C.F.G.S. DESARROLLO DE APLICACIONES WEB

DISTANCIA

MÓDULO:

BASES DE DATOS

Unidad 3:

Bases de Datos Relacionales.

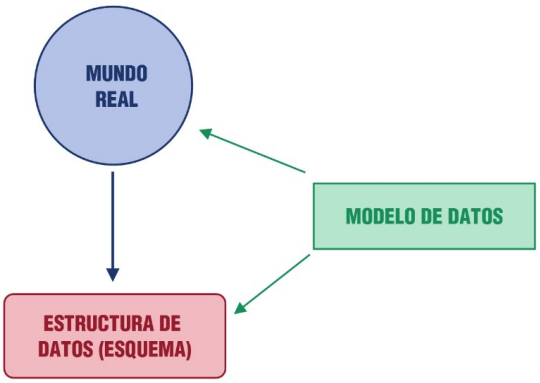
ÍNDICE DE CONTENIDOS

OBJETIVOS

- Analizar el formato de almacenamiento de la información.
- Crear tablas y relaciones entre las mismas.
- Seleccionar los tipos de datos adecuados.
- Definir los campos clave (primaria y ajena) en las tablas.
- Implementar de las restricciones reflejadas en el diseño lógico.

1.- Modelo de datos.

En informática, **un modelo de datos** es **un lenguaje utilizado para la descripción de una base de datos.** Con este lenguaje vamos a poder describir las **estructuras** de los datos (tipos de datos y relaciones entre ellos), **las restricciones de integridad** (condiciones que deben cumplir los datos, según las necesidades de nuestro modelo basado en la realidad) y **las operaciones de manipulación** de los datos (insertado, borrado, modificación de datos).



Es importante distinguir entre modelo de datos y esquema.

> Según Dittrich (1994):
> La descripción específica de un determinado mini-mundo en términos de un modelo de datos se denomina esquema (o esquema de datos) del mini-mundo. La colección de datos que representan la información acerca del mini-mundo constituye la base de datos.

Para clasificar los modelos debemos pensar en el nivel de abstracción, es decir, en lo alejado que esté del mundo real:

- **Los modelos de datos conceptuales** son aquellos que describen las estructuras de datos y restricciones de integridad. Se utilizan durante la etapa de análisis de un problema dado, y están orientados a representar los elementos que intervienen y sus relaciones. Ejemplo, Modelo Entidad-Relación.
- **Los modelos de datos lógicos** se centran en las operaciones y se implementan en algún sistema gestor de base de datos. Ejemplo, Modelo Relacional.
- **Los modelos de datos físicos** son estructuras de datos a bajo nivel, implementadas dentro del propio sistema gestor de base de datos.

Hemos dicho que un modelo de datos es un lenguaje y por lo general, presenta los siguientes sublenguajes:

- ✓ **Lenguaje de Definición de Datos o. LDD** cuya función es describir, de una forma abstracta, las estructuras de datos y las restricciones de integridad.
- ✓ **Lenguaje de Manipulación de Datos o LMD** que sirven para describir las operaciones de manipulación de los datos.
- ✓ **Lenguaje de Control de Datos o DCL** incluye una serie de comandos que permiten al administrador controlar el acceso a los datos contenidos en la Base de Datos.

2.- Terminología del modelo relacional.

El modelo relacional nos permite representar la información del mundo real de una manera intuitiva, pudiendo introducir conceptos cotidianos y fáciles de entender por cualquiera, aunque no sea experto en informática.

El modelo relacional fue propuesto por Edgar Frank Codd en los laboratorios de IBM en California. Se trata de un modelo lógico que establece una estructura sobre los datos, independientemente del modo en que luego los almacenemos.

El nombre de modelo relacional viene de la estrecha relación entre el elemento básico de este modelo y el concepto matemático de relación.

