos y funciones agrupados dentro de la misma estructura. Similar a las bibliotecas y librerías de los lenguajes convencionales.
- trigger (disparador) Programa PL/SQL que se ejecuta automáticamente cuando ocurre un determinado suceso a un objeto de la base de datos.

5.4.2 Bloques

Cada programa en PL/SQL está formado por grupos de órdenes SQL llamadas bloques. Cada bloque puede contener, a su vez, nuevos bloques.

ESTRUCTURA DE UN BLOQUE PL/SQL

La estructura más sencilla es la siguiente:

```
BEGIN
  Sentencias
  ...
END;
/
```

Ejemplo:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('Hola Mundo');
END;
/
```

La barra / se utiliza para ejecutar el código.

NOTA IMPORTANTE

Si usas SQL*Plus deberás ejecutar al inicio de sesión la siguiente orden para que se habilite la salida:

```
SET SERVEROUTPUT ON
```

La estructura general es:

```
[ DECLARE
  constantes,
  variables,
  cursores,
  excepciones definidas por el usuario
  ...
]
BEGIN
  Sentencias
[ EXCEPTION
  Acciones a realizar cuando se produce alguna excepción ]
END;
/
```

Ejemplo:

```
DECLARE
    fecha DATE;
BEGIN
    SELECT sysdate INTO fecha FROM dual;
    dbms_output.put_line
    (
        to_CHAR(fecha, 'day', 'dd' de 'month' de 'yyyy', a las 'hh24:mi:ss')
    );
END;
/
```

NOTA IMPORTANTE

Dentro de un bloque BEGIN ... END la sentencia SELECT adquiere la forma

```
SELECT campos INTO variable ...
```

Tipos de Bloques

- Anónimos (anonymous blocks): Se construyen normalmente de manera dinámica para un objetivo muy concreto y se ejecutan, en general, una única vez. Por eso no llevan nombre.
- Nominados (named blocks): Son bloques a los que se les pone un nombre. También se conocen como subprogramas. Los subprogramas pueden ser procedimientos o funciones.
 - Procedimientos: Se construyen para efectuar algún tipo de operación más o menos frecuente y se almacenan para ejecutarlos cuantas veces se desee. Se ejecutan con una llamada al procedimiento.
 - Funciones: Son similares a los procedimientos. Al igual que estos realizan algún tipo de operación, pero además las funciones devuelven un valor que puede ser usado en cualquier sentencia PL/SQL e incluso en sentencias SQL.
- Paquetes: Se usan para agrupar procedimientos y funciones. Facilitan la descomposición modular y el mantenimiento.
- Disparadores (triggers): Son bloques nominados que se almacenan en la BD. Su ejecución está condicionada a cierta condición, como por ejemplo usar una orden concreta del DML.
- Comentarios: Pueden incluirse siempre que se desee.
 - Monolínea: Empiezan con 2 guiones – y terminan a final de línea.
 - Multilínea: Empiezan con /* y terminan con */ (como en C).

Ejemplo de Bloque Anónimo

```
DECLARE
    fecha DATE;
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Salida de información');
    SELECT SYSDATE INTO fecha FROM DUAL;
    DBMS_OUTPUT.PUT_LINE ('Fecha: ' || fecha);
END;
/
```

Los otros tipos de bloques los veremos con mayor detenimiento más adelante.

5.4.3 Ejecución selectiva: Condicionales

Para ejecutar un serie de instrucciones según se cumpla o no una condición tenemos dos estructuras:

- **IF**
- **CASE**

Estas estructuras necesitan que indiquemos la condición o condiciones que deseamos evaluar. Dicha condición se evalúa en la mayoría de los lenguajes de programación como TRUE o FALSE. En SQL se usa una lógica trivaluada (TRUE, FALSE y NULL), donde NULL tiene el significado de "desconocido o no definido". Cualquier expresión relacional con un operando nulo, devuelve NULL. Tablas de Verdad:

NOT		AND	T	N	F	OR	T	N	F
T	F	T	T	N	F	T	T	T	T
N	N	N	N	N	F	N	T	N	N
F	T	F	F	F	F	F	T	N	F

Estructura IF-THEN-ELSE

Formato:

```
IF ExpresiónBooleana1 THEN
  SecuenciaÓrdenes1;
[ ELSIF ExpresiónBooleana2 THEN
  SecuenciaÓrdenes2; ]
...
[ ELSE
  SecuenciaÓrdenes; ]
END IF;
```

Como se muestra, las cláusulas ELSIF y ELSE son opcionales y puede haber tantas cláusulas ELSIF como se desee. Se ejecuta la SecuenciaÓrdenes1 si ExpresiónBooleana1 es TRUE. Si esa expresión vale FALSE o NULL, no se ejecutará y pasará a ejecutar las siguientes cláusulas.

Los valores NULL hacen que no sea equivalente intercambiar las secuencias de órdenes si se niega la ExpresiónBooleana1.

Ejemplos

Antes de pasar a ver el ejemplo vamos dar unas indicaciones sobre la importancia de comprobar previamente valores nulos, si los hubiera.

```
IF A < B THEN
  C := 1;
ELSE
  C := 2;
END IF;

-- NO ES EQUIVALENTE A LO SIGUIENTE, si las variables A o B
-- pueden tener valores NULL:

IF A >= B THEN
  C := 2;
ELSE
```

(continúe en la próxima página)

(proviene de la página anterior)

```
C := 1;
END IF;

-- LO MEJOR ES COMPROBAR PRIMERO si las variables A o B
-- tienen valores NULL

IF A IS NULL OR B IS NULL THEN
    C := 3;
ELSIF A < B THEN
    C := 1;
ELSE
    C := 2;
END IF;
```

El código completo del ejemplo anterior es:

```
SET SERVEROUTPUT ON

DECLARE
    A NUMBER := NULL;
    B NUMBER := 2;
    C NUMBER;

BEGIN

    IF A IS NULL OR B IS NULL THEN
        C := 3;
    ELSIF A < B THEN
        C := 1;
    ELSE
        C := 2;
    END IF;
    DBMS_OUTPUT.PUT_LINE ('El valor de C es ' || C);

END;
/
```

Y la salida sería:

El valor de C es 3

Ejemplo:

```
DECLARE
    nota NUMBER(2);

BEGIN
    nota := 7;

    IF nota = 10 OR nota = 9 THEN
        DBMS_OUTPUT.PUT_LINE('Sobresaliente');
    ELSIF nota = 8 OR nota = 7 THEN
        DBMS_OUTPUT.PUT_LINE('Notable');
    ELSIF nota = 6 THEN
        DBMS_OUTPUT.PUT_LINE('Bien');
    ELSIF nota = 5 THEN
        DBMS_OUTPUT.PUT_LINE('Suficiente');
```

(continué en la próxima página)

(proviene de la página anterior)

```

ELSIF nota < 5 AND nota >=0 THEN
    DBMS_OUTPUT.PUT_LINE('Insuficiente');
ELSE
    DBMS_OUTPUT.PUT_LINE('Nota no válida');
END IF;

END;
/

```

Estructura CASE

La estructura CASE tiene la misma finalidad que la estructura IF vista anteriormente. Es decir, para realizar una operación de selección podemos hacer uso de IF o de CASE: son equivalentes. A diferencia de IF, la estructura CASE no está limitada a expresiones booleanas. La evaluación de la expresión puede ser, y a menudo es, un valor numérico o texto.

Su sintaxis es la siguiente:

```

CASE Expresion
  WHEN valor1 THEN
    Secuencia_de_órdenes1;
  [ WHEN valor2 THEN
    Secuencia_de_órdenes2; ]
  ...
  [ ELSE
    Secuencia_de_órdenes; ]
END CASE;

```

Ejemplos:

El mismo caso anterior, esta vez con CASE.

```

DECLARE
    nota NUMBER(2);
BEGIN
    nota := 7;
    CASE nota
        WHEN 10 THEN DBMS_OUTPUT.PUT_LINE('Sobresaliente');
        WHEN 9 THEN DBMS_OUTPUT.PUT_LINE('Sobresaliente');
        WHEN 8 THEN DBMS_OUTPUT.PUT_LINE('Notable');
        WHEN 7 THEN DBMS_OUTPUT.PUT_LINE('Notable');
        WHEN 6 THEN DBMS_OUTPUT.PUT_LINE('Bien');
        WHEN 5 THEN DBMS_OUTPUT.PUT_LINE('Suficiente');
        WHEN 4 THEN DBMS_OUTPUT.PUT_LINE('Insuficiente');
        WHEN 3 THEN DBMS_OUTPUT.PUT_LINE('Insuficiente');
        WHEN 2 THEN DBMS_OUTPUT.PUT_LINE('Insuficiente');
        WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('Insuficiente');
        WHEN 0 THEN DBMS_OUTPUT.PUT_LINE('Insuficiente');
        ELSE DBMS_OUTPUT.PUT_LINE('Nota no válida');
    END CASE;
END;
/

```

O de forma más resumida:

```

DECLARE
    nota NUMBER(2);
BEGIN
    nota := 7;
    CASE
        WHEN nota=10 OR nota=9 THEN DBMS_OUTPUT.PUT_LINE('Sobresaliente');
        WHEN nota=8 OR nota=7 THEN DBMS_OUTPUT.PUT_LINE('Notable');
        WHEN nota=6 THEN DBMS_OUTPUT.PUT_LINE('Bien');
        WHEN nota=5 THEN DBMS_OUTPUT.PUT_LINE('Suficiente');
        WHEN nota<5 AND nota>=0 THEN DBMS_OUTPUT.PUT_LINE('Insuficiente');
        ELSE DBMS_OUTPUT.PUT_LINE('Nota no válida');
    END CASE;
END;
/

```

Observa como en este segundo caso no hay nada inmediatamente después de CASE.

5.4.4 Ejecución repetitiva: Bucles

Para realizar una operación un número elevado de veces utilizamos bucles. Un bucle es una estructura del lenguaje que nos permite indicar que determinado código se repetirá en su ejecución. Existen 3 formas de hacerlo que pasamos a ver a continuación:

- **LOOP**
- **WHILE**
- **FOR**

Estructura LOOP

```

LOOP
    Sentencias;
END LOOP;

```

Se ejecutará “infinitamente” hasta que se ejecute la orden:

```
EXIT [WHEN condición];
```

Son equivalentes:

- a. **EXIT WHEN** condición;
- b. IF condición **THEN EXIT;** END IF;

Ejemplo:

```

DECLARE
    i BINARY_INTEGER := 1;
BEGIN
    LOOP
        INSERT INTO Tabla_Temp VALUES (i*10);
        EXIT WHEN i>=10;
        i:= i+1;
    END LOOP;
END;
/

```

Este código inserta 10 filas en la tabla Tabla_Temp con valores del 10 al 100.

Estructura WHILE

```
WHILE condición LOOP
    Sentencias;
END LOOP;
```

Se ejecuta cero o más veces mientras la condición sea cierta. Puede usarse también la orden EXIT.

Ejemplo:

```
DECLARE
    i BINARY_INTEGER := 1;
BEGIN
    WHILE i<=10 LOOP
        INSERT INTO Tabla_Temp VALUES (i*10);
        i:= i+1;
    END LOOP;
END;
/
```

Este código inserta 10 filas en la tabla Tabla_Temp con valores del 10 al 100.

Estructura FOR

```
FOR i IN [REVERSE] min..max LOOP
    Sentencias;
END LOOP;
```

Nota: i es una variable que se declara automáticamente de tipo BINARY_INTEGER. No hay que declararla. Si se declara una variable de igual nombre ambas serán variables distintas (como en bloques anidados).

- La variable i tomará valores desde min hasta max,
- incrementándose automáticamente en una unidad.
- Con REVERSE los valores se toman en orden inverso, desde max hasta min.
- min y max pueden ser constantes, variables o expresiones.

Ejemplo:

```
BEGIN
    FOR i IN 1..10 LOOP
        INSERT INTO Tabla_Temp VALUES (i*10);
    END LOOP;
END;
/
```

Este código inserta 10 filas en la tabla Tabla_Temp con valores 10, 20, 30, ... , 100.

5.4.5 Variables

Una variable es el nombre que se da a una zona de memoria donde se guardarán ciertos datos. PL/SQL soporta todos los tipos de SQL más algunos más que veremos a continuación. Formato: Las variables PL/SQL se declaran con el siguiente formato:

Nombre Tipo [[CONSTANT NOT NULL] := Valor_Inicial];

Conversiones de Tipo:

- Explícita: Usando funciones de conversión: TO_CHAR, TO_NUMBER...
- Implícita: PL/SQL realiza conversiones automáticas si es posible, incluso entre datos de distintas familias (numéricos, caracteres, fechas...). Esto es desaconsejado para evitar problemas.

Tipos de datos:

Puede ser cualquier tipo válido en una columna de tabla (vistos anteriormente) y otros tipos adicionales. El valor por defecto es NULL, excepto que se especifique un valor como Valor_Inicial (puede sustituirse := por DEFAULT). Con NOT NULL se requiere la inicialización.

- Tipos Escalares:
 - Numéricos Reales.
 - NUMBER(p,e) y sus subtipos totalmente equivalentes definidos por cuestiones de compatibilidad: DEC, DECIMAL, DOUBLE PRECISION, INT, INTEGER, NUMERIC, SMALLINT y REAL. Se almacenan en formato decimal: Para operaciones aritméticas deben traducirse a binario.
 - Numéricos Enteros.
 - BINARY_INTEGER, que es un entero en binario (complemento a 2) con rango ± 2147483647 , ideal para variables sobre las que se efectuarán operaciones (contadores...). Tiene definidos subtipos restringidos en su rango: NATURAL [0, 2147483647], NATURALN (igual que NATURAL pero NOT NULL), POSITIVE [1, 2147483647], POSITIVEN, SIGNTYPE (-1, 0 y 1).
 - PLS_INTEGER es similar a BINARY_INTEGER, pero más rápido en las operaciones aritméticas y que genera un error si se produce un desbordamiento (ORA-1426) al asignarlo a un NUMBER.
- Carácter:
 - VARCHAR2(max_tam), con max_tam ≤ 32766 bytes (como columna de tabla admite 4000 $\rightarrow$ Cuidado con los errores). Si se usa un código distinto al código ASCII, el número total de caracteres puede ser menor.
 - CHAR (tam_fijo) con 1 por defecto y 32767 como máximo (como columna de tabla admite 255), se rellena siempre con blancos. LONG es una cadena de longitud variable con un máximo de 32760 (como columna de tabla admite 2GB).
 - NCHAR y NVARCHAR2 permiten almacenar cadenas en un conjunto de caracteres nacional distinto al propio de PL/SQL.
- Binarios:
 - RAW(max_tam), con max_tam ≤ 32766 bytes.
 - LONG RAW admite un máximo de 32760 bytes.
 - ROWID es un tipo para almacenar identificadores de fila (pseudocolumna ROWID).
- Fecha:
 - DATE, como en SQL de Oracle almacena siglo, año, mes, día, hora, min. y segs. (7 bytes).
- Lógico:

- BOOLEAN, con los siguientes valores posibles: TRUE, FALSE y NULL.
- Tipos Referencias (punteros):
 - REF CURSOR y REF TipoObjeto.
- Tipos LOB (Large Object):
 - BFILE, LOB, CLOB y NLOB (se usan con el paquete DBMS_LOB) .
- Tipos Compuestos:
 - RECORD, TABLE y VARRAY. El tipo RECORD o REGISTRO lo veremos en detalle en el siguiente apartado.
- Notaciones especiales:
 - Tabla.Columna %TYPE: el tipo que tenga asignado una columna de una tabla, independientemente de cómo esté definida ésta.
 - Tabla %ROWTYPE: el tipo que tenga asignado una fila de una tabla, independientemente de cómo esté definida ésta.

Ejemplos:

```
CODIGO      HOTEL.ID%TYPE;
HABS        HOTEL.NHABS%TYPE;
DEP         DEPARTAMENTOS%ROWTYPE;
HOTEL       HOTEL%ROWTYPE;
```

Esto hace los programas más robustos frente a cambios de tipo. Ejemplo completo: Supongamos que disponemos de una tabla llamada Hotel con el siguiente diseño físico:

```
CREATE TABLE HOTEL (ID NUMBER(2) PRIMARY KEY, NHABS NUMBER(3) );
```

Si insertamos los siguientes valores:

```
INSERT INTO HOTEL VALUES (1, 10);
INSERT INTO HOTEL VALUES (2, 60);
INSERT INTO HOTEL VALUES (3, 200);
INSERT INTO HOTEL VALUES (99, NULL);
```

Y definimos el siguiente procedimiento:

```
CREATE OR REPLACE
PROCEDURE TAMHOTEL (cod Hotel.ID%TYPE)
AS
  NumHabitaciones Hotel.Nhabs%TYPE;
  Comentario       VARCHAR2(40);
BEGIN
  -- Número de habitaciones del Hotel cuyo ID es cod
  SELECT Nhabs INTO NumHabitaciones
  FROM Hotel WHERE ID=cod;

  IF NumHabitaciones IS NULL THEN
    Comentario := 'de tamaño indeterminado';
  ELSIF NumHabitaciones < 50 THEN
    Comentario := 'Pequeño';
  ELSIF NumHabitaciones < 100 THEN
    Comentario := 'Mediano';
  ELSE
```

(continué en la próxima página)

(proviene de la página anterior)

```
Comentario := 'Grande';
END IF;

DBMS_OUTPUT.PUT_LINE ('El hotel con ID '|| cod ||' es '|| Comentario);
END;
/
```

Si ejecutamos:

```
BEGIN
    TAMHOTEL(1);
    TAMHOTEL(2);
    TAMHOTEL(3);
    TAMHOTEL(99);
END;
/
```

Los resultados serán:

```
El hotel con ID 1 es Pequeño
El hotel con ID 2 es Mediano
El hotel con ID 3 es Grande
El hotel con ID 99 es de tamaño indeterminado
```

5.4.6 Registros

Son agrupaciones de datos relacionados. Permite crear estructuras que albergan un conjunto de tipos de datos. Por ejemplo, podemos crear el registro PERSONA con los campos código, nombre y edad, cada uno de estos campos con diferentes tipos de datos. En PL/SQL su importancia proviene de su similitud a la fila (registro) de tabla. Es necesario definir un Tipo de Dato Registro, para declarar variables.

Formato:

```
TYPE Tipo_Registro IS RECORD
(
    Campo1 Tipo1 [[NOT NULL] :=Expr1],
    Campo2 Tipo2 [[NOT NULL] :=Expr2],
    . . .
    CampoN TipoN [[NOT NULL] :=ExprN]
);
```

Igual que en los lenguajes de programación, para acceder a un campo se usa la Notación Punto: VariableRegistro.Campo. Se permite asignar registros si son del mismo tipo. Si son de tipos distintos no se pueden asignar, aunque estén definidos igual. En ese caso, se pueden asignar campo a campo.

También se pueden asignar los campos de un SELECT en un registro compatible.

Tabla %ROWTYPE

Esta especificación sirve para dar a una variable el tipo que tenga asignado una fila de una tabla, independientemente de cómo esté definida ésta. Esto hace los programas más robustos frente a cambios de tipo.

Ejemplo 1

Podemos crear un registro indicando el tipo de datos de cada campo.