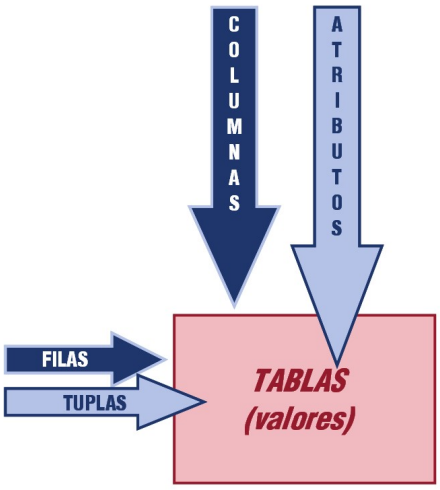
2.1.- Relación o tabla. Tuplas. Dominio.

A partir de ahora, nosotros veremos una relación como una **tabla** con **filas** y **columnas**. Podemos asociar **atributos a columna** y **tuplas a filas**.

- **Atributos**: Es el **nombre de cada dato** que se almacena en la relación (tabla). Ejemplos serían: DNI, nombre, apellidos, etc.
  El nombre del atributo debe describir el significado de la información que representa. En la tabla Empleados, el atributo Sueldo almacenará el valor en euros del sueldo que recibe cada empleado. A veces es necesario añadir una pequeña descripción para aclarar un poco más el contenido. Por ejemplo, si el sueldo es neto o bruto.

- **Tuplas**: Se refiere a **cada elemento de la relación**. Si una tabla guarda datos de un cliente, como su DNI o Nombre, una tupla o registro sería ese DNI y nombre concreto de un cliente.
  Cada una de las filas de la tabla se corresponde con la idea de registro y tiene que cumplirse que:
  - ✓ Cada tupla se debe corresponder con un elemento del mundo real.
  - ✓ No puede haber dos tuplas iguales (con todos los valores iguales).

Un atributo en una tupla no puede tomar cualquier valor. No sería lógico que en un atributo Población se guarde "250€". Estaríamos cometiendo un error, para evitar este tipo de situaciones obligaremos a que cada atributo sólo pueda tomar los valores pertenecientes a un conjunto de valores previamente establecidos, es decir, **un atributo tiene asociado un dominio** de valores.



A menudo un dominio se define a través de la declaración de un tipo para el atributo (por ejemplo, diciendo que es un número entero entre 1 y 16), pero también se pueden definir dominios más complejos y precisos. Por ejemplo, para el atributo Sexo de mis usuarios, podemos definir un dominio en el que los valores posibles sean "M" o "F" (masculino o femenino).

Una característica fundamental de los dominios es que sean atómicos, es decir, que los valores contenidos en los atributos no se pueden separar en valores de dominios más simples.

Un dominio debe tener:

- ✓ Nombre
- ✓ Definición lógica
- ✓ Tipo de datos
- ✓ Formato.

Por ejemplo, si consideramos el Sueldo de un empleado, tendremos:

- ✓ Nombre: Sueldo.
- ✓ Definición lógica: Sueldo neto del empleado
- ✓ Tipo de datos: número entero.
- ✓ Formato: 9.999€.

2.2.- Grado. Cardinalidad.

Hemos visto que una relación es una tabla con filas y columnas. Pero ¿hasta cuántas columnas puede contener? ¿Cuántos atributos podemos guardar en una tabla?

Llamaremos **grado** al **tamaño** de una tabla en base a **su número de atributos** (columnas). Mientras mayor sea el grado, mayor será su complejidad para trabajar con ella.

¿Y cuántas tuplas (filas o registros) puede tener?

Llamaremos **cardinalidad** al **número de tuplas** o filas de una relación o tabla.

2.3.- Sinónimos.

Los términos vistos hasta ahora tienen distintos sinónimos según la nomenclatura utilizada. Trabajaremos con tres:

- En el **modelo relacional**: Relación - Tupla - Atributo - Grado - Cardinalidad.
- En **tablas**: Tabla - Fila - Columnas - Número columnas - Número filas.
- En términos de **registros**: Ficheros - Registros - Campos - Número campos - Número registros.

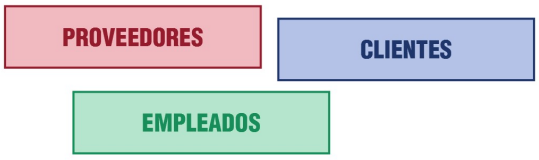
Nomenclatura relacional	Nomenclatura tabla	Nomenclatura ficheros
relación =	tabla =	fichero
tupla =	fila =	registros
atributo =	columna =	campos
grado =	nº columnas =	nº campos
cardinalidad =	nº filas =	nº registros

3.- Relaciones. Características de una relación (Tabla).

¿En un modelo relacional se puede utilizar cualquier relación? ¿Es válida cualquier tabla o se deben cumplir algunas propiedades?

Debes saber que:

- Cada **tabla** tiene un **nombre distinto**.



- Cada **atributo** (columna) de la tabla toma **un solo valor en cada tupla** (fila).
- Cada **atributo** (columna) tiene un nombre distinto en cada tabla (pero puede ser el mismo en tablas distintas).
- **No puede** haber dos **tuplas** (filas) **completamente iguales**

Carlos	Matemáticas	Aprobado
Carlos	Matemáticas	Suspenso
Carlos	Lengua	Aprobado
~~Carlos~~	~~Lengua~~	~~Aprobado~~

- El **orden de las tuplas** (filas) no importa.

Nombre	Edad
Carlos	18
Elena	23
María	15

Es equivalente a

Nombre	Edad
María	15
Carlos	18
Elena	23

- El **orden de los atributos** (columnas) no importa

Nombre	Edad
Carlos	18
Elena	23
María	15

Es equivalente a

Edad	Nombre
18	Carlo
23	Elena
15	María

- Todos los **datos de un atributo** (columna) deben ser **del mismo dominio**.

Nombre	Asignatura	Nota
Carlos	Matemáticas	Aprobado
Elena	Física	Notable
María	Matemáticas	7

3.1.- Tipos de relaciones (tablas)

Existen varios tipos de relaciones y las vamos a clasificar en:

- **Persistentes**:
  Sólo pueden ser borradas por los usuarios.
  - **Base**: Independientes, se crean indicando su estructura y sus ejemplares (conjunto de tuplas o filas).
  - **Vistas**: Son tablas que sólo almacenan una definición de consulta, resultado de la cual se produce una tabla cuyos datos proceden de las bases o de otras vistas e instantáneas. Si los datos de las tablas base cambian, los de la vista que utilizan esos datos también cambiarán.
  - **Instantáneas**: Son vistas (se crean de la misma forma) que sí almacenan los datos que muestran, además de la consulta que la creó. Solo modifican su resultado cuando el sistema se refresca cada cierto tiempo. Es como una fotografía de la relación, que sólo es válida durante un periodo de tiempo concreto.

- **Temporales**:
  Son tablas que son eliminadas automáticamente por el sistema.

4.- Tipos de Datos.

Hasta ahora hemos visto que vamos a guardar información relacionada en forma de filas y columnas. Las columnas son los atributos o información que nos interesa incluir del mundo real que estamos modelando.

Hemos visto que esos atributos se mueven dentro de un dominio, que formalmente es un conjunto de valores. Pues bien, en términos de sistemas de base de datos, se habla más de tipos de datos que de dominios. Al crear la relación (tabla) decidimos qué conjunto de datos deberá ser almacenado en las filas de los atributos que hemos considerado. Tenemos que asignar un tipo de dato a cada atributo.

Con la asignación de tipos de datos, también habremos seleccionado un dominio para un atributo.

Cada campo debe tener:

- ✓ Un Nombre (relacionado con los datos que va a contener)
- ✓ Un Tipo de dato asociado.

Existen distintas formas de nombrar los tipos de datos dependiendo del lenguaje que utilicemos. Los más comunes con los que nos encontraremos generalmente:

- **Texto**: almacena cadenas de caracteres (números con los que no vamos a realizar operaciones matemáticas, letras o símbolos).
- **Numérico**: almacena números con los que vamos a realizar operaciones matemáticas.
- **Fecha/hora**: almacena fechas y horas.
- **Sí/No**: almacena datos que solo tienen dos posibilidades (verdadero/falso).
- **Autonumérico**: valor numérico secuencial que el SGBD incrementa de modo automático al añadir un registro (fila).
- **Memo**: almacena texto largo (mayor que un tipo texto).
- **Moneda**: se puede considerar un subtipo de Numérico ya que almacena números, pero con una característica especial, y es que los valores representan cantidades de dinero.
- **Objeto OLE**: almacena gráficos, imágenes o textos creados por otras aplicaciones.