```

DECLARE
  TYPE RegPersona IS RECORD
  (
    CODIGO NUMBER(2),
    NOMBRE VARCHAR2(40),
    EDAD    NUMBER
  );
  Pepe RegPersona;
BEGIN
  Pepe.CODIGO := 1;
  Pepe.NOMBRE := 'Pepe';
  Pepe.EDAD   := 30;
  DBMS_OUTPUT.PUT_LINE ('Código: ' || Pepe.CODIGO);
  DBMS_OUTPUT.PUT_LINE ('Nombre: ' || Pepe.NOMBRE);
  DBMS_OUTPUT.PUT_LINE ('Edad   : ' || Pepe.EDAD);
  INSERT INTO PERSONAS VALUES Pepe;
END;
/

```

Nota: Para no tener problemas con la sentencia INSERT deberemos tener creada previamente una tabla cuyas filas (o registros) sean idénticas al registro Pepe.

Por ejemplo:

```

CREATE TABLE PERSONAS
(
  CODIGO  NUMBER(2),
  NOMBRE  VARCHAR2(40),
  EDAD    NUMBER
);

```

Ejemplo 2 (Usando atributo %TYPE)

Podemos crear un registro indicando que el tipo de datos de cada campo coincide con campos de una tabla.

```

DECLARE
  TYPE RegHotel IS RECORD
  (
    COD  Hotel.ID%TYPE,
    HABS Hotel.NHABS%TYPE
  );
  Hotel99 RegHotel;
BEGIN
  SELECT ID, NHABS INTO Hotel99 FROM HOTEL WHERE ID=99;
  DBMS_OUTPUT.PUT_LINE ('Cód. Hotel : ' || Hotel99.COD);
  DBMS_OUTPUT.PUT_LINE ('Habitaciones: ' || Hotel99.HABS);
END;
/

```

Ejemplo 3 (Usando atributo %ROWTYPE)

Podemos indicar que una variable registro coincide con el formato de fila de una tabla. El ejemplo anterior puede simplificarse así:

```
DECLARE
    Hotel99 Hotel%ROWTYPE;
BEGIN
    SELECT * INTO Hotel99 FROM HOTEL WHERE ID=99;
    DBMS_OUTPUT.PUT_LINE ('Cód. Hotel : ' || Hotel99.ID);
    DBMS_OUTPUT.PUT_LINE ('Habitaciones: ' || Hotel99.NHABS);
END;
/
```

5.4.7 Cursores

PL/SQL utiliza cursores para gestionar las instrucciones SELECT. Un cursor es un conjunto de registros devuelto por una instrucción SQL. Técnicamente los cursores son fragmentos de memoria reservados para procesar los resultados de una consulta SELECT.

Hay dos tipos de cursores: **Implícitos** y **Explícitos**.

Un cursor se define como cualquier otra variable de PL/SQL y debe nombrarse de acuerdo a los mismos convenios que cualquier otra variable. Los cursores implícitos no necesitan declaración como tales. Los cursores explícitos debemos declararlos con la palabra CURSOR.

- **IMPLÍCITOS**: Este tipo de cursores se utiliza para operaciones SELECT INTO. Se usan cuando la consulta devuelve un único registro y no es necesario declararlos como tales.

Ejemplo:

```
DECLARE
    Hotel99 Hotel%ROWTYPE;
BEGIN
    SELECT * INTO Hotel99      -- Hotel99 es un cursor implícito
    FROM HOTEL WHERE ID=99;    -- Almacena un solo registro

    DBMS_OUTPUT.PUT_LINE ('Cód. Hotel : ' || Hotel99.ID);
    DBMS_OUTPUT.PUT_LINE ('Habitaciones: ' || Hotel99.NHABS);
END;
/
```

- **EXPLÍCITOS**: Se utilizan cuando la consulta devuelve un conjunto de registros. Ocasionalmente también se utilizan en consultas que devuelven un único registro por razones de eficiencia. Son más rápidos.

Ejemplo:

```
DECLARE
    CURSOR Hoteles IS          -- Hoteles es un cursor explícito
    SELECT * FROM Hotel;       -- Almacena varios registros
BEGIN
    FOR registro IN Hoteles    -- No es necesario declarar registro
    LOOP
        DBMS_OUTPUT.PUT_LINE ('Cód. Hotel : ' || registro.ID);
        DBMS_OUTPUT.PUT_LINE ('Habitaciones: ' || registro.NHABS);
        DBMS_OUTPUT.PUT_LINE ('      ----      ');
    END LOOP;
END;
/
```

Nota: Existe otra forma de trabajar con cursores explícitos, pero resulta más complicada, por lo que se recomienda

utilizar, siempre que se pueda, la forma indicada anteriormente.

La forma complicada de hacerlo sería así:

```
DECLARE
  CURSOR Hoteles IS          -- Hoteles es un cursor explícito
    SELECT * FROM Hotel;      -- Almacena varios registros
    registro Hoteles%ROWTYPE; -- Es necesario declarar esta variable
BEGIN
  OPEN Hoteles;              -- Abrimos cursor
  LOOP
    FETCH Hoteles INTO registro; -- Recuperamos un registro
    EXIT WHEN Hoteles%NOTFOUND;  -- Salimos si no hay más registros
    DBMS_OUTPUT.PUT_LINE ('Cód. Hotel : ' || registro.ID);
    DBMS_OUTPUT.PUT_LINE ('Habitaciones: ' || registro.NHABS);
    DBMS_OUTPUT.PUT_LINE ('      ---- ');
  END LOOP;
  CLOSE Hoteles;             -- Cerramos cursor
END;
/
```

Ejemplo de cursor explícito con parámetros:

Un cursor puede aceptar parámetros. Por ejemplo cuando deseamos que los resultados de la consulta dependan de ciertas variables. Para hacer que el cursor varíe según esos parámetros, se han de indicar los mismos en la declaración del cursor. Para ello se pone entre paréntesis su nombre y tipo tras el nombre del cursor en la declaración. A continuación se muestra un cursor que admite un parámetro que luego será utilizado en la consulta para mostrar los hoteles cuyo ID sea menor que el valor de dicho parámetro.

```
DECLARE
  CURSOR Hoteles(num NUMBER) IS          -- Hoteles es un cursor explícito
    SELECT * FROM Hotel WHERE ID<num;    -- Almacena varios registros
BEGIN
  FOR registro IN Hoteles(10)            -- Hoteles con ID < 10
  LOOP
    DBMS_OUTPUT.PUT_LINE ('Cód. Hotel : ' || registro.ID);
    DBMS_OUTPUT.PUT_LINE ('Habitaciones: ' || registro.NHABS);
    DBMS_OUTPUT.PUT_LINE ('      ---- ');
  END LOOP;
END;
/
```

Atributos de los cursores

Cada cursor, tanto implícito como explícito, tiene una serie de atributos, cada uno de los cuales devuelve información útil sobre la ejecución de una instrucción de manipulación de datos. Los 4 atributos más frecuentes que podemos encontrar son:

- **%ISOPEN**: Devuelve TRUE si un cursor está abierto. Para cursores implícitos siempre devuelve FALSE, porque Oracle cierra el cursor SQL automáticamente después de ejecutar su sentencia SQL asociada.
- **%FOUND**: Devuelve TRUE si una instrucción INSERT, UPDATE o DELETE afectó a una o más filas o una instrucción SELECT INTO devolvió una o más filas. De lo contrario, devuelve FALSE.
- **%NOTFOUND**: Devuelve TRUE si una instrucción INSERT, UPDATE o DELETE no afectó a ninguna fila o una instrucción SELECT INTO no devolvió ninguna fila. De lo contrario, devuelve FALSE.

- **%ROWCOUNT**: Devuelve el número de filas afectadas por una sentencia INSERT, UPDATE o DELETE o devuelta por una sentencia SELECT INTO.

Nota:

- Cuando se usa con cursores implícitos se antepone la palabra SQL.
 - Cuando se usa con cursores explícitos se antepone el nombre del cursor.
-

Ejemplos para cursores implícitos:

```
UPDATE empleados SET salar = salar * 1.05 WHERE numem = 120;
IF SQL%NOTFOUND THEN
    INSERT INTO empleados VALUES (120, 'José', ...);
END IF;
```

En el siguiente ejemplo, usamos %ROWCOUNT para lanzar una excepción si se borran más de 5 filas:

```
DELETE empleados WHERE numhi > 0;
IF SQL%ROWCOUNT > 5 THEN -- más de 5 filas borradas
    RAISE borrado_masivo;
END IF;
```

Ejemplos para cursores explícitos:

```
DECLARE
    CURSOR c IS
        -- ordenamos por empleado con mayor salario
        SELECT nomem, numem, salar FROM empleados ORDER BY salar DESC;

    nombre CHAR(10);
    numero NUMBER(3);
    salario NUMBER(7,2);
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO nombre, numero, salario;
        -- mostramos un máximo de 5 empleados
        EXIT WHEN (c%ROWCOUNT > 5) OR (c%NOTFOUND);
        DBMS_OUTPUT.PUT_LINE (c%ROWCOUNT
            || ' - ' || salario || ' - ' || numero || ' - ' || nombre);
    END LOOP;
    CLOSE c;
END;
```

5.4.8 Gestión de excepciones

Se llama excepción a todo hecho que le sucede a un programa que causa que la ejecución del mismo finalice. Lógicamente eso causa que el programa termine de forma anormal. Las excepciones se deben a:

Que ocurra un error detectado por Oracle (por ejemplo si un SELECT no devuelve datos ocurre el error ORA-01403 llamado NO_DATA_FOUND). Que el propio programador las lance (comando RAISE). Las excepciones se pueden capturar a fin de que el programa controle mejor la existencia de las mismas.

Captura de excepciones

La captura se realiza utilizando el bloque `EXCEPTION` que es el bloque que está justo antes del `END` del bloque. Cuando una excepción ocurre, se comprueba el bloque `EXCEPTION` para ver si ha sido capturada, si no se captura, el error se propaga a Oracle que se encargará de indicar el error existente.

Las excepciones pueden ser de estos tipos:

- Excepciones predefinidas de Oracle. Que tienen ya asignado un nombre de excepción.
- Excepciones de Oracle sin definir. No tienen nombre asignado pero se les puede asignar.
- Definidas por el usuario. Las lanza el programador.

La captura de excepciones se realiza con esta sintaxis:

```
DECLARE
    sección de declaraciones
BEGIN
    instrucciones
EXCEPTION
    WHEN excepción1 [OR excepción2 ...] THEN
        instrucciones que se ejecutan si suceden esas excepciones
    [WHEN excepción3 [OR...] THEN
        instrucciones que se ejecutan si suceden esas excepciones]
    [WHEN OTHERS THEN
        instrucciones que se ejecutan si suceden otras excepciones]
END;
```

Cuando ocurre una determinada excepción, se comprueba el primer `WHEN` para comprobar si el nombre de la excepción ocurrida coincide con el que dicho `WHEN` captura; si es así se ejecutan las instrucciones, si no es así se comprueba el siguiente `WHEN` y así sucesivamente.

Si existen cláusula `WHEN OTHERS`, entonces las excepciones que no estaban reflejadas en los demás apartados `WHEN` ejecutan las instrucciones del `WHEN OTHERS`. Ésta cláusula debe ser la última.

Excepciones predefinidas

Oracle tiene muchas excepciones predefinidas. Son errores a los que Oracle asigna un nombre de excepción. Algunas de las que aparecen con mayor frecuencia son:

Nombre de excepción	Número	Ocorre cuando...
CA-SE_NOT_FOUND	ORA-06592	Ninguna opción <code>WHEN</code> dentro de la instrucción <code>CASE</code> captura el valor, y no hay instrucción <code>ELSE</code>
DUP_VAL_ON_INDEX	ORA-00001	Se intentó añadir una fila que provoca que un índice único repita valores
INVALID_NUMBER	ORA-01722	Falla la conversión de carácter a número
NO_DATA_FOUND	ORA-01403	El <code>SELECT</code> de fila única no devolvió valores
TOO_MANY_ROWS	ORA-01422	El <code>SELECT</code> de fila única devuelve más de una fila
VALUE_ERROR	ORA-06502	Hay un error aritmético, de conversión, de redondeo o de tamaño en una operación
ZERO_DIVIDE	ORA-01476	Se intenta dividir entre el número cero.

Ejemplos:

En el siguiente ejemplo se producirá una excepción ZERO_DIVIDE puesto que el divisor x es igual a 0.

```
DECLARE
  x NUMBER := 0;
  y NUMBER := 3;
  res NUMBER;
BEGIN
  res:=y/x;
  DBMS_OUTPUT.PUT_LINE(res);

  EXCEPTION
    WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE('No se puede dividir por cero') ;

    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error inesperado') ;
END;
/
```

En el siguiente ejemplo el cursor implícito Hotel99 sólo puede recibir una única fila o registro como resultado de una consulta. En este caso podrían producirse 2 excepciones: NO_DATA_FOUND (la consulta select no devuelve ningún registro) o TOO_MANY_ROWS (la consulta select devuelve más de un registro). En el primer caso insertamos un nuevo registro. En el segundo caso borramos el registro duplicado.

```
DECLARE
  Hotel99 Hotel%ROWTYPE;
BEGIN
  SELECT * INTO Hotel99 WHERE Nombre='Costanera';
  -- IF SQL%NOTFOUND THEN ... // Esto no tiene sentido aquí
  -- IF SQL%ROWCOUNT > 1 THEN ... // Tampoco tiene sentido aquí

  EXCEPTION
    WHEN NO_DATA_FOUND THEN -- Cuando no se recupera ninguna fila
      INSERT INTO Hotel VALUES (99, 'Costanera', 110, 60, 'S', 3 );

    WHEN TOO_MANY_ROWS THEN -- Cuando se recuperan varias filas
      DELETE Hotel WHERE Nombre='Costanera' AND HotelID<>99;
END;
```

Si una instrucción SELECT INTO no devuelve una fila, PL/SQL lanza la excepción predefinida NO_DATA_FOUND tanto si se comprueba SQL %NOTFOUND en la línea siguiente como si no. Si una instrucción SELECT INTO devuelve más de una fila, PL/SQL lanza la excepción predefinida TOO_MANY_ROWS tanto si se comprueba SQL %ROWCOUNT en la línea siguiente como si no.

Funciones de uso con excepciones

Se suelen usar dos funciones cuando se trabaja con excepciones:

- **SQLCODE**. Retorna el código de error del error ocurrido
- **SQLERRM**. Devuelve el mensaje de error de Oracle asociado a ese número de error.

Ejemplo:

```
EXCEPTION
...
```

(continué en la próxima página)

(proviene de la página anterior)

```

WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE
        ('Ocurrió el error ' || SQLCODE || 'mensaje: ' || SQLERRM);
END;

```

Excepciones de usuario

El programador puede lanzar sus propias excepciones simulando errores del programa. Para ello hay que:

- 1) Declarar un nombre para la excepción en el apartado DECLARE, al igual que para las excepciones sin definir:

```
miExcepcion EXCEPTION;
```

- 2) En la sección ejecutable (BEGIN ... END) utilizar la instrucción RAISE para lanzar la excepción:

```
RAISE miExcepcion;
```

- 3) En el apartado de excepciones capturar el nombre de excepción declarado:

```

EXCEPTION
...
WHEN miExcepcion THEN
...

```

Ejemplo:

```

DECLARE
    error_al_eliminar EXCEPTION;
BEGIN
    DELETE piezas WHERE tipo='ZU' AND modelo=26;
    IF SQL%NOTFOUND THEN
        RAISE error_al_eliminar;
    END IF;
EXCEPTION
    WHEN error_al_eliminar THEN
        DBMS_OUTPUT.PUT_LINE ('Error -20001: No existe esa pieza');
END;
/

```

Otra forma es utilizar la función RAISE_APPLICATION_ERROR que simplifica los tres pasos anteriores. Sintaxis:

```
RAISE_APPLICATION_ERROR (noDeError, mensaje, [, {TRUE|FALSE}]);
```

Esta instrucción se coloca en la sección ejecutable o en la de excepciones y sustituye a los tres pasos anteriores. Lo que hace es lanzar un error cuyo número debe de estar entre el -20000 y el -20999 y hace que Oracle muestre el mensaje indicado. El tercer parámetro opciones puede ser TRUE o FALSE (por defecto TRUE) e indica si el error se añade a la pila de errores existentes.

Ejemplo con RAISE_APPLICATION_ERROR:

```

DECLARE
BEGIN
    DELETE piezas WHERE tipo='ZU' AND modelo=26;
    IF SQL%NOTFOUND THEN
        RAISE_APPLICATION_ERROR(-20001, 'No existe esa pieza');

```

(continué en la próxima página)

(proviene de la página anterior)

```
END IF;
END;
/
```

En el ejemplo, si la pieza no existe, entonces SQL %NOTFOUND devuelve verdadero ya que el DELETE no elimina ninguna pieza. Se lanza la excepción de usuario -20001 haciendo que Oracle utilice el mensaje indicado. Oracle lanzará el mensaje: ORA-20001: No existe esa pieza.

5.4.9 Procedimientos

Un procedimiento es un bloque que puede recibir parámetros, lo cual permite trabajar con unos datos de entrada, realizar las operaciones deseadas con dichos datos y, en algunos casos guardar ciertos resultados como parámetros de salida. Se usa la palabra reservada PROCEDURE. Su estructura simplificada es:

```
PROCEDURE nombre IS
bloque sin palabra DECLARE
```

Su estructura en detalle es:

```
PROCEDURE nombre
[(parámetro1 [modo] tipoDatos[, parámetro2 [modo] tipoDatos [, ...]])]
{IS|AS}
bloque sin palabra DECLARE
```

Los procedimientos permiten utilizar parámetros para realizar su tarea. El modo, que es opcional, puede ser de 3 tipos: **IN**, **OUT** o **IN OUT**. Si no se indica nada, por defecto es **IN**.

- **Parámetros IN.** Son los parámetros que en otros lenguajes se denominan como parámetros por valor. El procedimiento recibe una copia del valor o variable que se utiliza como parámetro al llamar al procedimiento. Estos parámetros pueden ser: valores literales (18 por ejemplo), variables (v_num por ejemplo) o expresiones (como v_num+18). A estos parámetros se les puede asignar un valor por defecto.
- **Parámetros OUT.** Relacionados con el paso por variable de otros lenguajes. Sólo pueden ser variables y no pueden tener un valor por defecto. Se utilizan para que el procedimiento almacene en ellas algún valor. Es decir, los parámetros OUT son variables sin declarar que se envían al procedimiento de modo que si en el procedimiento cambian su valor, ese valor permanece en ellas cuando el procedimiento termina.
- **Parámetros IN OUT.** Son una mezcla de los dos anteriores. Se trata de variables declaradas anteriormente cuyo valor puede ser utilizado por el procedimiento que, además, puede almacenar un valor en ellas. No se las puede asignar un valor por defecto.

Para crear el procedimiento debemos anteponer la sentencia

```
CREATE [ OR REPLACE ]
```

La opción REPLACE hace que si ya existe un procedimiento con ese nombre, se reemplaza con el que se crea ahora. Los parámetros son la lista de variables que necesita el procedimiento para realizar su tarea. Para invocar al procedimiento o procedimientos definidos debemos hacerlo dentro de un bloque BEGIN ... END; o también con la sentencia EXEC si lo ejecutamos desde SQL*Plus.