5.- Claves.

¿Cómo diferenciamos unos usuarios de otros? ¿Cómo sabemos que no estamos recogiendo la misma información? ¿Cómo vamos a distinguir unas tuplas de otras? Lo haremos mediante los valores de sus atributos. Para ello, buscaremos un atributo o un conjunto de atributos que identifiquen de modo único las tuplas (filas) de una relación (tabla). A ese atributo o conjunto de atributos lo llamaremos **superclaves**.

Hemos visto que una característica de las tablas era que no puede haber dos tuplas (filas) completamente iguales, con lo que podemos decir que toda la fila como conjunto sería una **superclave**.

Tendríamos que elegir alguna de las superclaves para diferenciar las tuplas. En el modelo relacional trabajamos con tres tipos de claves:

- ✓ Claves candidatas.
- ✓ Claves primarias o primary key
- ✓ Claves alternativas
- ✓ Claves ajenas o extranjera o foreign key.

5.1.- Clave candidata. Clave primaria. Clave alternativa.

Si puedo elegir entre tantas claves, ¿con cuál me quedo? Tendremos que elegir entre las claves "candidatas" la que mejor se adapte a mis necesidades. ¿Y cuáles son éstas? Las **claves candidatas** serán aquel conjunto de atributos que **identifiquen de manera única** cada tupla (fila) de la relación (tabla). Es decir, las columnas cuyos valores no se repiten en ninguna otra fila de la tabla. Por tanto, cada tabla debe tener al menos una clave candidata aunque puede haber más de una.

Por ejemplo: Si tenemos la tabla **DEPARTAMENTO**.

NUMEDPTO	NOMDPTO	PRESUPUESTO
D1	Marketing	60000
D2	Desarrollo	100000
D3	Investigación	30000

En esta tabla tenemos **NUMEDPTO**. y **NOMPPTO** que identifican de forma única cada tupa.
Por lo tanto son **claves candidatas**.

Las claves candidatas pueden estar formadas por más de un atributo, siempre y cuando éstos identifiquen de forma única a la fila. Cuando una **clave candidata está formada por más de un atributo**, se dice que es una **clave compuesta**.

Una clave candidata debe cumplir los siguientes requisitos:

- ✓ **Unicidad**: no puede haber dos tuplas (filas) con los mismos valores para esos atributos.
- ✓ **Irreducibilidad:** si se elimina alguno de los atributos deja de ser única.

Para identificar las claves candidatas de una relación no nos fijaremos en un momento concreto en el que vemos una base de datos. Puede ocurrir que en ese momento no haya duplicados para un atributo o conjunto de atributos, pero esto no garantiza que se puedan producir. El único modo de identificar las claves candidatas es conociendo el significado real de los atributos (campos), ya que así podremos saber si es posible que aparezcan duplicados. Es posible desechar claves como candidatas fijándonos en los posibles valores que podemos llegar a tener.

En nuestro ejemplo tenemos varias claves con la que identificamos de modo único nuestra relación. De ahí el nombre de candidatas. Hemos de quedarnos con una.

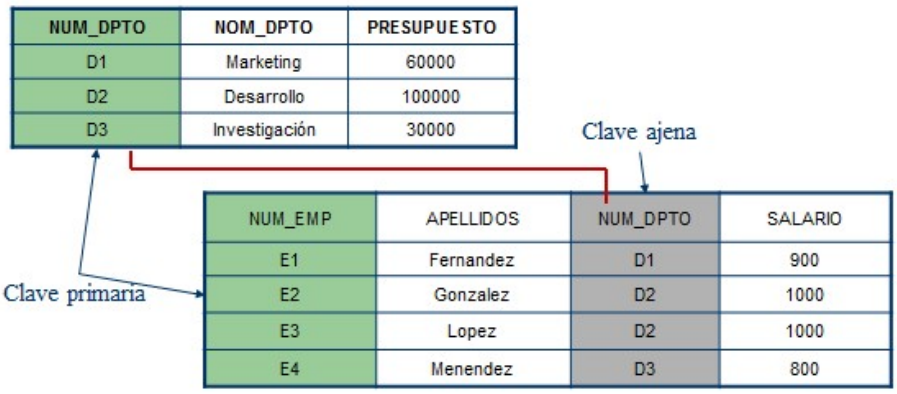
La **clave primaria** de una relación **es aquella clave candidata que se escoge** para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria. En el peor caso, la clave primaria estará formada por todos los atributos de la relación, pero normalmente habrá un pequeño subconjunto de los atributos que haga esta función. En otros casos, podemos crear un campo único que identifique las tuplas, por ejemplo un código de usuario, que podrían estar constituidos por valores autonuméricos.