```
BEGIN
  procedimiento1;
  procedimiento2();
  procedimiento3(parametro1, parametro2);
  ...
```

(continúe en la próxima página)

(proviene de la página anterior)

```
END;
/
```

o también en SQL*Plus:

```
EXEC procedimiento1;
EXEC procedimiento2();
EXEC procedimiento3(parametro1, parametro2);
```

Cuando se invoca a un procedimiento, si éste no tiene parámetros, se pueden omitir los paréntesis (es decir la llamada al procedimiento procedimiento2() se puede hacer simplemente escribiendo procedimiento2, sin paréntesis)

Para eliminar un procedimiento utilizamos la sentencia DROP PROCEDURE.

```
DROP PROCEDURE procedimiento;
```

Ejemplo:

```
CREATE OR REPLACE
PROCEDURE muestra_fecha IS
    fecha DATE;
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Salida de información');
    SELECT SYSDATE INTO fecha FROM DUAL;
    DBMS_OUTPUT.PUT_LINE ('Fecha: ' || fecha);
END muestra_fecha;
/
```

Para crear el procedimiento muestra_fecha sin parámetros.

Para invocar el procedimiento muestra_fecha:

```
BEGIN
    muestra_fecha;
END;
/
```

o también en SQL*Plus:

```
EXEC muestra_fecha;
```

Ejemplo de procedimiento con parámetros:

```
CREATE OR REPLACE
PROCEDURE escribe (texto VARCHAR2)
IS
BEGIN
    DBMS_OUTPUT.PUT_LINE(texto);
END;
/

BEGIN
    ESCRIBE('HOLA');
END;
/
```

o también en SQL*Plus:

```
EXEC ESCRIBE('HOLA');
```

Al declarar cada parámetro se indica el tipo de los mismos, pero no su tamaño; es decir sería VARCHAR2 y no VARCHAR2(50).

5.4.10 Funciones

Una función es prácticamente idéntica a un procedimiento. También puede recibir parámetros de entrada y realizar operaciones con dichos datos. Lo que distingue a una función de un procedimiento es que la función siempre devuelve algún valor. Se usa la palabra reservada FUNCTION. Su estructura simplificada es:

```
FUNCTION nombre RETURN tipoDeDatos IS  
bloque sin palabra DECLARE
```

Su estructura en detalle es:

```
FUNCTION nombre  
[(parámetro1 [modelo] tipoDatos  
[,parámetro2 [modelo] tipoDatos [...]])]  
RETURN tipoDeDatos  
{IS|AS}  
bloque sin palabra DECLARE
```

Para crear la función debemos anteponer la sentencia

```
CREATE [ OR REPLACE ]
```

La opción REPLACE hace que si ya existe una función con ese nombre, se reemplaza con la que se crea ahora. Los parámetros son la lista de variables que necesita la función para realizar su tarea. Para invocar la función debemos hacerlo dentro de una expresión. Ejemplo:

```
SELECT ...función... FROM DUAL;
```

Para eliminar una función utilizamos la sentencia DROP FUNCTION.

```
DROP FUNCTION función;
```

Ejemplo:

```
CREATE OR REPLACE  
FUNCTION SUMA (NUM1 NUMBER, NUM2 NUMBER)  
RETURN NUMBER  
IS  
BEGIN  
    RETURN NUM1+NUM2;  
END SUMA;  
/
```

Para invocar la función definida debemos hacerlo dentro de una expresión. Ejemplos:

```
SELECT SUMA(5.7, 9.3) FROM DUAL;  
SELECT SUMA(5.7, 9.3)*3 FROM DUAL;  
SELECT 150/(SUMA(5.7, 9.3)*3) FROM DUAL;  
SELECT SYSDATE+SUMA(10,2)-2 FROM DUAL;
```

5.4.11 Paquetes

Los paquetes sirven para agrupar bajo un mismo nombre funciones y procedimientos. Facilitan la modularización de programas y su mantenimiento. Los paquetes constan de dos partes:

- **Especificación.** Que sirve para declarar los elementos de los que consta el paquete. En esta especificación se indican los procedimientos, funciones y variables públicos del paquete (los que se podrán invocar desde fuera del paquete). De los procedimientos sólo se indica su nombre y parámetros (sin el cuerpo).
- **Cuerpo.** En la que se especifica el funcionamiento del paquete. Consta de la definición de los procedimientos indicados en la especificación. Además se pueden declarar y definir variables y procedimientos privados (sólo visibles para el cuerpo del paquete, no se pueden invocar desde fuera del mismo).

```
-- PAQUETE ARITMETICA - Especificación
-- PACKAGE_ARITMETICA.SQL
CREATE OR REPLACE
PACKAGE aritmetica IS
    version NUMBER := 1.0;

    PROCEDURE mostrar_info;
    FUNCTION suma      (a NUMBER, b NUMBER) RETURN NUMBER;
    FUNCTION resta     (a NUMBER, b NUMBER) RETURN NUMBER;
    FUNCTION multiplica (a NUMBER, b NUMBER) RETURN NUMBER;
    FUNCTION divide    (a NUMBER, b NUMBER) RETURN NUMBER;
END aritmetica;
/

-- PAQUETE ARITMETICA - Cuerpo
-- PACKAGE_BODY_ARITMETICA.SQL
CREATE OR REPLACE
PACKAGE BODY aritmetica IS

    PROCEDURE mostrar_info IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE
            ('Paquete de operaciones aritméticas. Versión ' || version);
    END mostrar_info;

    FUNCTION suma      (a NUMBER, b NUMBER) RETURN NUMBER IS
    BEGIN
        RETURN (a+b);
    END suma;

    FUNCTION resta     (a NUMBER, b NUMBER) RETURN NUMBER IS
    BEGIN
        RETURN (a-b);
    END resta;

    FUNCTION multiplica (a NUMBER, b NUMBER) RETURN NUMBER IS
    BEGIN
        RETURN (a*b);
    END multiplica;

    FUNCTION divide    (a NUMBER, b NUMBER) RETURN NUMBER IS
    BEGIN
        RETURN (a/b);
    END divide;
```

(continué en la próxima página)

(proviene de la página anterior)

```
END aritmetica;  
/
```

Para utilizar el paquete debemos llamar al procedimiento y funciones deseadas.

Ejemplo de uso, sencillo:

```
BEGIN  
    ARITMETICA.MOSTRAR_INFO;  
END;  
/  
  
SELECT ARITMETICA.SUMA(4,3) FROM DUAL;  
SELECT ARITMETICA.RESTA(4,3) FROM DUAL;  
SELECT ARITMETICA.MULTIPLICA(4,3) FROM DUAL;  
SELECT ARITMETICA.DIVIDE(4,3) FROM DUAL;
```

Ejemplo de uso, más elaborado:

```
DECLARE  
    num1      NUMBER:= 2;  
    num2      NUMBER:= 5;  
    resultado NUMBER;  
BEGIN  
    ARITMETICA.MOSTRAR_INFO;  
  
    SELECT ARITMETICA.SUMA(num1,num2) INTO resultado FROM DUAL;  
    DBMS_OUTPUT.PUT_LINE  
        ('La suma de ' || num1 || ' y ' || num2 || ' es ' || resultado);  
  
    SELECT ARITMETICA.RESTA(num1,num2) INTO resultado FROM DUAL;  
    DBMS_OUTPUT.PUT_LINE  
        ('La resta de ' || num1 || ' y ' || num2 || ' es ' || resultado);  
END;  
/
```

Oracle incorpora una serie de paquetes para ser utilizados dentro del código PL/SQL. Es el caso del paquete DBMS_OUTPUT que sirve para utilizar funciones y procedimientos de escritura como PUT_LINE. Otro ejemplo es el paquete DBMS_RANDOM, que contiene diversas funciones para utilizar número aleatorios. Quizá la más útil es la función DBMS_RANDOM.RANDOM que devuelve un número entero (positivo o negativo) aleatorio (y muy grande). Ejemplos:

```
-- Si deseamos un número aleatorio entre 1 y 10  
MOD (ABS (DBMS_RANDOM.RANDOM) , 10) + 1  
  
-- Entre 20 y 50  
MOD (ABS (DBMS_RANDOM.RANDOM) , 31) + 20
```

5.4.12 Disparadores (Triggers)

Es un bloque PL/SQL que se ejecuta de forma implícita cuando se ejecuta cierta operación DML: INSERT, DELETE o UPDATE. Contrariamente, los procedimientos y las funciones se ejecutan haciendo una llamada explícita a ellos. Un disparador NO admite argumentos.

Sus aplicaciones son inmensas, como por ejemplo:

- Mantenimiento de Restricciones de Integridad complejas. Ej: Restricciones de Estado (como que el sueldo sólo puede aumentar).
- Auditoría de una Tabla, registrando los cambios efectuados y la identidad del que los llevó a cabo.
- Lanzar cualquier acción cuando una tabla es modificada.

Su estructura general es:

```
CREATE [OR REPLACE]
TRIGGER Nombre
{ BEFORE | AFTER } Suceso_Disparo ON Tabla
[ FOR EACH ROW [ WHEN Condición_Disparo ] ]
Bloque_del_TRIGGER;
```

Para borrar un disparador:

```
DROP TRIGGER Nombre ;
```

Para habilitar/deshabilitar un disparador:

```
ALTER TRIGGER Nombre { ENABLE | DISABLE };
```

Para desactivar o activar todos los triggers de una tabla:

```
ALTER TABLE nombreTabla { DISABLE | ENABLE } ALL TRIGGERS;
```

Eso permite en una sola instrucción operar con todos los triggers relacionados con una determinada tabla (es decir actúa sobre los triggers que tienen dicha tabla en el apartado ON del trigger).

Tipos de disparadores

Tenemos tres tipos de triggers:

- **Triggers de tabla.** Se trata de triggers que se disparan cuando ocurre una acción DML sobre una tabla.
- **Triggers de vista.** Se lanzan cuando ocurre una acción DML sobre una vista.
- **Triggers de sistema.** Se disparan cuando se produce un evento sobre la base de datos (conexión de un usuario, borrado de un objeto,...)

Aquí sólo veremos los del primer y segundo tipo. Por lo que se dará por hecho en todo momento que nos referiremos siempre a ese tipo de triggers.

Los triggers se utilizan para:

- Ejecutar acciones relacionadas con la que dispara el trigger
- Centralizar operaciones globales
- Realizar tareas administrativas de forma automática
- Evitar errores
- Crear reglas de integridad complejas

El código que se lanza con el trigger es PL/SQL. No es conveniente realizar excesivos triggers, sólo los necesarios, de otro modo se ralentiza en exceso la base de datos.

Elementos de los triggers

Puesto que un trigger es un código que se dispara, al crearle se deben indicar las siguientes cosas:

- 1) El evento que da lugar a la ejecución del trigger:

```
INSERT
UPDATE
DELETE
```

- 2) Cuándo se lanza el evento en relación a dicho evento:

```
BEFORE
AFTER
INSTEAD OF
```

- **BEFORE:** El código del trigger se ejecuta antes de ejecutar la instrucción DML que causó el lanzamiento del trigger.
- **AFTER:** El código del trigger se ejecuta después de haber ejecutado la instrucción DML que causó el lanzamiento del trigger.
- **INSTEAD OF:** El trigger sustituye a la operación DML. Se utiliza para vistas que no admiten instrucciones DML.

- 3) Las veces que el trigger se ejecuta o tipo de trigger:

- **de Instrucción.** El cuerpo del trigger se ejecuta una sola vez por cada evento que lance el trigger. Esta es la opción por defecto. El código se ejecuta aunque la instrucción DML no genere resultados.
- **de Fila.** El código se ejecuta una vez por cada fila afectada por el evento. Por ejemplo si hay una cláusula UPDATE que desencadena un trigger y dicho UPDATE actualiza 10 filas; si el trigger es de fila se ejecuta una vez por cada fila, si es de instrucción se ejecuta sólo una vez.

- 4) El cuerpo del trigger, es decir el código que ejecuta dicho trigger.

Ejemplo:

```
-- TRIGGER para realizar auditoría sobre operaciones en Empleados
CREATE OR REPLACE
TRIGGER Control_Empleados
AFTER INSERT OR DELETE OR UPDATE ON Empleados
BEGIN
    INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha)
    VALUES ('Empleados', USER, SYSDATE);
END Control_Empleados;
/

-- La información de auditoría se guardará en la siguiente tabla
CREATE TABLE CTRL_EMPLEADOS
(
    TABLA      VARCHAR2(50),
    USUARIO    VARCHAR2(50),
    FECHA      DATE
);
```

Triggers de instrucción

```
CREATE [ OR REPLACE ]
TRIGGER Nombre
{ BEFORE | AFTER } evento1 [OR evento2[, ...]] ON tabla
[ DECLARE
  declaraciones ]
BEGIN
  cuerpo
  [ EXCEPTION
    captura de excepciones ]
END;
```

El evento tiene esta sintaxis:

```
{ INSERT | UPDATE [OF columna1 [, columna2, ...]] | DELETE}
```

Los eventos asocian el trigger al uso de una instrucción DML. En el caso de la instrucción UPDATE, el apartado OF hace que el trigger se ejecute sólo cuando se modifique la columna indicada (o columnas si se utiliza una lista de columnas separada por comas).

En la sintaxis del trigger, el apartado OR permite asociar más de un evento al trigger (se puede indicar INSERT OR UPDATE por ejemplo).

Ejemplo:

```
CREATE OR REPLACE
TRIGGER ins_personal
BEFORE INSERT ON personal
BEGIN
  IF (TO_CHAR(SYSDATE, 'HH24') NOT IN ('10', '11', '12')) THEN
    RAISE_APPLICATION_ERROR
      (-20001, 'Sólo se puede añadir personal entre las 10 y las 12:59');
  END IF;
END;
```

Este trigger impide que se puedan añadir registros a la tabla de personal si no estamos entre las 10 y las 13 horas.

Triggers de fila

Sintaxis básica:

```
CREATE [ OR REPLACE ]
TRIGGER Nombre
{ BEFORE | AFTER } evento1 [OR evento2[, ...]] ON tabla
FOR EACH ROW [ WHEN condición ]
[ DECLARE
  declaraciones ]
BEGIN
  cuerpo
  [ EXCEPTION
    captura de excepciones ]
END;
```

La cláusula FOR EACH ROW hace que el trigger sea de fila, es decir que se repita su ejecución por cada fila afectada en la tabla por la instrucción DML. El apartado WHEN permite colocar una condición que deben de cumplir los registros para que el trigger se ejecute. Sólo se ejecuta el trigger para las filas que cumplan dicha condición.

Referencias NEW y OLD

Cuando se ejecutan instrucciones UPDATE, hay que tener en cuenta que se modifican valores antiguos (OLD) para cambiarlos por valores nuevos (NEW). Las palabras NEW y OLD permiten acceder a los valores nuevos y antiguos respectivamente. En el apartado de instrucciones del trigger (BEGIN ... END) serían :NEW.nombre y :OLD.nombre. Imaginemos que deseamos hacer una auditoría sobre una tabla en la que tenemos un listado de las piezas mecánicas que fabrica una determinada empresa. Esa tabla es PIEZAS y contiene el tipo y el modelo de la pieza (los dos campos forman la clave de la tabla) y el precio de venta de la misma. Deseamos almacenar en otra tabla diferente los cambios de precio que realizamos a las piezas, para lo cual creamos la siguiente tabla:

```
CREATE TABLE PIEZAS_AUDITORIA
(
    precio_viejo NUMBER(11,4),
    precio_nuevo NUMBER(11,4),
    tipo          VARCHAR2(2),
    modelo        NUMBER(2),
    fecha         DATE
);
```

Como queremos que la tabla se actualice automáticamente, creamos el siguiente trigger:

```
CREATE OR REPLACE
TRIGGER hacer_auditoria_piezas
BEFORE UPDATE OF precio_venta ON PIEZAS
FOR EACH ROW WHEN (OLD.precio_venta < NEW.precio_venta)

BEGIN

    INSERT INTO PIEZAS_AUDITORIA
    VALUES (
        :OLD.precio_venta,
        :NEW.precio_venta,
        :OLD.tipo,
        :OLD.modelo,
        SYSDATE );

END hacer_auditoria_piezas;
```

Con este trigger cada vez que se modifiquen un registros de la tabla de piezas, siempre y cuando se esté incrementado el precio, se añade una nueva fila por registro modificado en la tabla de auditorías, observar el uso de NEW y de OLD y el uso de los dos puntos (:NEW y :OLD) en la sección ejecutable. Cuando se añaden registros, los valores de OLD son todos nulos. Cuando se borran registros, son los valores de NEW los que se borran.

IF INSERTING, IF UPDATING e IF DELETING

Son palabras que se utilizan para determinar la instrucción DML que se estaba realizando cuando se lanzó el trigger. Esto se utiliza en triggers que se lanza para varias operaciones (utilizando INSERT OR UPDATE por ejemplo). En ese caso se pueden utilizar sentencias IF seguidas de INSERTING, UPDATING o DELETING; éstas palabras devolverán TRUE si se estaba realizando dicha operación.

```
CREATE OR REPLACE
TRIGGER nombre
BEFORE INSERT OR DELETE OR UPDATE OF campol ON tabla
FOR EACH ROW

BEGIN
    IF DELETING THEN
        instrucciones que se ejecutan si el TRIGGER saltó por borrar filas
```

(continué en la próxima página)

(proviene de la página anterior)

```

ELSIF INSERTING THEN
    instrucciones que se ejecutan si el TRIGGER saltó por insertar filas
ELSE
    instrucciones que se ejecutan si el TRIGGER saltó por modificar fila
END IF;
END;

```

Triggers sobre vistas

Hay un tipo de trigger especial que se llama **INSTEAD OF** y que sólo se utiliza con las vistas. Una vista es una consulta **SELECT** almacenada. En general sólo sirven para mostrar datos, pero podrían ser interesantes para actualizar, por ejemplo en esta declaración de vista:

```

CREATE VIEW existenciasCompleta (tipo,modelo,precio,almacen,cantidad)
AS
SELECT p.tipo, p.modelo, p.precio_venta,e.n_almacen, e.cantidad
FROM PIEZAS p JOIN EXISTENCIAS e ON p.tipo=e.tipo AND p.modelo=e.modelo
ORDER BY p.tipo,p.modelo,e.n_almacen;

```

Esta instrucción daría lugar a error

```

INSERT INTO existenciasCompleta VALUES ('ZA', 3, 4, 3, 200);

```

Indicando que esa operación no es válida en esa vista (al utilizar dos tablas). Esta situación la puede arreglar un trigger que inserte primero en la tabla de piezas (sólo si no se encuentra ya insertada esa pieza) y luego inserte en existencias.

Eso lo realiza el trigger de tipo **INSTEAD OF**, que sustituirá el **INSERT** original por el código indicado por el trigger:

```

CREATE OR REPLACE
TRIGGER ins_piezas_exis
INSTEAD OF INSERT ON existenciasCompleta

BEGIN
    INSERT INTO piezas(tipo,modelo,precio_venta)
    VALUES (:NEW.tipo, :NEW.modelo, :NEW.precio);

    INSERT INTO existencias(tipo,modelo,n_almacen,cantidad)
    VALUES (:NEW.tipo, :NEW.modelo, :NEW.almacen, :NEW.cantidad);
END;

```

Este trigger permite añadir a esa vista añadiendo los campos necesarios en las tablas relacionadas en la vista. Se podría modificar el trigger para permitir actualizar, eliminar o borrar datos directamente desde la vista y así cualquiera desde cualquier acceso a la base de datos utilizaría esa vista como si fuera una tabla más. Orden de ejecución de los triggers

Puesto que sobre una misma tabla puede haber varios triggers, es necesario conocer en qué orden se ejecutan los mismos. El orden es:

1. Primero disparadores de tipo **BEFORE** de tipo instrucción
2. Disparadores de tipo **BEFORE** por cada fila
3. Se ejecuta la propia orden que desencadenó al trigger.
4. Disparadores de tipo **AFTER** con nivel de fila.
5. Disparadores de tipo **AFTER** con nivel de instrucción.

Errores de compilación

Es frecuente que cometamos algún tipo de error cuando definimos distintos tipos de bloques. Podemos consultar los errores de compilación mediante la vista `USER_ERRORS`;

```
SELECT * FROM USER_ERRORS;
```

Podemos acotar la consulta estableciendo una condición de búsqueda. Las más interesantes suelen ser el nombre del bloque y el tipo de bloque.

```
SELECT * FROM USER_ERRORS  
WHERE NAME='nombre_bloque' AND TYPE='tipo_bloque';
```

La columna `TYPE` puede tomar uno de los siguientes valores:

```
PROCEDURE  
FUNCTION  
PACKAGE  
PACKAGE BODY  
TRIGGER
```

Por ejemplo, para ver los errores producidos en el siguiente trigger:

```
CREATE OR REPLACE  
TRIGGER Control_Empleados  
AFTER INSERT OR DELETE OR UPDATE ON Empleados  
  
BEGIN  
    INSERT INTO Ctrl_Empleados (Tabla, Usuario, Fecha)  
    VALUES ('Empleados', USER, SYSDATE);  
END Control_Empleados;  
/
```

podemos realizar la siguiente consulta:

```
SELECT name, type, text FROM user_errors  
WHERE name='CONTROL_EMPLEADOS' AND type='TRIGGER';
```

Otra forma más cómoda de hacerlo es con la sentencia:

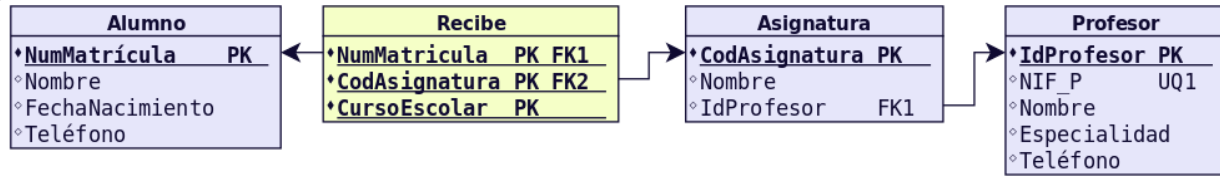
```
SHOW ERRORS;
```

5.5 ACTIVIDADES PROPUESTAS

5.5.1 Práctica 1

INSERT / UPDATE / DELETE

Haciendo uso del esquema `E01` cuyo diseño físico realizamos en el tema 3, realiza las operaciones de manipulación de datos indicadas a continuación.



Realiza las siguientes inserciones (los datos puedes inventarlos).

- Inserta 2 profesores.
- Inserta 4 asignaturas.
- Inserta 10 alumnos.
- Cada alumno debe realizar al menos 2 asignaturas.

Introduce 2 profesores con el mismo NIF. ¿Qué sucede? ¿Por qué?

Introduce 2 alumnos con el mismo NumMatricula. ¿Qué sucede? ¿Por qué?

Introduce 3 alumnos para los cuales no conocemos el número de teléfono.

Modifica los datos de los 3 alumnos anteriores para establecer un número de teléfono.

Para todos los alumnos, poner 2000 como año de nacimiento.

Para los profesores que tienen número de teléfono y NIF no comience por 9, poner “Informática” como especialidad.

Cambia la asignación de asignaturas para los profesores. Es decir, las asignaturas impartidas por un profesor las dará el otro profesor y viceversa.

En la tabla Recibe borra todos los registros que pertenecen a una de las asignaturas.

En la tabla Asignatura borra dicha asignatura.

Borra el resto de asignaturas. ¿Qué sucede? ¿Por qué? ¿Como lo solucionarías? ¿Podría haberse evitado el problema con otro diseño físico? ¿Cómo?

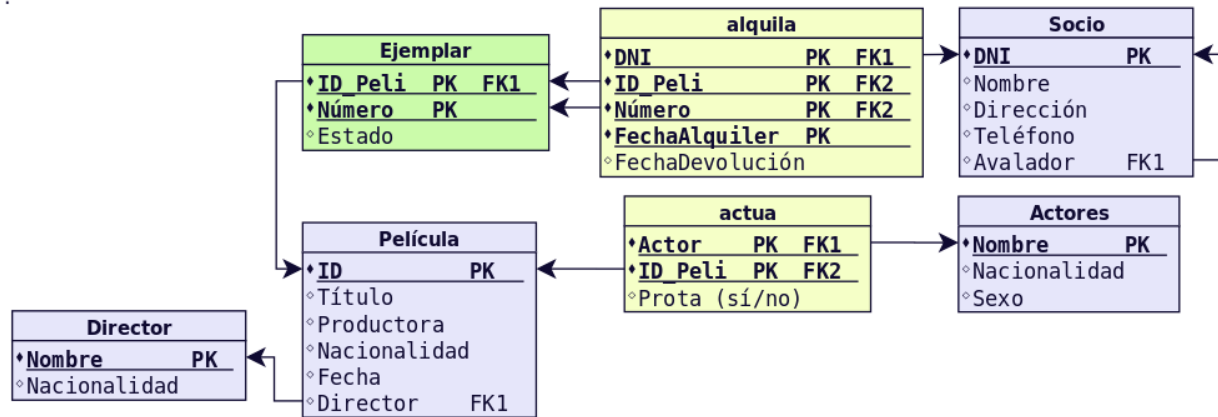
Borra todos los profesores. ¿Qué sucede? ¿Por qué? ¿Como lo solucionarías? ¿Podría haberse evitado el problema con otro diseño físico? ¿Cómo?

Borra todos los alumnos. ¿Qué sucede? ¿Por qué? ¿Como lo solucionarías? ¿Podría haberse evitado el problema con otro diseño físico? ¿Cómo?

5.5.2 Práctica 2

INSERT / UPDATE / DELETE

Haciendo uso del esquema E07 cuyo diseño físico realizamos en el tema 3, realiza las operaciones de manipulación de datos indicadas a continuación.



Teniendo en cuenta las siguientes restricciones que teníamos declaradas:

- No pueden ser nulos los siguientes campos: Nombre de Socio, Título de Película.
- Sexo toma los valores H o M.
- Por defecto si no se indica nada un actor o actriz no es Protagonista (este campo toma valores S o N).
- FechaDevolución debe ser mayor que FechaAlquiler.

Realiza las siguientes inserciones (los datos puedes inventarlos).

- Inserta 2 directores.
- Inserta 4 películas. Todas ellas están dirigidas por alguno de los directores anteriores.
- Inserta 2 ejemplares de cada película.
- Inserta 4 socios.
- Inserta como mínimo 6 actores.
- Cada película debe tener al menos el actor/atriz protagonista asociado.
- Todos los ejemplares deben tener al menos 1 alquiler.

Inserta valores para comprobar que la siguiente restricción funciona correctamente:

- No pueden ser nulos los siguientes campos: Nombre de Socio, Título de Película.

Inserta valores para comprobar que la siguiente restricción funciona correctamente:

- Sexo toma los valores H o M.

Inserta valores para comprobar que la siguiente restricción funciona correctamente:

- Por defecto si no se indica nada un actor o actriz no es Protagonista (este campo toma valores S o N).

Inserta valores para comprobar que la siguiente restricción funciona correctamente:

- FechaDevolución debe ser mayor que FechaAlquiler.

Cambia la nacionalidad para los directores. Por ejemplo de “Estadounidense” a “USA” o similar, dependiendo de los valores que hayas introducido.

Cambia la nacionalidad para los actores. Por ejemplo de “Estadounidense” a “USA” o similar, dependiendo de los valores que hayas introducido.

Modifica los datos de todos los socios para que el avalista sea un único socio, siempre el mismo para todos, excepto para el avalista mismo que no dispone de ninguno.

Elimina los socios cuyo número de teléfono empiece por una cifra inferior a 5. ¿Qué sucede? ¿Por qué?

Elimina los socios cuyo número de teléfono empiece por una cifra superior o igual a 5. ¿Qué sucede? ¿Por qué?

¿Como lo solucionarías los problemas anteriores? ¿Podría haberse evitado el problema con otro diseño físico? ¿Cómo?

Elimina todos los directores. ¿Qué sucede? ¿Por qué?

Elimina 2 películas, las que desees. ¿Qué sucede? ¿Por qué? ¿Como lo solucionarías? ¿Podría haberse evitado el problema con otro diseño físico? ¿Cómo?

5.5.3 Práctica 3

PL/SQL: Introducción

Las siguientes prácticas se realizarán dentro del esquema EMPLEADOS.

Realiza una conexión utilizando el cliente SQL*Plus y muestra el valor de las siguientes variables: USER, ESCAPE, LINESIZE, COLSEP, PAGESIZE, ECHO, SQLPROMPT

Desde el cliente SQL*Plus muestra el valor de las variables AUTOCOMMIT y SERVEROUTPUT.

Desde el cliente SQL*Plus ejecuta el comando HELP SHOW para ver la ayuda acerca del comando SHOW.

Desde el cliente SQL*Plus ejecuta el comando HELP SET para ver la ayuda acerca del comando SET.

Desde el cliente SQL*Plus pon a ON las variables SERVEROUTPUT y AUTOCOMMIT.

Crea un esquema llamado PLSQL con contraseña PLSQL y rol DBA para realizar las siguientes actividades. Ejecuta el siguiente bloque. Indica cuál es la salida.

```
BEGIN
  IF 10 > 5 THEN
    DBMS_OUTPUT.PUT_LINE ('Cierto');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('Falso');
  END IF;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
BEGIN
  IF 10 > 5 AND 5 > 1 THEN
    DBMS_OUTPUT.PUT_LINE ('Cierto');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('Falso');
  END IF;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
BEGIN
  IF 10 > 5 AND 5 > 50 THEN
    DBMS_OUTPUT.PUT_LINE ('Cierto');
  ELSE
    DBMS_OUTPUT.PUT_LINE ('Falso');
  END IF;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
BEGIN
  CASE
    WHEN 10 > 5 AND 5 > 50 THEN
      DBMS_OUTPUT.PUT_LINE ('Cierto');
    ELSE
      DBMS_OUTPUT.PUT_LINE ('Falso');
  END CASE;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
BEGIN
  FOR i IN 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
BEGIN
  FOR i IN REVERSE 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE (i);
  END LOOP;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
DECLARE
  num NUMBER(3) := 0;
BEGIN
  WHILE num<=100 LOOP
    DBMS_OUTPUT.PUT_LINE (num);
    num:= num+2;
  END LOOP;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
DECLARE
  num NUMBER(3) := 0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE (num);
    IF num > 100 THEN EXIT; END IF;
    num:= num+2;
  END LOOP;
END;
/
```

Ejecuta el siguiente bloque. Indica cuál es la salida.

```
DECLARE
  num NUMBER(3) := 0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE (num);
    EXIT WHEN num > 100;
    num:= num+2;
  END LOOP;
END;
/
```

5.5.4 Práctica 4

PL/SQL: Procedimientos y Funciones

Crea un procedimiento llamado **ESCRIBE** para mostrar por pantalla el mensaje **HOLA MUNDO**.

Crea un procedimiento llamado **ESCRIBE_MENSAJE** que tenga un parámetro de tipo **VARCHAR2** que recibe un texto y lo muestre por pantalla. La forma del procedimiento será la siguiente:

ESCRIBE_MENSAJE (mensaje VARCHAR2)

Crea un procedimiento llamado **SERIE** que muestre por pantalla una serie de números desde un mínimo hasta un máximo con un determinado paso. La forma del procedimiento será la siguiente:

SERIE (minimo NUMBER, maximo NUMBER, paso NUMBER)

Crea una función **AZAR** que reciba dos parámetros y genere un número al azar entre un mínimo y máximo indicado. La forma de la función será la siguiente:

AZAR (minimo NUMBER, maximo NUMBER) RETURN NUMBER

Crea una función **NOTA** que reciba un parámetros que será una nota numérica entre 0 y 10 y devuelva una cadena de texto con la calificación (Suficiente, Bien, Notable, ...). La forma de la función será la siguiente:

NOTA (nota NUMBER) RETURN VARCHAR2

5.5.5 Práctica 5

PL/SQL: Procedimientos y Funciones

Escribe un procedimiento que muestre el número de empleados y el salario mínimo, máximo y medio del departamento de FINANZAS. Debe hacerse uso de cursores implícitos, es decir utilizar **SELECT ... INTO**.

Escribe un procedimiento que suba un 10 % el salario a los EMPLEADOS con más de 2 hijos y que ganen menos de 2000 €. Para cada empleado se mostrará por pantalla el código de empleado, nombre, salario anterior y final. Utiliza un cursor explícito. La transacción no puede quedarse a medias. Si por cualquier razón no es posible actualizar todos estos salarios, debe deshacerse el trabajo a la situación inicial.

Escribe un procedimiento que reciba dos parámetros (número de departamento, hijos). Deberá crearse un cursor explícito al que se le pasarán estos parámetros y que mostrará los datos de los empleados que pertenezcan al departamento y con el número de hijos indicados. Al final se indicará el número de empleados obtenidos.

Escribe un procedimiento con un parámetro para el nombre de empleado, que nos muestre la edad de dicho empleado en años, meses y días.

5.5.6 Práctica 6

PL/SQL: Paquetes

Desarrolla el paquete **ARITMETICA** cuyo código fuente viene en este tema. Crea un archivo para la especificación y otro para el cuerpo. Realiza varias pruebas para comprobar que las llamadas a funciones y procedimiento funcionan correctamente.

Al paquete anterior añade una función llamada **RESTO** que reciba dos parámetros, el dividendo y el divisor, y devuelva el resto de la división.

Al paquete anterior añade un procedimiento sin parámetros llamado AYUDA que muestre un mensaje por pantalla de los procedimientos y funciones disponibles en el paquete, su utilidad y forma de uso.

Desarrolla el paquete GESTION. En un principio tendremos los procedimientos para gestionar los departamentos. Dado el archivo de especificación mostrado más abajo crea el archivo para el cuerpo. Realiza varias pruebas para comprobar que las llamadas a funciones y procedimientos funcionen correctamente.

5.5.7 Práctica 7

PL/SQL: Triggers

Previamente deberemos crear una tabla AUDITORIA_EMPLEADOS para registrar los eventos a auditar que ocurran sobre la tabla EMPLEADOS.

```
CREATE TABLE AUDITORIA_EMPLEADOS (descripcion VARCHAR2(200));
```

Y también crearemos una vista SEDE_DEPARTAMENTOS acerca de los departamentos y su localización.

```
CREATE VIEW SEDE_DEPARTAMENTOS AS
SELECT C.NUMCE, C.NOMCE, C.DIRCE,
       D.NUMDE, D.NOMDE, D.PRESU, D.DIREC, D.TIDIR, D.DEPDE
FROM CENTROS C JOIN DEPARTAMENTOS D ON C.NUMCE=D.NUMCE;
```

También insertaremos en la tabla DEPARTAMENTOS uno llamado TEMP donde serán movidos los empleados cuyo departamento desaparezca.