Las claves candidatas que **no son escogidas** como clave primaria son denominadas **claves alternativas.**

Si en nuestra tabla **DEPARTAMENTO** escogemos **NUMEDPTO** como clave primaria, el **NOMDPTO** será nuestra clave alternativa.

5.2.- Clave externa, ajena o secundaria.

Una **clave ajena**, también llamada **externa o secundaria**, es un atributo o conjunto de atributos de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (o de la misma). Las claves ajenas representan relaciones entre datos. Dicho de otra manera, son los datos de atributos de una tabla cuyos valores están relacionados con atributos de otra tabla.



NUM_DPTO	NOM_DPTO	PRESUPUESTO
D1	Marketing	60000
D2	Desarrollo	100000
D3	Investigación	30000

NUM_EMP	APELLIDOS	NUM_DPTO	SALARIO
E1	Fernandez	D1	900
E2	Gonzalez	D2	1000
E3	Lopez	D2	1000
E4	Menendez	D3	800

Es lógico que las claves ajenas no tengan las mismas propiedades y restricciones que tienen como clave primaria en su tabla, por tanto, sí que pueden repetirse en la tabla.

Las claves ajenas tienen por objetivo establecer una conexión con la clave primaria que referencian. Por lo tanto, los valores de una clave ajena deben estar presentes en la clave primaria correspondiente, o bien deben ser valores nulos. En caso contrario, la clave ajena representaría una referencia o conexión incorrecta.

6.- Índices.

Un **índice** es una estructura de datos que permite acceder a **diferentes filas** de una misma tabla **a través de un campo** o campos. Esto permite un acceso mucho más rápido a los datos.

Los índices son útiles cuando se realizan consultas frecuentes a un rango de filas o una fila de una tabla.

Los cambios en los datos de las tablas (agregar, actualizar o borrar filas) son incorporados automáticamente a los índices con transparencia total.

Debes saber que los índices son independientes, lógica y físicamente de los datos, es por eso que pueden ser creados y eliminados en cualquier momento, sin afectar a las tablas ni a otros índices.

¿Cuándo indexamos? No hay un límite de columnas a indexar, si quisiéramos podríamos crear un índice para cada columna, pero no sería operativo. Normalmente tiene sentido crear índices para ciertas columnas ya que agilizan las operaciones de búsqueda de base de datos grandes.

Al crear índices, las operaciones de modificar o agregar datos se ralentizan, ya que al realizarlas es necesario actualizar tanto la tabla como el índice.

Si se elimina un índice, el acceso a datos puede ser más lento a partir de ese momento.

7.- El valor NULL. Operaciones con este valor.

¿Qué sucede si al guardar los datos de los Usuarios hay algún dato que no tengo o no necesito guardarlo porque no corresponde?

Independientemente del dominio al que pertenezca un campo, éste puede tomar un valor especial denominado **NULO** (**NULL** en inglés) que designará la ausencia de dato.

Cuando por cualquier motivo se desconoce el valor de un campo, por ejemplo, desconocemos el teléfono del usuario, o bien ese campo carece de sentido (siguiendo con el mismo ejemplo, puede que el usuario no tenga teléfono), podemos asignar a ese campo el valor especial NULO.

Cuando trabajamos con claves secundarias el valor nulo indica que la tupla o fila no está relacionada con ninguna otra tupla o fila. Este valor NULO es común a cualquier dominio.

Pero ten en cuenta una cosa, **no es lo mismo valor NULO que ESPACIO EN BLANCO**.