```
INSERT INTO DEPARTAMENTOS VALUES (0, 10, 260, 'F', 10, 100, 'TEMP');
```

Crea un trigger que, cada vez que se inserte o elimine un empleado, registre una entrada en la tabla AUDITORIA_EMPLEADOS con la fecha del suceso, número y nombre de empleado, así como el tipo de operación realizada (INSERCIÓN o ELIMINACIÓN).

Crea un trigger que, cada vez que se modifiquen datos de un empleado, registre una entrada en la tabla AUDITORIA_EMPLEADOS con la fecha del suceso, valor antiguo y valor nuevo de cada campo, así como el tipo de operación realizada (en este caso MODIFICACIÓN).

Crea un trigger para que registre en la tabla AUDITORIA_EMPLEADOS las subidas de salarios superiores al 5%.

Deseamos operar sobre los datos de los departamentos y los centros donde se hallan. Para ello haremos uso de la vista SEDE_DEPARTAMENTOS creada anteriormente.

Al tratarse de una vista que involucra más de una tabla, necesitaremos crear un trigger de sustitución para gestionar las operaciones de actualización de la información. Crea el trigger necesario para realizar inserciones, eliminaciones y modificaciones en la vista anterior.

Realiza las siguientes operaciones para comprobar si el disparador anterior funciona correctamente.

SEGURIDAD DE LOS DATOS. LENGUAJE DE CONTROL DE DATOS

6.1 CONFIGURACIÓN

6.1.1 Usuarios

A nivel global existen 2 grandes grupos de usuarios:

- Usuarios de la Base de Datos Oracle.
- Usuarios de la interfaz web APEX.

Usuarios de la base de datos

Los usuarios de la base de datos son los usuarios más importantes. Cada usuario de la base de datos tiene asignado un esquema del mismo nombre (USUARIO = ESQUEMA). Para crear, modificar y eliminar este tipo de usuarios se utilizan las siguientes sentencias:

```
CREATE USER ...  
ALTER USER ...  
DROP USER ...
```

Podemos ver todos los usuarios con una consulta a la vista ALL_USERS;

```
SELECT USERNAME FROM ALL_USERS;
```

Existen 2 usuarios, que se crean automáticamente durante la instalación. Los dos poseen rol DBA (DataBase Administrator), es decir son administradores.

- **SYS**

Esta cuenta puede realizar todas las funciones administrativas. Todas las tablas base (subyacentes) y las vistas del diccionario de datos de base de datos se almacenan en el esquema SYS. Estas tablas base y vistas son fundamentales para el funcionamiento de Oracle Database. Para mantener la integridad del diccionario de datos, las tablas del esquema SYS solo son manipuladas por la base de datos.

Nunca deben ser modificadas por ningún usuario o administrador de base de datos. No se debe crear ninguna tabla en el esquema SYS. Al usuario SYS se le concede el privilegio SYSDBA, que permite al usuario realizar tareas administrativas de alto nivel, como copia de seguridad y recuperación.

■ SYSTEM

Esta cuenta puede realizar todas las funciones administrativas excepto las siguientes:

- Copia de seguridad y recuperación.
- Actualización de la base de datos.

Si bien esta cuenta puede utilizarse para realizar tareas administrativas diarias, Oracle recomienda encarecidamente la creación de cuentas de usuarios con nombre para administrar la base de datos Oracle para permitir el seguimiento de la actividad de la base de datos.

Podemos modificar la contraseña del usuario SYSTEM desde SQLPLUS de la siguiente forma:

```
SQLPLUS / AS SYSDBA

ALTER USER SYSTEM IDENTIFIED BY "SYSTEM";
```

Usuarios de la interfaz web APEX

Los usuarios de APEX son los utilizados para trabajar en los espacios de trabajo (Workspace).

Importante: Estos usuarios NO son usuarios de la base de datos. Por tanto, no podremos conectarnos a la base de datos con estos usuarios. Sólo pueden conectarse a un workspace de APEX.

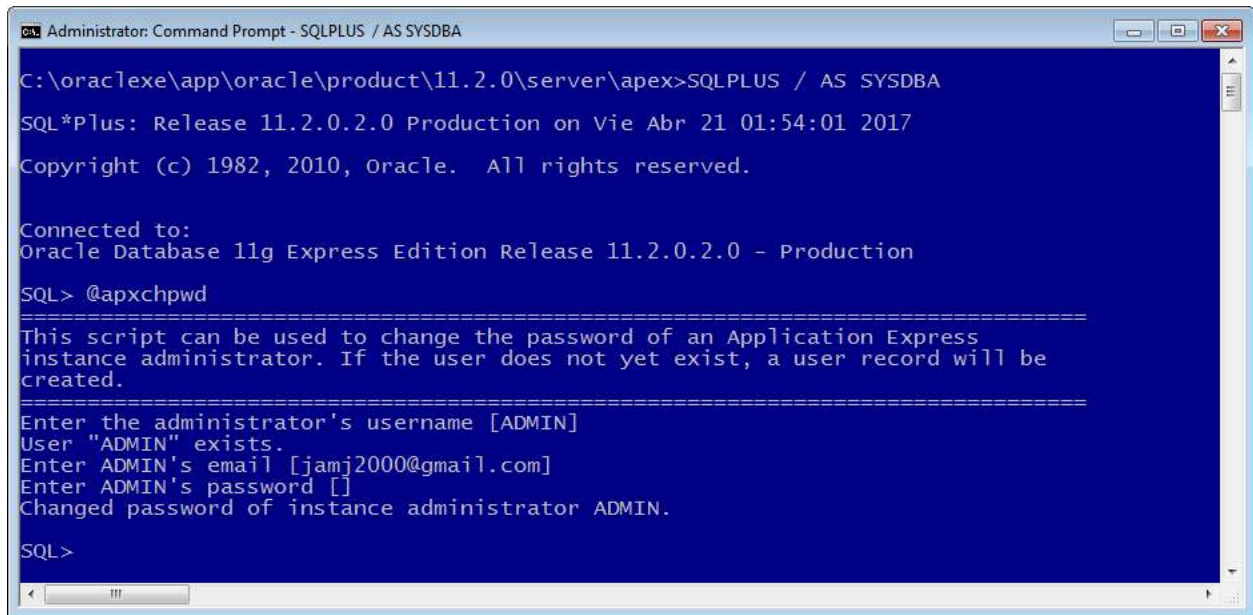
Existe un usuario administrador cuyo nombre es **ADMIN**, que se encarga de gestionar la creación, modificación y eliminación de espacios de trabajo, entre otras cosas.

Justo después de la instalación este usuario tiene la misma contraseña que el usuario SYSTEM de la base de datos. Pero en el primer acceso a APEX con ADMIN, se nos pedirá que cambiemos la contraseña.

Si por cualquier motivo perdemos u olvidamos su contraseña, podemos establecerla de nuevo con el script C:\oracle\app\oracle\product\11.2.0\server\apexapxchpwd.sql.

Para ello ejecutamos en un terminal de texto:

```
CD C:\oracle\app\oracle\product\11.2.0\server\apex
SQLPLUS / AS SYSDBA
@APXCHPWD
```



```

Administrator: Command Prompt - SQLPLUS / AS SYSDBA
C:\oracle\app\oracle\product\11.2.0\server\apex>SQLPLUS / AS SYSDBA
SQL*Plus: Release 11.2.0.2.0 Production on Vie Abr 21 01:54:01 2017
Copyright (c) 1982, 2010, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - Production
SQL> @apxchpwd
=====
This script can be used to change the password of an Application Express
instance administrator. If the user does not yet exist, a user record will be
created.
=====
Enter the administrator's username [ADMIN]
User "ADMIN" exists.
Enter ADMIN's email [jamj2000@gmail.com]
Enter ADMIN's password []
Changed password of instance administrator ADMIN.

SQL>

```

En este caso es aconsejable poner una contraseña corta y sencilla, puesto que después, cuando accedamos a APEX, nos pedirá que la cambiemos, y entonces la política de contraseñas es bastante estricta:

- al menos 6 caracteres
- debe contener algún carácter numérico
- debe contener algún carácter no alfanumérico: % & \$ _ . . .
- debe contener alguna minúscula
- no contener el nombre de usuario
- no parecerse a la contraseña anterior

6.1.2 Interfaces

Durante este curso hemos trabajado con Oracle Express Edition (XE) 11gR2. Hemos hecho uso de 2 interfaces:

- Interfaz de texto o comandos: **SQL*PLUS**.
- Interfaz web: **APEX** (Application Express)

La interfaz de comandos es muy útil para realizar tareas administrativas de forma rápida y cómoda. Por ejemplo, crear, modificar o eliminar esquemas (usuarios); conceder privilegios; ejecutar scripts SQL, etc.

La interfaz web es más cómoda para trabajar con código PL/SQL. También para la importación y exportación de datos en un entorno amigable. Por supuesto, además puede utilizarse para realizar operaciones DML (consultas, inserciones, modificaciones y eliminaciones).

Podemos comprobar la versión de APEX consultando la vista APEX_RELEASE:

```
SELECT VERSION_NO FROM APEX_RELEASE;
```

Captura de pantalla de interface APEX 4, la que viene con Oracle XE 11gR2:

Observa, donde se encuentra la página de administración para APEX 4. En ella podremos crear, modificar y eliminar espacios de trabajo (workspaces), entre otras cosas.

6.1.3 Configuración de Oracle XE

Cuanto instalamos Oracle Express Edition, éste viene con una configuración que establece una serie de parámetros que condicionan la forma en la que se guardan y representan los datos.

Debemos distinguir entre 2 tipos de configuraciones:

- Configuración del Sistema Gestor de BBDD.
- Configuración de las sesiones.

Configuración del Sistema Gestor de BBDD.

Para ver dichos parámetros ejecutamos la siguiente sentencia:

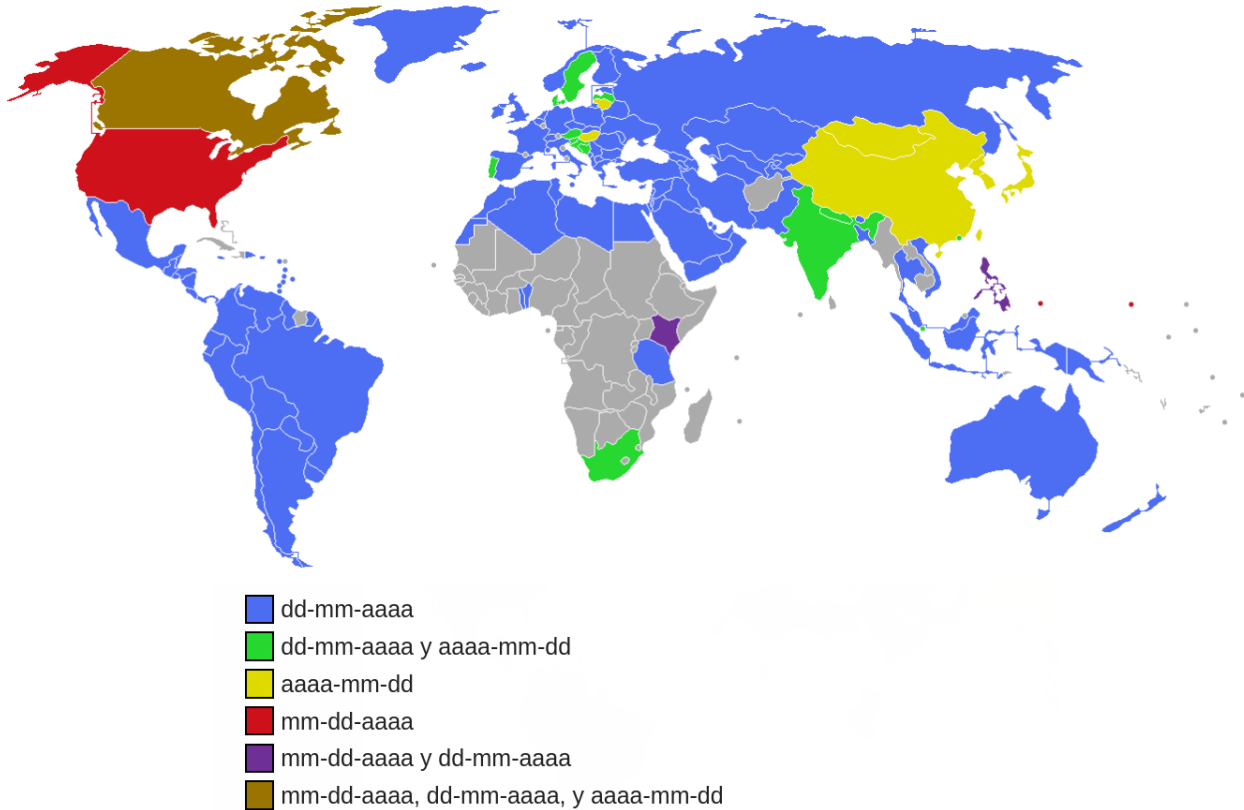
```
SELECT * FROM NLS_DATABASE_PARAMETERS;
```

PARAMETER	VALUE
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	\$
NLS_ISO_CURRENCY	AMERICA
NLS_NUMERIC_CHARACTERS	.,
NLS_CHARACTERSET	AL32UTF8
NLS_CALENDAR	GREGORIAN
NLS_DATE_FORMAT	DD-MON-RR
NLS_DATE_LANGUAGE	AMERICAN
NLS_SORT	BINARY
More than 10 rows available. Increase rows selector to view more rows.	

Los parámetros más importantes son:

- La codificación de caracteres (**NLS_CHARACTERSET**): Unicode (UTF8 u otro), Windows-1252, ... Esto afecta a la representación de tildes y caracteres como Ñ.
- El idioma (**NLS_LANGUAGE**)
- El país (**NLS_TERRITORY**)

- La moneda (**NLS_CURRENCY**): \$, €, ...
- Los separadores en los números (**NLS_NUMERIC_CHARACTERS**):
 - Separador decimal (D). En Estados Unidos es el punto.
 - Separador de grupo (G). En Estados Unidos es la coma.
- El formato de fecha (**NLS_DATE_FORMAT**). Este puede variar en gran manera de un país a otro. A continuación se muestra un gráfico tomado de Wikipedia.



Configuración de las sesiones.

Para comprobar los parámetros NLS de la sesión:

```
SELECT * FROM NLS_SESSION_PARAMETERS;
```

Ejecutado en APEX:

PARAMETER	VALUE
NLS_LANGUAGE	AMERICAN
NLS_TERRITORY	AMERICA
NLS_CURRENCY	\$
NLS_ISO_CURRENCY	AMERICA
NLS_NUMERIC_CHARACTERS	.,
NLS_CALENDAR	GREGORIAN
NLS_DATE_FORMAT	mm/dd/yyyy
NLS_DATE_LANGUAGE	AMERICAN
NLS_SORT	BINARY
NLS_TIME_FORMAT	HH.MI.SSXFF AM
More than 10 rows available. Increase rows selector to view more rows.	

Ejecutado en SQL*Plus:

```

Administrator: Command Prompt - sqlplus E01/E01
SQL> SET PAGES 0
SQL>
SQL> COLUMN PARAMETER FORMAT A40
SQL> COLUMN VALUE      FORMAT A40
SQL>
SQL> SELECT * FROM NLS_SESSION_PARAMETERS
2 ;
NLS_LANGUAGE           SPANISH
NLS_TERRITORY          SPAIN
NLS_CURRENCY            €
NLS_ISO_CURRENCY       SPAIN
NLS_NUMERIC_CHARACTERS ,.
NLS_CALENDAR            GREGORIAN
NLS_DATE_FORMAT        DD/MM/RR
NLS_DATE_LANGUAGE      SPANISH
NLS_SORT               SPANISH
NLS_TIME_FORMAT        HH24:MI:SSXFF
NLS_TIMESTAMP_FORMAT   DD/MM/RR HH24:MI:SSXFF
NLS_TIME_TZ_FORMAT     HH24:MI:SSXFF TZR
NLS_TIMESTAMP_TZ_FORMAT DD/MM/RR HH24:MI:SSXFF TZR
NLS_DUAL_CURRENCY      €
NLS_COMP               BINARY
NLS_LENGTH_SEMANTICS   BYTE
NLS_NCHAR_CONV_EXCP   FALSE

17 rows selected.

SQL>

```

Podemos cambiar la configuración de la sesión. Ejemplos:

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY';
ALTER SESSION SET NLS_LANGUAGE = SPANISH;
```

Importante: Las sentencias anteriores funcionan sólo para SQL*Plus.

Para APEX deberemos reinstalar dicho entorno y habilitar el idioma español. Lo vemos en el siguiente apartado.

6.1.4 Instalación de APEX 5.1

Pasos para instalar una nueva versión de APEX y ponerla en Español:

1. Descargar Oracle Application Express - All languages. El archivo apex_5.1.zip contiene APEX 5.1. Es posible bajarlo desde el sitio web oficial de Oracle habiéndonos registrado previamente.
2. Descomprimir apex_5.1.zip en C:\oraclexe\app\oracle\product\11.2.0\server sobrescribiendo la carpeta apex existente.

Archivo comprimido Archivo Preferencias Ayuda							
Extraer Vista previa Abrir Añadir archivo Borrar							
Nombre	Tamaño	Comprimido	Modo	CRC	Método	Fecha	
apex	5 carpetas, 18 archivos				Store	18/2/12 11:06	
builder	9 carpetas, 14 archivos				Store	18/2/12 10:51	
core	570 archivos				Store	20/2/12 19:49	
images	23 carpetas, 830 archivos				Store	18/2/12 10:51	
owa	62 archivos				Store	18/2/12 11:06	
utilities	2 carpetas, 3 archivos				Store	18/2/12 10:51	
apex_epg_config.sql	690 B	388 B		61409014	Deflate	10/9/08 10:17	
apex_epg_config_core.sql	15,7 KiB	4,5 KiB		C6305A88	Deflate	16/4/10 15:23	
apexins.sql	7,7 KiB	2,3 KiB		3F6ECCFE	Deflate	11/12/11 2:46	
apexvalidate.sql	13,6 KiB	3,2 KiB		B4F4BD18	Deflate	17/6/11 19:08	
apxchpwd.sql	1,7 KiB	700 B		3F0CF8D9	Deflate	9/8/11 13:27	
apxconf.sql	2,9 KiB	1,1 KiB		5B131B9B	Deflate	4/5/11 0:33	
apxdevrm.sql	9,7 KiB	2,8 KiB		20F5B396	Deflate	8/2/12 19:48	
apxdvins.sql	3,2 KiB	1,3 KiB		096F1850	Deflate	9/8/11 13:27	
apxldimg.sql	8,1 KiB	2,4 KiB		976B06EC	Deflate	16/4/10 15:23	
apxremov.sql	4,7 KiB	1,5 KiB		F9A6C075	Deflate	4/5/11 0:33	
apxrtins.sql	5,6 KiB	1,8 KiB		8C93EAF8	Deflate	11/12/11 2:46	
apxsqler.sql	44 B	42 B		D2C52CAC	Deflate	26/4/07 16:39	
apxxemig.sql	8,8 KiB	2,9 KiB		177E51E8	Deflate	4/5/11 0:33	
apxxepwd.sql	1,6 KiB	639 B		7B9517E2	Deflate	4/5/11 0:33	
coreins.sql	105,8 KiB	24,6 KiB		CA166469	Deflate	2/2/12 19:12	
devins.sql	13,4 KiB	3,4 KiB		08350653	Deflate	27/1/12 22:03	
endins.sql	2,3 KiB	954 B		2BBAF6DE	Deflate	9/8/11 13:27	
load_trans.sql	1,4 KiB	631 B		A8CF0356	Deflate	4/5/11 0:33	

3. Abrir un terminal CMD
4. Ejecutar: CD C:\oraclexe\app\oracle\product\11.2.0\server\apex

5. Ejecutar: `SQLPLUS / AS SYSDBA`
6. Ejecutar script del instalador: `@apexins SYS AUX SYS AUX TEMP /i/`

Nota: Tardará un buen rato. Al finalizar se cierra SQL*Plus automáticamente.

7. Volver a SQLPLUS: `SQLPLUS / AS SYSDBA`
8. Ejecutar script de carga de imágenes: `@apex_epg_config.sql C:\oraclexe\app\oracle\product\11.2.0\server`
9. Ya tenemos instalada la nueva versión de APEX.

6.1.5 Poner en español APEX 5.1

Para poner el idioma en español realizamos los siguientes pasos:

1. Abrir un terminal CMD
2. Ejecutar: `CD C:\oraclexe\app\oracle\product\11.2.0\server\apex`
3. Poner página de códigos a Unicode: `CHCP 65001`
4. Establecer variable de entorno: `SET NLS_LANG=SPANISH_SPAIN.AL32UTF8`
5. Iniciar SQLPLUS `/ AS SYSDBA`
6. Ejecutar script: `@load_trans.sql SPANISH`

La variable de entorno **NLS_LANG** indica a Oracle qué codificación usa el cliente, así puede hacer las conversiones necesarias para que el cliente visualice correctamente el contenido.

`NLS_LANG=LANGUAGE_TERRITORY.CHARACTERSET`

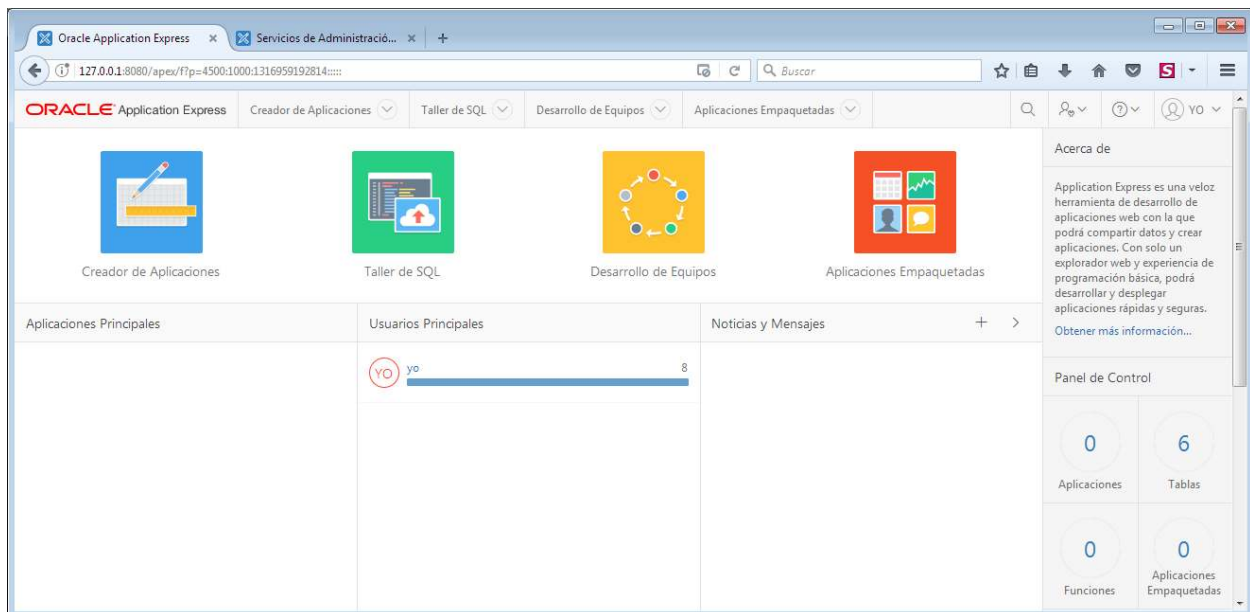
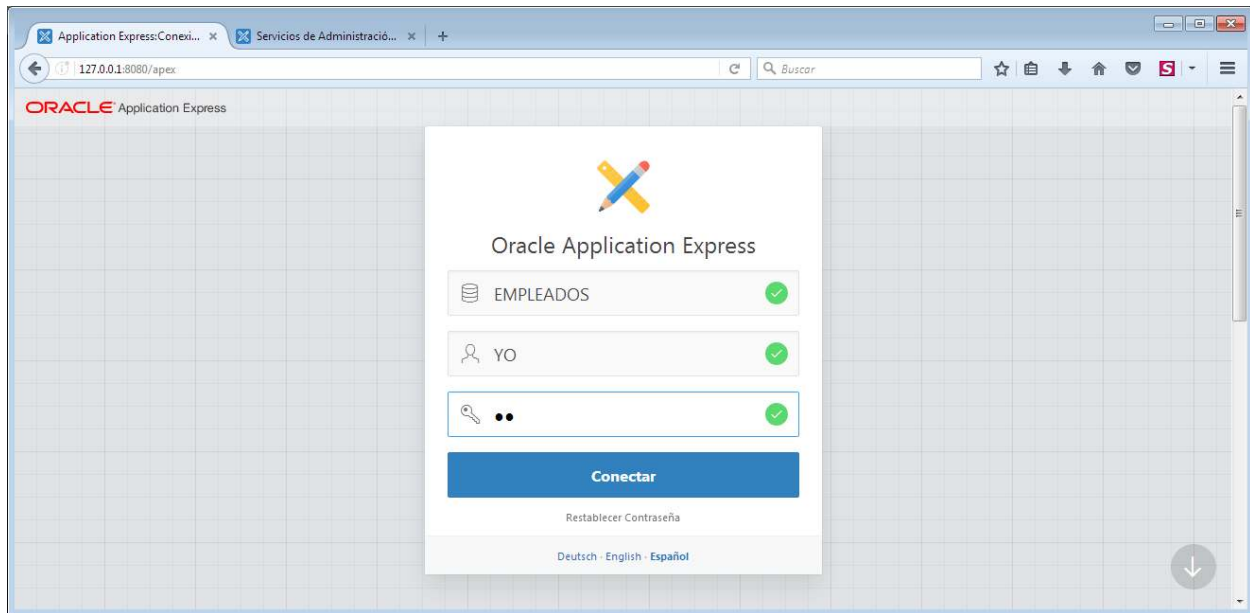
Ejemplos en CMD de Windows:

```
SET NLS_LANG=AMERICAN_AMERICA.AL32UTF8
SET NLS_LANG=SPANISH_SPAIN.WE8ISO8859P1
```

6.1.6 Acceso a APEX 5.1

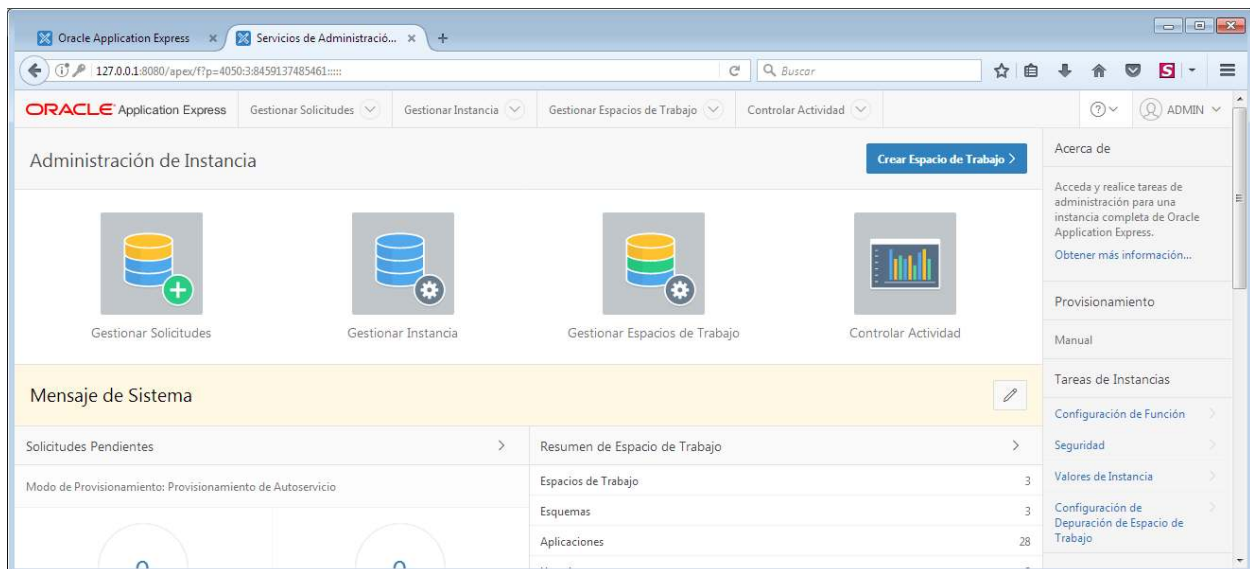
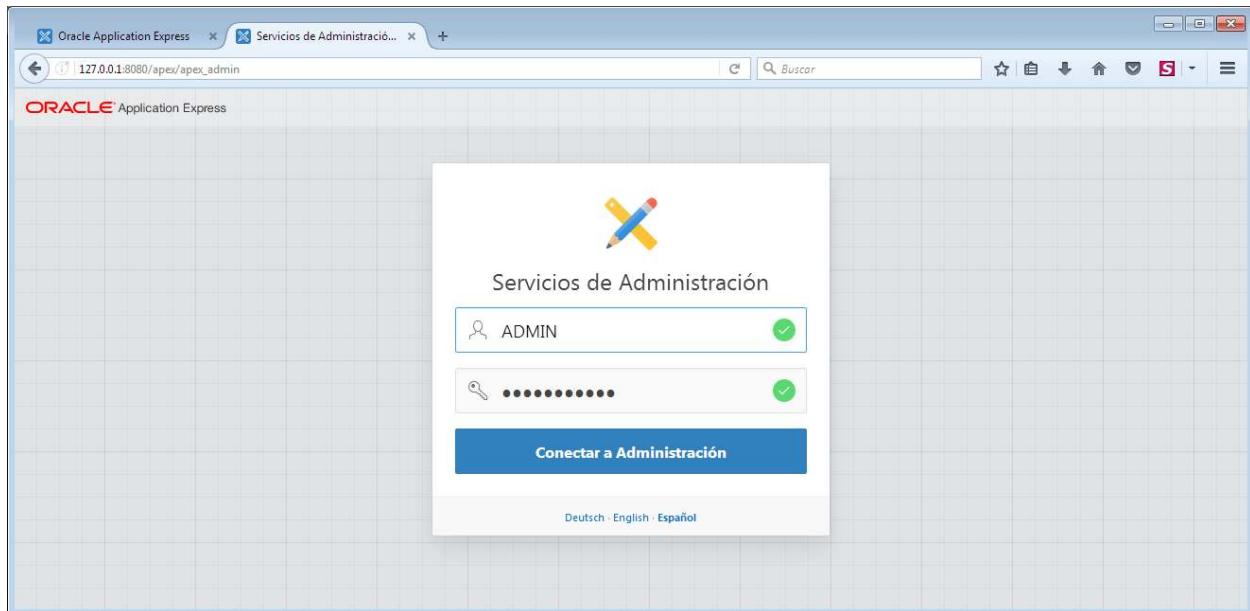
Usuario normal

<http://127.0.0.1:8080/apex>



Usuario ADMIN

http://127.0.0.1:8080/apex/apex_admin



6.2 SEGURIDAD DE LOS DATOS

6.2.1 Privilegios

Los privilegios son permisos que damos a los usuarios para que puedan realizar ciertas operaciones con la base de datos. En Oracle hay más de cien posibles privilegios. Se dividen en:

- **Privilegios de sistema.** Son permisos para modificar el funcionamiento de la base de datos. Son cambios, en definitiva, que afectan a todos los usuarios.
- **Privilegios de objeto.** Son permisos que se aplican a un objeto concreto de la base de datos.

Privilegios del sistema

Hay más de 100 privilegios de sistema distintos. Cada privilegio del sistema permite al usuario realizar una operación de base de datos o una clase de operaciones de base de datos concretas. Algunos de los privilegios del sistema, entre los muchos que existen, son:

Tabla 1: PRIVILEGIOS DE SISTEMA MÁS FRECUENTES

Privilegio	Descripción
CREATE SESSION	Permite al usuario conectar con la base de datos.
CREATE TABLE	Permite crear tablas. Incluye la posibilidad de modificarlas y borrarlas.
CREATE VIEW	Permite crear vistas. Incluye la posibilidad de modificarlas y borrarlas.
CREATE MATERIALIZED VIEW	Permite crear vistas materializadas. Incluye la posibilidad de modificarlas y borrarlas.
CREATE SEQUENCE	Permite crear secuencias. Incluye la posibilidad de modificarlas y borrarlas.
CREATE SYNONYM	Permite crear sinónimos. Incluye la posibilidad de modificarlos y borrarlos.
CREATE PROCEDURE	Permite crear, modificar y borrar un procedimiento PL/SQL, una función o un paquete.
CREATE TRIGGER	Permite crear triggers. Incluye la posibilidad de modificarlos y borrarlos.

Oracle posee dos privilegios de sistema asociados a tareas administrativas, son:

- **SYSDBA**. Con capacidad de parar e iniciar (instrucciones SHUTDOWN y STARTUP) la instancia de base de datos; modificar la base de datos (ALTER DATABASE), crear y borrar bases de datos (CREATE y DROP DATABASE), crear el archivo de parámetros (CREATE SPFILE), cambiar el modo de archivado de la base de datos, recuperar la base de datos y además incluye el privilegio de sistema RESTRICTED SESSION. En la práctica es usar el usuario SYS.
- **SYSOPER**. Permite lo mismo que el anterior salvo: crear y borrar la base de datos y recuperar en todas las formas la base de datos (hay modos de recuperación que requieren el privilegio anterior).

Privilegios de objeto

Los privilegios de objeto más frecuentes son:

Tabla 2: PRIVILEGIOS DE OBJETO MÁS FRECUENTES

Privilegio	Objeto	Descripción
INSERT	Tabla o sinónimo	Permite al usuario insertar en una tabla directamente o a través de un sinónimo.
UPDATE	Tabla	Permite al usuario modificar una tabla.
DELETE	Tabla	Permite al usuario borrar una tabla.
SELECT	Tabla, vista, vista materializada, secuencia o sinónimo	Permite al usuario seleccionar desde una tabla, secuencia, vista, vista materializada o sinónimo.
EXECUTE	Paquete, procedimiento, función de PL/SQL	Permite al usuario ejecutar directamente un paquete, procedimiento o función.

Conceder privilegios de sistema

Se usa con la instrucción GRANT que funciona así:

```
GRANT privilegio1 [,privilegio2[,...]] TO usuario
[WITH ADMIN OPTION];
```

La opción **WITH ADMIN OPTION** permite que el usuario al que se le concede el privilegio puede conceder dicho privilegio a otros usuarios. Es, por tanto, una opción a utilizar con cautela. Se utiliza únicamente con privilegios de sistema.

Ejemplo:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE PROCEDURE
TO usuario;
```

Conceder privilegios de objeto

Se trata de privilegios que se colocan a un objeto para dar permiso de uso a un usuario.

Sintaxis:

```
GRANT {privilegio [(listaColumnas)] [, ...] | ALL [PRIVILEGES]}
ON [esquema.]objeto
TO {usuario | rol} [, {usuario | rol } [, ...]]
[WITH GRANT OPTION];
```

La opción **ALL** concede todos los privilegios posibles sobre el objeto. Se pueden asignar varios privilegios a la vez y también varios posibles usuarios.

La opción **WITH GRANT OPTION** permite al usuario al que se le conceden los privilegios, que pueda, a su vez, concederlos a otro. Se utiliza únicamente con privilegios de objeto.

Ejemplo:

```
GRANT UPDATE, INSERT ON EMPLEADOS.centros TO jose;
```

Revocar privilegios de sistema

```
REVOKE privilegio1 [,privilegio2 [, ...]] FROM usuario;
```

Revocar privilegios de objeto

```
REVOKE {privilegio1 [,privilegio2] [, ...] | ALL [PRIVILEGES]}
ON [esquema.]objeto
FROM {usuario | rol } [, {usuario | rol } [, ...]]
[CASCADE CONSTRAINTS];
```

CASCADE CONSTRAINTS elimina cualquier restricción que impida el borrado del privilegio.

Consultar privilegios

Para ver los privilegios en activo para el usuario y sesión actuales ejecutamos la sentencia:

```
SELECT * FROM SESSION_PRIVS;
```

Nota: El uso de «WITH ADMIN OPTION» y «WITH GRANT OPTION» se considera peligroso en Oracle, porque si no se administran cuidadosamente puede tener efectos secundarios no deseados, resultando en un agujero de seguridad.

6.2.2 Roles

Un rol es un conjunto de privilegios bajo un nombre.

Creación de rol

```
CREATE ROLE nombre_rol;
```

Asignación y retirada de privilegios a roles

Se realiza con la instrucción **GRANT**. A los roles se les asignan privilegios igual que a los usuarios, pueden ser de sistema y/o de objeto. Lógicamente se eliminan mediante **REVOKE**.