Tampoco será lo mismo valor NULO que el valor CERO.

Un ordenador tomará un espacio en blanco como un carácter como otro cualquiera. Por tanto, si introducimos el carácter "espacio en blanco" estaríamos introduciendo un valor que pertenecería al dominio texto y sería distinto al concepto "ausencia de valor" que sería no incluir nada (nulo).

Este valor se va a utilizar con frecuencia en las bases de datos y es imprescindible saber cómo actúa cuando se emplean operaciones lógicas sobre ese valor. En la lógica booleana tenemos los valores VERDADERO y FALSO, pero un valor NULO no es ni verdadero ni falso.

Cuando necesitemos comparar dos campos, si ambos son nulos no podremos obtener ni verdadero ni falso. Necesitaremos definir la lógica con este valor.

Veamos los operadores lógicos más comunes y sus resultados utilizando el valor nulo:

- **Operador AND**

Valor1	Valor2	Operador Y(AND)
VERDADERO	NULO	NULO
FALSO	NULO	FALSO

- **Operador OR**

Valor1	Valor2	Operador O (OR)
VERDADERO	NULO	VERDADERO
FALSO	NULO	NULO

- **Operador NOT.**

Valor1	Operador NO (NOT)
NULO	NULO

En todas las bases de datos relacionales se utiliza un operador llamado **ES NULO (IS NULL)** que devuelve VERDADERO si el valor con el que se compara es NULO.

8.- Vistas.

Una **vista** es una **"tabla virtual"** cuyas filas y columnas se obtienen a partir de una o de varias tablas que constituyen nuestro modelo. Lo que se almacena no es la tabla en sí, sino su definición, por eso decimos que es "virtual". Una vista actúa como filtro de las tablas a las que hace referencia en ella.

La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la base de datos actual u otras bases de datos.

Corresponde al **esquema externo** o visión que necesita el usuario de la BD.

No existe ninguna restricción a la hora de consultar vistas y muy pocas restricciones a la hora de modificar los datos de éstas.

Podemos dar dos razones por las que queramos crear vistas:

- **Seguridad**: Nos puede interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.
- **Comodidad**: Al pasar nuestras tablas/relaciones a un lenguaje de base de datos, puede que tengamos que escribir sentencias bastante complejas, las vistas no son tan complejas.

Las vistas no tienen una copia física de los datos, son consultas a los datos que hay en las tablas, por lo que si actualizamos los datos de una vista, estamos actualizando realmente la tabla, y si actualizamos la tabla estos cambios serán visibles desde la vista.

Aunque no siempre podremos actualizar los datos de una vista, dependerá de la complejidad de la misma y del gestor de base de datos.

9.- Usuarios. Roles. Privilegios.

A la hora de conectarnos a la base de datos es necesario que utilicemos un modo de acceso, de manera que queden descritos los permisos de que dispondremos durante nuestra conexión. En función del **nombre de usuario** tendremos unos permisos u otros.

Un **usuario** es un conjunto de permisos que se aplican a una conexión de base de datos. Tiene además otras funciones como son:

- ✓ Ser el **propietario** de ciertos objetos (tablas, vistas, etc.).
- ✓ Realiza las **copias de seguridad**.
- ✓ Define una **cuota de almacenamiento**.
- ✓ Define el **tablespace por defecto** para los objetos de un usuario en Oracle.

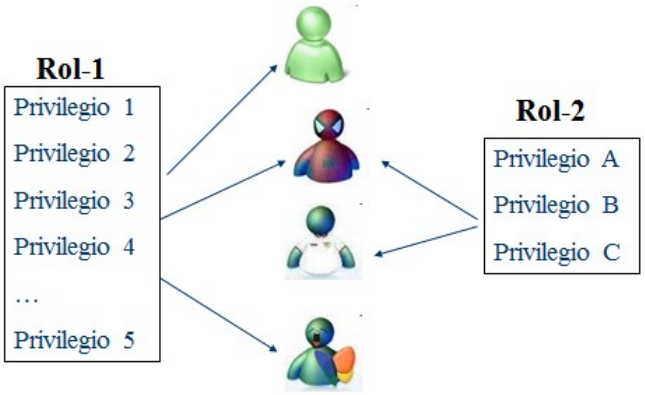
Pero no todos los usuarios deberían poder hacer lo mismo cuando acceden a la base de datos. Por ejemplo, un administrador debería tener más privilegios que un usuario que quiere realizar una simple consulta.