```
-- Asignación de privilegios de sistema al rol
GRANT privilegios_de_sistema TO nombre_rol;

-- Asignación de privilegios de objeto al rol
GRANT privilegios_de_objeto ON objeto TO nombre_rol;

-- Retirada de privilegios de sistema al rol
REVOKE privilegios_de_sistema FROM nombre_rol;

-- Retirada de privilegios de objeto al rol
REVOKE privilegios_de_objeto ON objeto FROM nombre_rol;
```

Eliminación de rol

```
DROP ROLE nombre_rol;
```

Ejemplo:

```
-- Creación de rol
CREATE ROLE JEFE;

-- Añadimos privilegios de objeto al rol
GRANT INSERT, SELECT, UPDATE, DELETE ON EMPLEADOS.departamentos TO JEFE;

-- Añadimos privilegios de sistema al rol
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW TO JEFE;

-- Eliminación de rol
DROP ROLE JEFE;
```

Asignación de roles a los usuarios

Se pueden asignar roles a un usuario e incluso a otro rol. La sintaxis es:

```
GRANT rol1 [,rol2 [,...]]
TO {usuario|rol [, {usuario|rol } [,...]] }
[WITH ADMIN OPTION];
```

Al igual que en las instrucciones anteriores, WITH ADMIN OPTION permite al usuario al que se le concede el rol, conceder él dicho rol a otros usuarios/as.

Roles predefinidos

Existen 3 roles predefinidos en Oracle:

Tabla 3: ROLES PREDEFINIDOS

Rol	Descripción
CONNECT	Permite al usuario conectarse a la base de datos. Debemos conceder este rol a cualquier usuario o aplicación que necesite acceso a la base de datos.
RESOURCE	Permite a un usuario crear, modificar y eliminar ciertos tipos de objetos de esquema en el esquema asociado con ese usuario. Debemos conceder este rol sólo a los desarrolladores y a otros usuarios que deben crear objetos de esquema. Esta función otorga un subconjunto de los privilegios del sistema de objetos de creación. Por ejemplo, concede el privilegio del sistema CREATE TABLE, pero no otorga el privilegio del sistema CREATE VIEW . Sólo otorga los siguientes privilegios: CREATE CLUSTER, CREATE INDEX TYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER y CREATE TYPE.
DBA	Permite al usuario realizar la mayoría de las funciones administrativas, incluyendo la creación de usuarios y la concesión de privilegios; Crear y otorgar roles; Crear, modificar y eliminar objetos de esquema en cualquier esquema; y más. Concede todos los privilegios del sistema, pero no incluye los privilegios para iniciar o cerrar la instancia de la base de datos. Esto se concede por defecto a los usuarios SYS y SYSTEM.

Consultar roles

Para ver los roles en activo para el usuario y sesión actuales ejecutamos la sentencia:

```
SELECT * FROM SESSION_ROLES;
```

6.3 COPIAS DE SEGURIDAD

La realización de copias de seguridad de los datos es una tarea imprescindible y altamente recomendable que todo administrador de bases de datos debe conocer y realizar con asiduidad. A continuación comentamos como se realizan para el SGBD Oracle.

6.3.1 Tipos de copias de seguridad

Según la forma de realizar la copia de seguridad se distingue 2 tipos:

- **Copia lógica:** Se copia el contenido de la BD pero sin almacenar la posición física de los datos.
- **Copia física:** Se copian físicamente los datos. A su vez, la copia física puede ser:
 - **Copia en frío:** también llamada copia off-line. Hay que parar previamente la BD y las aplicaciones que estén trabajando con la BD. Una vez realizada la copia de los ficheros, la BD se puede volver a arrancar.
 - **Copia en caliente:** también llamada copia on-line. Se realiza mientras la BD está abierta y funcionando en modo ARCHIVELOG. Se realiza cuando se requiere disponibilidad de los datos 24h al día, 7 días a la semana. Habrá que tener cuidado de realizarlo cuando la carga de la BD sea pequeña.

6.3.2 Herramientas de Oracle

En Oracle podemos usar 2 herramientas para realizar copias de seguridad:

- **Data Pump (EXPDP e IMPDP):** permite realizar una **copia de seguridad «lógica»**, que normalmente se realiza al especificar tablas específicas. Si no realiza una exportación «coherente», o si no incluye las tablas relacionadas y utiliza el modo RESTRICTED y CONSISTENT (con restricciones de integridad referencial), es posible que no pueda recuperar correctamente. La exportación se utiliza a menudo como un complemento de RMAN, generalmente para la restauración de tablas específicas.
- **Recovery manager (RMAN):** RMAN está diseñado para copia de seguridad y recuperación, una extensión de Enterprise Backup Utility (EBU). RMAN realiza **copias de seguridad completas, físicas y consistentes** de los archivos de su base de datos.

Ventajas y desventajas de Data Pump:

- Gratis y fácil de usar.
- Muy lento, en comparación con RMAN (examina cada bloque de datos).
- Restauración fácil de una tabla específica.
- No requiere modo ARCHIVELOG.

Ventajas y desventajas de RMAN:

- RMAN tiene recuperación de medios a nivel de bloque.
- Tiene un catálogo para el seguimiento de respaldo y una utilidad de informe.
- Rápido: si dedica un dispositivo de copia de seguridad para cada disco de producción, puede hacer una copia de seguridad de los terabytes en el tiempo que lleva realizar una copia de seguridad de un solo disco.
- Hace copias de seguridad en caliente o en frío.
- Copias de seguridad y restauraciones se pueden hacer en paralelo.
- Permite copias de seguridad incrementales (seguimiento de cambio de bloque).
- Interfaces con sistemas de gestión de medios (TMS).

6.3.3 Copia lógica

Se realiza con **Data Pump**. Es una copia en caliente. Las aplicaciones usadas son:

- **EXPDP** para exportación.
- **IMPDP** para importación.

Pasos previos

Antes de crear las copias de seguridad es aconsejable tener configurado un directorio de destino donde se almacenarán dichas copias. A continuación se muestra cómo.

Primero debemos crear dicho directorio. Puede usarse las herramientas del sistema operativo. Por ejemplo en un terminal de texto de Windows, para crear una carpeta podemos hacer:

```
MKDIR C:\BACKUP-ORACLE
```

A continuación registramos dicho directorio en el SGBD de Oracle.

```

SQLPLUS /NOLOG
CONNECT / AS SYSDBA

-- Indicamos directorio donde almacenar los backups
-- Dicho directorio debe haberse creado previamente. Si no dará error.
CREATE DIRECTORY BACKUP AS 'C:\BACKUP-ORACLE';

-- Damos permisos de lectura y escritura sobre dicho directorio
-- al usuario deseado.
GRANT READ, WRITE ON DIRECTORY BACKUP TO usuario;

-- Podemos comprobar que se ha registrado correctamente el directorio.
SELECT DIRECTORY_NAME FROM DBA_DIRECTORIES;

-- Salimos del SGBD
EXIT

```

Después ya podemos usar las herramientas **EXPDP** e **IMPDP** desde el terminal de texto del sistema operativo. A continuación se muestra el formato simplificado de uso y algunos ejemplos.

Formato para exportación

```

expdp _usuario_/_clave_ dumpfile=_archivo_backup_ logfile=_archivo_log_ _
↳opciones_

```

Copia de la base de datos completa al directorio de backup por defecto

```

expdp SYSTEM/SYSTEM dumpfile=BD_COMPLETA.dmp logfile=BD_COMPLETA.log full=y

```

Copia del esquema EMPLEADOS al directorio de backup por defecto:

```

expdp SYSTEM/SYSTEM dumpfile=ESQUEMA-EMPLEADOS.dmp logfile=ESQUEMA-EMPLEADOS.log _
↳schemas=EMPLEADOS

```

Copia del esquema EMPLEADOS al directorio de backup definido:

```

expdp SYSTEM/SYSTEM directory=BACKUP dumpfile=ESQUEMA-EMPLEADOS.dmp _
↳logfile=ESQUEMA-EMPLEADOS.log schemas=EMPLEADOS

```

Restauración del esquema empleados con la copia del directorio de backup definido:

```

impdp SYSTEM/SYSTEM directory=BACKUP dumpfile=ESQUEMA-EMPLEADOS.dmp

```

Nota: Si no indicamos la opción `directory`, los backups se guardan por defecto en **C:\oracle\app\oracle\admin\XE\dpdump**, o similar, dependiendo del directorio donde se tenga instalado Oracle.

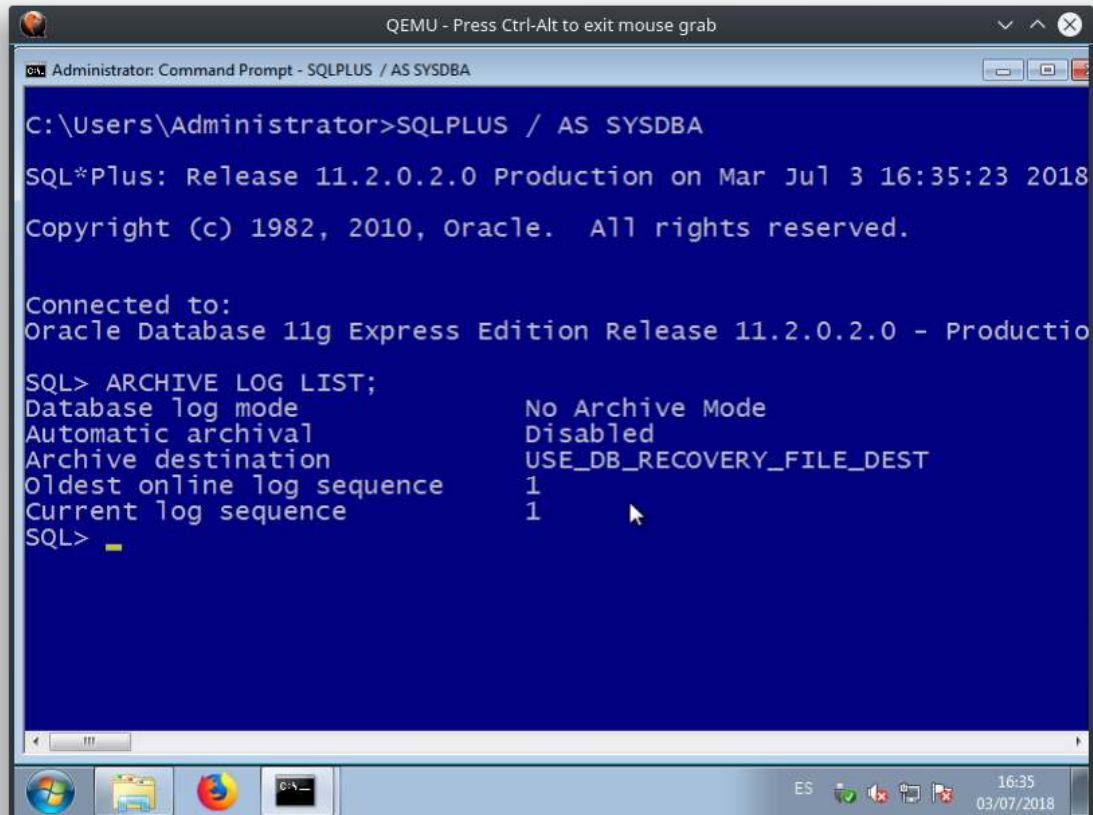
En los siguientes apartados se ve de forma muy resumida como realizar copias físicas de la base de datos con la aplicación de terminal *RMAN*. No se verá su uso detallado puesto que dispone de innumerables opciones. Por tanto se indicarán los comandos principales.

6.3.4 Copia física en caliente (en línea)

La copia física en caliente sólo es posible en el modo ARCHIVELOG.

Por defecto, la base de datos Oracle no está configurada para trabajar en este modo. Para ver si la base de datos está en el modo ARCHIVELOG o no, se puede usar en SQLPLUS el comando que se muestra a continuación:

```
SQL> ARCHIVE LOG LIST;
```



Podemos poner la base de datos en modo ARCHIVELOG con las sentencias:

```
SQL> shutdown immediate
SQL> startup mount
SQL> alter database archivelog;
      Database altered.

SQL> alter database open;
      Database altered.
```

Para realizar una copia de seguridad de la base de datos completa, desde el indicador RMAN, debemos usar el comando BACKUP DATABASE.

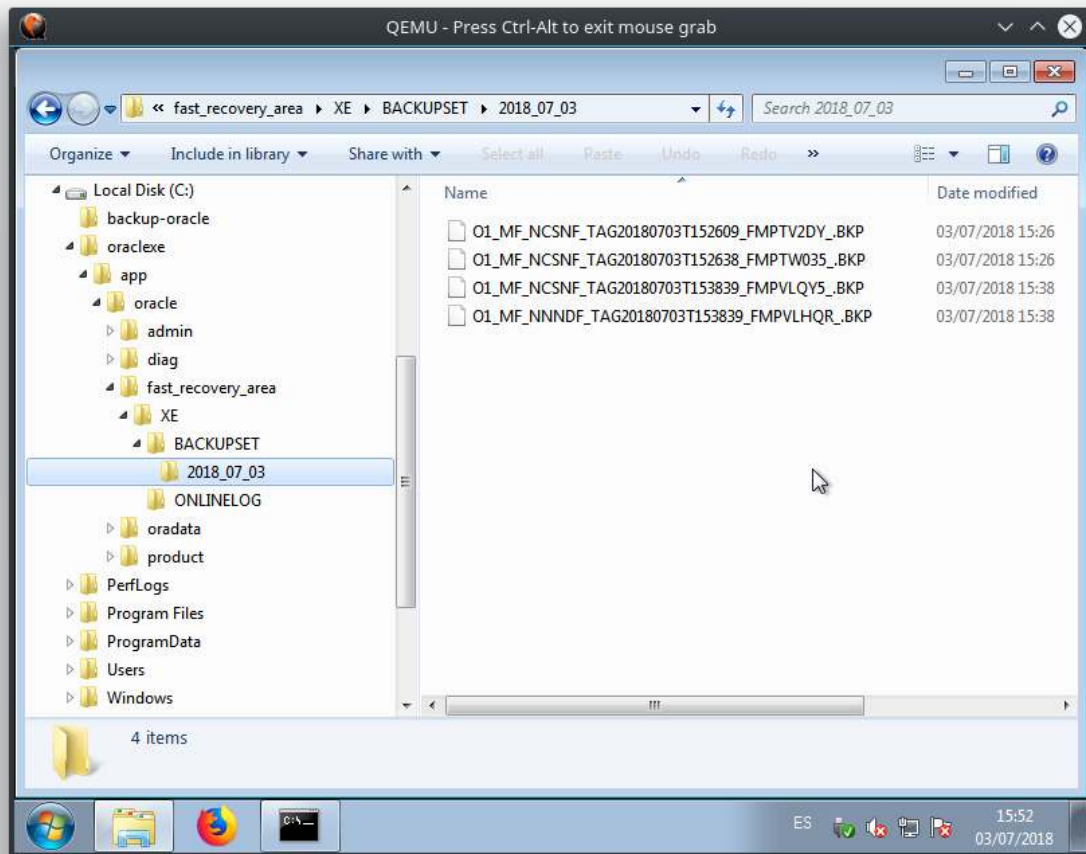
```
RMAN TARGET /

RMAN> BACKUP DATABASE;

RMAN> EXIT;
```

La copia de seguridad se guardará en **C:\oraclexe\app\oracle\fast_recovery_area\XE\backupset**, o similar, depen-

diendo del directorio donde se tenga instalado Oracle.



6.3.5 Copia física en frío (fuera de línea)

Si la base de datos está en modo NOARCHIVELOG, deberemos detener la base de datos, realizar la copia (``BACKUP``) y volver a abrir la base de datos.

Para crear copia de seguridad

```

RMAN TARGET /

RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP FORCE DBA;
RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;

RMAN> BACKUP DATABASE;

RMAN> ALTER DATABASE OPEN;

```

Para restaurar copia de seguridad

Para restaurar la copia debemos parar la base de datos, restaurar y recuperar (``RESTORE``, ``RECOVER``) y volver a abrir la BD.

```

RMAN TARGET /

RMAN> SHUTDOWN IMMEDIATE;
RMAN> STARTUP MOUNT;

RMAN> RESTORE DATABASE PREVIEW;
RMAN> RESTORE DATABASE;
RMAN> RECOVER DATABASE;

RMAN> ALTER DATABASE OPEN RESETLOGS;
    
```

QEMU - Press Ctrl-Alt to exit mouse grab

Administrator: Command Prompt - RMAN TARGET /

```

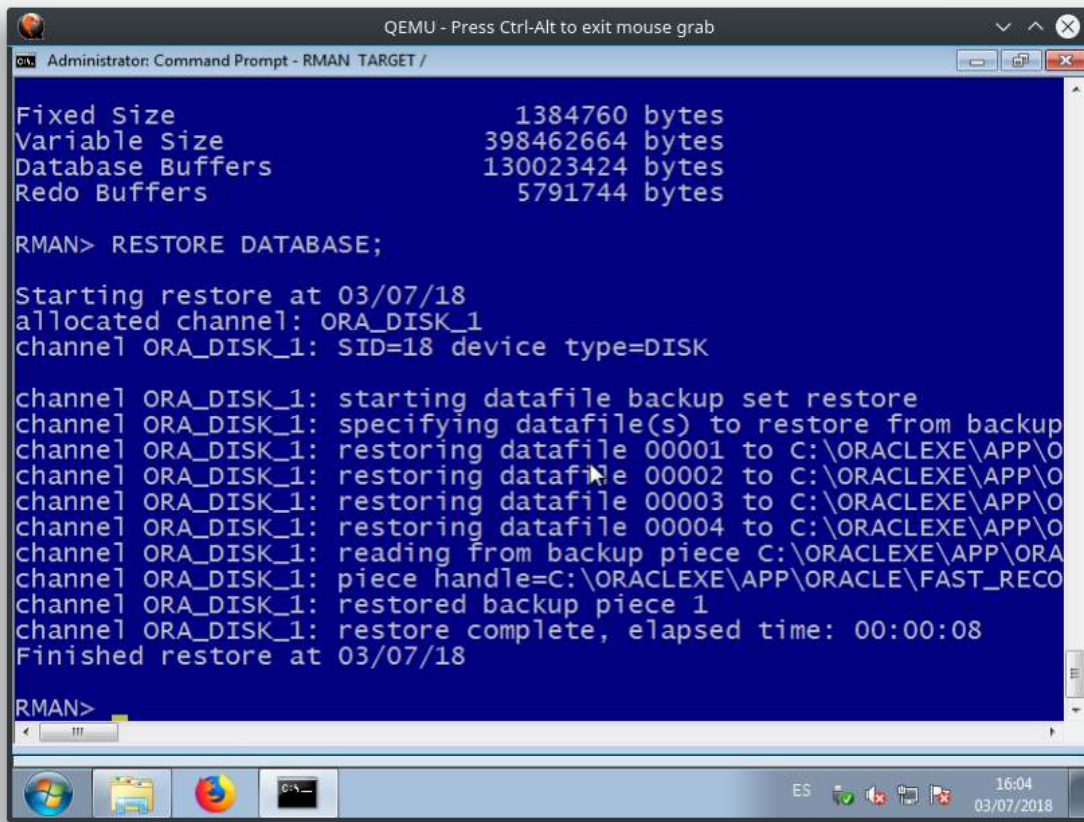
RMAN> RESTORE DATABASE PREVIEW;

Starting restore at 03/07/18
using channel ORA_DISK_1

List of Backup Sets
=====

BS Key   Type LV Size       Device Type Elapsed Time Completion Time
-----
3        Full  826.62M    DISK        00:00:06    03/07/18
BP Key: 3  Status: AVAILABLE Compressed: NO Tag: TAG20
Piece Name: C:\ORACLE\APP\ORACLE\FAST_RECOVERY_AREA\XE\
List of Datafiles in backup set 3
File LV Type Ckp SCN      Ckp Time Name
-----
1        Full 1389636    03/07/18 C:\ORACLE\APP\ORACLE\ORADATA
2        Full 1389636    03/07/18 C:\ORACLE\APP\ORACLE\ORADATA
3        Full 1389636    03/07/18 C:\ORACLE\APP\ORACLE\ORADATA
4        Full 1389636    03/07/18 C:\ORACLE\APP\ORACLE\ORADATA
Media recovery start SCN is 1389636
Recovery must be done beyond SCN 1389636 to clear datafile fuzziness
Finished restore at 03/07/18
    
```

ES 15:54 03/07/2018



The screenshot shows a QEMU window titled "QEMU - Press Ctrl-Alt to exit mouse grab". Inside is a Windows XP-style desktop with a taskbar. The taskbar includes icons for Start, My Computer, Internet Explorer, and a command prompt. The system tray shows the date and time as "16:04 03/07/2018". The command prompt window is titled "Administrator: Command Prompt - RMAN TARGET /" and has a blue background. It displays the following text:

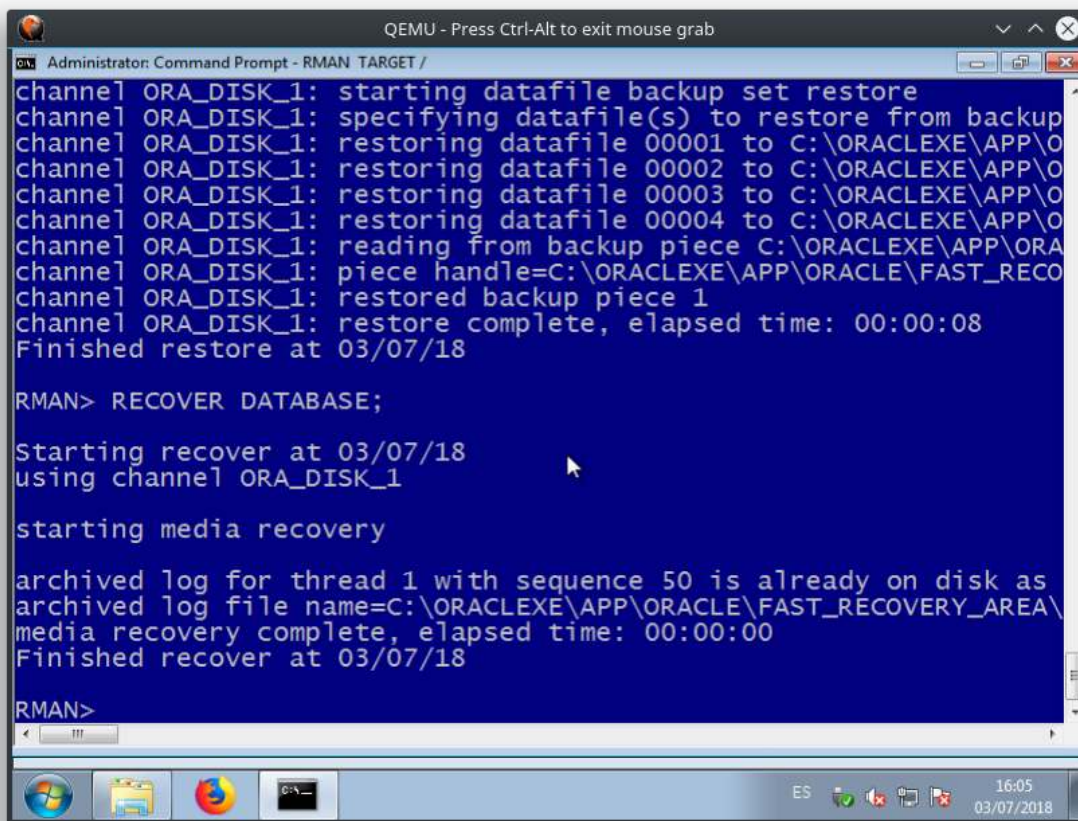
```
Fixed Size                1384760 bytes
Variable Size             398462664 bytes
Database Buffers          130023424 bytes
Redo Buffers               5791744 bytes

RMAN> RESTORE DATABASE;

Starting restore at 03/07/18
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=18 device type=DISK

channel ORA_DISK_1: starting datafile backup set restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup
channel ORA_DISK_1: restoring datafile 00001 to C:\ORACLEXE\APP\O
channel ORA_DISK_1: restoring datafile 00002 to C:\ORACLEXE\APP\O
channel ORA_DISK_1: restoring datafile 00003 to C:\ORACLEXE\APP\O
channel ORA_DISK_1: restoring datafile 00004 to C:\ORACLEXE\APP\O
channel ORA_DISK_1: reading from backup piece C:\ORACLEXE\APP\ORA
channel ORA_DISK_1: piece handle=C:\ORACLEXE\APP\ORACLE\FAST_RECO
channel ORA_DISK_1: restored backup piece 1
channel ORA_DISK_1: restore complete, elapsed time: 00:00:08
Finished restore at 03/07/18

RMAN>
```



```
QEMU - Press Ctrl-Alt to exit mouse grab
Administrator: Command Prompt - RMAN TARGET /
channel ORA_DISK_1: starting datafile backup set restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup
channel ORA_DISK_1: restoring datafile 00001 to C:\ORACLE\APP\O
channel ORA_DISK_1: restoring datafile 00002 to C:\ORACLE\APP\O
channel ORA_DISK_1: restoring datafile 00003 to C:\ORACLE\APP\O
channel ORA_DISK_1: restoring datafile 00004 to C:\ORACLE\APP\O
channel ORA_DISK_1: reading from backup piece C:\ORACLE\APP\ORA
channel ORA_DISK_1: piece handle=C:\ORACLE\APP\ORACLE\FAST_RECO
channel ORA_DISK_1: restored backup piece 1
channel ORA_DISK_1: restore complete, elapsed time: 00:00:08
Finished restore at 03/07/18

RMAN> RECOVER DATABASE;

Starting recover at 03/07/18
using channel ORA_DISK_1

starting media recovery

archived log for thread 1 with sequence 50 is already on disk as
archived log file name=C:\ORACLE\APP\ORACLE\FAST_RECOVERY_AREA\
media recovery complete, elapsed time: 00:00:00
Finished recover at 03/07/18

RMAN>
```

Enlaces de interés

- <http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=560>
- <https://oracle-base.com/articles/10g/oracle-data-pump-10g>
- https://docs.oracle.com/cd/B28359_01/backup.111/b28270/rcmquick.htm#BRADV90058

6.4 IMPORTACIÓN Y EXPORTACIÓN DE DATOS

En este apartado comentaremos brevemente como importar y exportar datos en Oracle, aunque en muchos casos es aplicable a otros sistemas gestores de BB.DD. puesto que la forma de proceder suele ser parecida.

Antes de empezar, debemos diferenciar entre dos conceptos:

- **Migración:** mover de un sitio a otro datos y esquemas.
- **Importación/Exportación:** mover de un sitio a otro datos únicamente.

6.4.1 Migración

La migración es el proceso de pasar una base de datos desarrollada bajo un SGBD a otro SGBD distinto. Por ejemplo pasar una base de datos Oracle a MySQL o SQL Server, por poner sólo dos ejemplos.

Los principales aspectos a tener en cuenta, y que pueden ser problemáticos son:

- **La definición e interpretación de los distintos tipos de datos** Deberán realizarse cambios en algunos tipos de datos que existen en una base de datos pero no en otras. Especialmente delicados son los campos fecha, los numéricos (enteros, reales, etc) y los de tipo texto variable, entre otros, ya que cada SGBD los trata o los «espera» de manera diferente.
- **El código procedimental almacenado** Procedimientos, funciones, paquetes y triggers se escriben en un lenguaje específico para cada SGBD. Por ejemplo Oracle lo hace en PL/SQL, mientras SQL Server lo hace en Transact-SQL. Asimismo MySQL posee también su propio lenguaje procedimental.

Actualmente la mayoría de SGBD incluyen herramientas de ayuda a la migración más o menos «fiables» que permiten realizar este proceso de manera automatizada en gran medida.

Otra forma de realizar la migración es hacerlo de forma manual, mediante la revisión y adaptación del código, tanto declarativo (SQL) como procedimental (PL/SQL, Transact-SQL, ...), así como la revisión y formateo de datos.

No obstante, ni que decir tiene que el proceso de migración de datos es lo suficientemente delicado como para realizarlo en un entorno de pruebas, contemplando toda la casuística posible en cuanto a tipos de datos a manejar, tablas involucradas y sus relaciones, etc. Sólo en el momento en el que estemos seguros de que la migración se ha realizado con éxito, sin problemas de interpretación de datos ni pérdida de ellos, podemos pasar a un entorno de producción, teniendo en cuenta que una migración mal realizada podría dar por terminada una estructura de información completa.

La migración es un proceso complejo y que debería evitarse siempre que sea posible. Para ello es importante decidir correctamente en un principio el SGBD a utilizar y mantenerlo durante toda la vida útil de la base de datos.

La importación/exportación de datos es un proceso relativamente sencillo, si la comparamos con la migración. Básicamente, el aspecto más importante a tener en cuenta es el formato de datos que se utilizará para el intercambio de datos.

6.4.2 Importación/Exportación

Tanto en la importación como la exportación es bastante frecuente utilizar texto plano para realizar el intercambio de información. Los formatos más usados son:

- CSV
- XML
- JSON
- SQL

CSV (Comma Separate Values)

Es un formato extremadamente sencillo pero muy potente.

Los archivos CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma en donde la coma es el separador decimal: Argentina, España, Brasil...) y las filas por saltos de línea.

El formato CSV es muy sencillo y no indica un juego de caracteres concreto, ni cómo van situados los bytes, ni el formato para el salto de línea. Estos puntos deben indicarse muchas veces al abrir el archivo, por ejemplo, con una hoja de cálculo.

El formato CSV no está estandarizado. La idea básica de separar los campos con una coma es muy clara, pero se vuelve complicada cuando el valor del campo también contienen comillas dobles o saltos de línea. Las implementaciones de CSV pueden no manejar esos datos, o usar comillas de otra clase para envolver el campo. Pero esto no resuelve el problema: algunos campos también necesitan embeber estas comillas, así que las implementaciones de CSV pueden incluir caracteres o secuencias de escape.

Además, el término «CSV» también denota otros formatos de valores separados por delimitadores que usan delimitadores diferentes a la coma (como los valores separados por tabuladores).

Ejemplo en forma de tabla:

NUMCE	NOMCE	DIRCE
10	SEDE CENTRAL	C/ ATOCHA, 820, MADRID
20	RELACIÓN CON CLIENTES	C/ ATOCHA, 405, MADRID

Datos en formato CSV:

```
NUMCE,NOMCE,DIRCE
10,SEDE CENTRAL,"C/ ATOCHA, 820, MADRID"
20,RELACIÓN CON CLIENTES,"C/ ATOCHA, 405, MADRID"
```

Una de las grandes ventajas del uso de este formato es que puede editarse utilizando cualquier editor de texto. Además está muy bien soportado por cualquier hoja de cálculo, por ejemplo puede abrirse con LibreOffice Calc y nos mostrará su contenido en forma de tabla.

Importación de texto - [centros.csv]

Importar

Conjunto de caracteres: Unicode (UTF-8)

Idioma: Predeterminado - Español (España)

Desde la fila: 1

Opciones de separador

☐ Anchura fija ☒ Separado por

☒ Tabulador ☒ Coma ☒ Punto y coma ☐ Espacio ☐ Otro

☐ Fusionar los delimitadores

Delimitador de texto: "

Otras opciones

☐ Campo entrecomillado como texto ☐ Detectar los números especiales

Campos

Tipo de columna:

	Predeterm	Predeterminado	Predeterminado
1	NUMCE	NOMCE	DIRCE
2	10	SEDE CENTRAL	C/ ATOCHA, 820, MADRID
3	20	RELACIÓN CON CLIENTES	C/ ATOCHA, 405, MADRID

Ayuda Aceptar Cancelar

Esto permite aplicar cualquier tipo de operación disponible en una hoja de cálculo: ordenar por columnas, realizar cálculos, etc y finalmente volver a guardar el resultado en formato CSV.

XML (eXtensible Markup Language)

XML, siglas en inglés de eXtensible Markup Language, traducido como «Lenguaje de Marcado Extensible» o «Lenguaje de Marcas Extensible», es un meta-lenguaje que permite definir lenguajes de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

XML no ha nacido únicamente para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande, con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

JSON (JavaScript Object Notation)

JSON, acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. Si bien es frecuente ver JSON posicionado contra XML, también es frecuente el uso de JSON y XML en la misma aplicación.

XML goza de mayor soporte y ofrece muchas más herramientas de desarrollo (tanto en el lado del cliente como en el lado del servidor). Hay muchos analizadores JSON en el lado del servidor, existiendo al menos un analizador para la mayoría de los entornos. En algunos lenguajes, como Java o PHP, hay diferentes implementaciones donde escoger. En JavaScript, el análisis es posible de manera nativa con la función `eval()`. Ambos formatos carecen de un mecanismo para representar grandes objetos binarios.

Actualmente JSON tiene su principal nicho en aplicaciones JavaScript y en algunas bases de datos noSQL.

SQL (Structured Query Language)

Por último, tenemos el lenguaje SQL, soportado por todas las bases de datos relacionales.

Podemos hacer uso de scripts SQL con sentencias INSERT para realizar la “importación” de datos dentro de un SGBDR. Si bien en este caso es necesario el procesamiento previo de las sentencias, algo que no sucedía con los formatos anteriores.

Ejemplo de script SQL para importación de datos:

```
INSERT INTO DEPARTAMENTOS VALUES (100, 10, 260, 'P', 72, NULL, 'DIRECCIÓN GENERAL');
INSERT INTO DEPARTAMENTOS VALUES (110, 20, 180, 'P', 90, 100, 'DIRECC.COMERCIAL');
INSERT INTO DEPARTAMENTOS VALUES (111, 20, 180, 'F', 66, 110, 'SECTOR INDUSTRIAL');
INSERT INTO DEPARTAMENTOS VALUES (112, 20, 270, 'P', 54, 110, 'SECTOR SERVICIOS');
INSERT INTO DEPARTAMENTOS VALUES (120, 10, 150, 'F', 18, 100, 'ORGANIZACIÓN');
INSERT INTO DEPARTAMENTOS VALUES (121, 10, 150, 'P', 12, 120, 'PERSONAL');
INSERT INTO DEPARTAMENTOS VALUES (122, 10, 350, 'P', 36, 120, 'PROCESO DE DATOS');
INSERT INTO DEPARTAMENTOS VALUES (130, 10, 310, 'P', 12, 100, 'FINANZAS');

COMMIT;
```

6.5 ACTIVIDADES PROPUESTAS

6.5.1 Práctica 1

Realiza la instalación de APEX 5.1.

6.5.2 Práctica 2

Realiza la configuración de APEX 5.1 poniendo el idioma en español.

6.5.3 Práctica 3

Un usuario puede dar privilegios a otro. No es necesario ser sysdba para conseguirlo: el usuario propietario del esquema otorga los privilegios deseados al otro usuario. La única necesidad, y esa sí es importante, es que el usuario del esquema que se va a compartir, debe tener suficientes privilegios para otorgarlos.

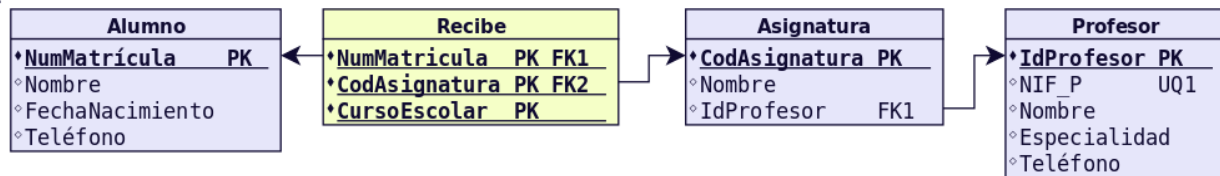
Veamos el código (debe ejecutarlo el usuario propietario del esquema que se desea compartir):

```
BEGIN
  FOR tabla IN (SELECT * FROM USER_TABLES) LOOP
    EXECUTE IMMEDIATE
      'GRANT SELECT ON ' || tabla.table_name || ' TO OTRO_USUARIO';
  END LOOP;
END;
/
```

Lo que este código hace es, mediante un bucle, recorre todas las tablas del usuario que comparte su esquema y otorga los privilegios seleccionados (en este caso solo SELECT) a cada tabla para el usuario OTRO_USUARIO (sustituir por el nombre del usuario al que deseamos otorgar los privilegios).

Realiza las siguientes actividades:

Tenemos un esquema (usuario) llamando E01 con 4 tablas y queremos conceder privilegios al esquema (usuario) llamado EMPLEADOS sobre las tablas anteriores.



Para ello procedemos de la siguiente forma:

1. Accedemos como usuario E01 y concedemos privilegios al usuario EMPLEADOS.

```
CONN E01/E01
GRANT ALL ON ALUMNO TO EMPLEADOS;
GRANT ALL ON PROFESOR TO EMPLEADOS;
GRANT ALL ON ASIGNATURA TO EMPLEADOS;
GRANT ALL ON RECIBE TO EMPLEADOS;
```

2. Accedemos como usuario EMPLEADOS y comprobamos que podemos trabajar con las tablas anteriores.

```
CONN EMPLEADOS/EMPLEADOS

DESCRIBE E01.ALUMNO;
DESCRIBE E01.PROFESOR;

SELECT * FROM E01.ASIGNATURA;

INSERT INTO E01.ALUMNO VALUES (1, 'JOSE', '01/01/1990', '600111222');
```

El usuario EMPLEADOS puede incluso crear vistas y sinónimos sobre las tablas anteriores. Para ello, el administrador de la base de datos, deberá concederle dichos privilegios.

```
CONN SYSTEM/SYSTEM

GRANT CREATE VIEW, CREATE SYNONYM TO EMPLEADOS;
```

Ahora EMPLEADOS puede crear una vista sobre las tablas anteriores.

```
CONN EMPLEADOS/EMPLEADOS

CREATE VIEW PROF_ASIG (CODPROF, NOMPROF, ESPECIALIDAD, CODASIG, NOMASIG)
AS
SELECT P.IDPROFESOR, P.NOMBRE, P.ESPECIALIDAD, A.CODASIGNATURA, A.NOMBRE
FROM E01.PROFESOR P JOIN E01.ASIGNATURA A ON P.IDPROFESOR=A.IDPROFESOR;
```

Para no tener que escribir E01.PROFESOR y E01.ASIGNATURA cada vez, podemos crear sinónimos. Por ejemplo:

```
CREATE SYNONYM PROF FOR E01.PROFESOR;
CREATE SYNONYM ASIG FOR E01.ASIGNATURA;
```

A partir de ahora podemos acceder a las tablas E01.PROFESOR y E01.ASIGNATURA a través de los sinónimos PROF y ASIG. Por ejemplo:

```
SELECT * FROM PROF;
```

6.5.4 Práctica 4

Inicia sesión como usuario EMPLEADOS. Da privilegios al usuario E01 para que pueda acceder a todas las tablas. Guíate por los pasos seguidos en la práctica anterior.

6.5.5 Práctica 5

Inicia sesión como usuario SYSTEM. Concede privilegios de *CREATE VIEW* y *CREATE SYNONYM* a E01.

6.5.6 Práctica 6

Inicia sesión como usuario E01. Crea un sinónimo corto para cada tabla del esquema EMPLEADOS. Crea una vista que contenga toda la información de las tablas del esquema EMPLEADOS. Deberás utilizar un JOIN o NATURAL JOIN.

6.5.7 Práctica 7

En el modo ARCHIVELOG, con la herramienta RMAN, haz una copia de seguridad de toda la base de datos de Oracle. A continuación realiza su restauración. Documenta todo el proceso.

6.5.8 Práctica 8

En el modo NOARCHIVELOG, con la herramienta RMAN, haz una copia de seguridad de toda la base de datos de Oracle. A continuación realiza su restauración. Documenta todo el proceso.

6.5.9 Práctica 9

Borra el contenido de las tablas EMPLEADOS, DEPARTAMENTOS y CENTROS del esquema EMPLEADOS. Importa los datos de estas tablas desde los archivos empleados.csv, departamentos.csv y centros.csv, disponibles en la plataforma Moodle.

6.5.10 Práctica 10

Exporta el contenido de las tablas EMPLEADOS, DEPARTAMENTOS y CENTROS del esquema EMPLEADOS a archivos xml. Busca un programa que al abrir dichos archivos los muestre en forma de tabla.

- genindex