¿**Qué es un privilegio**? No es más que **un permiso** dado a un usuario para que realice ciertas operaciones, que pueden ser de dos tipos:

- **De sistema**: dan derecho a ejecutar un tipo de comando SQL o a realizar alguna acción sobre objetos.
  Ejemplo. Crear tablespaces, crear vistas,…
- **Sobre objeto**: permiten acceder y realizar cambios en los datos de los usuarios.
  Ejemplo: consulta de tablas de otro usuario, insertar, modificar

¿Y no sería interesante poder agrupar esos permisos para darlos juntos? Para eso tenemos el **rol.**

Un rol de base de datos no es más que **una agrupación de permisos** de sistema y de objeto.



Podemos tener a un grupo determinado de usuarios que tengan permiso para consultar los datos de una tabla concreta y no tener permiso para actualizarlos. Luego un rol permite asignar un grupo de permisos a un usuario. De este modo, si asignamos un rol con 5 permisos a 200 usuarios y luego queremos añadir un permiso nuevo al rol, no tendremos que ir añadiendo este nuevo permiso a los 200 usuarios, ya que el rol se encarga de propagarlo automáticamente.

10.- SQL.

SQL (**Structured Query Language**) es el lenguaje fundamental de los SGBD relacionales. Es uno de los lenguajes más utilizados en informática en todos los tiempos. Es un **lenguaje declarativo** y por tanto, lo más importante **es definir qué se desea hacer**, y **no cómo hacerlo**. De esto último ya se encarga el SGBD.

Hablamos por tanto de un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje (ASP o PHP) en combinación con cualquier tipo de base de datos (Access, SQL Server, MySQL, Oracle, etc.).

El hecho de que sea **estándar** no quiere decir que sea idéntico para cada base de datos. Así es, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aunque SQL está estandarizado, siempre es recomendable revisar la documentación del SGBD con el que estemos trabajando para conocer su sintaxis concreta, ya que algún comando, tipo de dato, etc., puede no seguir el estándar.

SQL posee dos características muy apreciadas, **potencia** y **versatilidad**, que contrastan con su facilidad para el aprendizaje, ya que utiliza un lenguaje bastante natural. Es por esto que las instrucciones son muy parecidas a órdenes humanas. Por esta característica se le considera un **Lenguaje de Cuarta Generación**.

Aunque frecuentemente oigas que SQL es un "lenguaje de consulta", ten en cuenta que no es exactamente cierto ya que contiene muchas otras capacidades además de la de consultar la base de datos:

- ✓ La **definición** de la propia estructura de los datos.
- ✓ Su **manipulación**.
- ✓ Y la **especificación** de conexiones seguras.

Por tanto, el **lenguaje estructurado de consultas SQL es un lenguaje que permite operar con los datos almacenados en las bases de datos relacionales.**

10.1.- Elementos del lenguaje. Normas de Escritura.

El lenguaje SQL está compuesto por comandos, cláusulas, operadores, funciones y literales. Todos estos elementos se combinan en las instrucciones y se utilizan para crear, actualizar y manipular bases de datos. Estos conceptos son bastante amplios por eso será mejor que vayamos por partes.

- **Comandos**:
  Van a ser las instrucciones que se pueden crear en SQL. Se pueden distinguir en tres grupos que veremos con más detenimiento a lo largo de las siguientes unidades:
  - De **definición de datos** (**DDL**, Data Definition Language), que permiten crear y definir nuevas bases de datos, tablas, campos, etc.
  - De **manipulación de datos** (**DML**, Data Manipulation Language), que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.
  - De **control y seguridad de datos** (**DCL**, Data Control Language), que administran los derechos y restricciones de los usuarios.

- **Cláusulas:**
  Llamadas también condiciones o criterios, son palabras especiales que permiten modificar el funcionamiento de un comando.
  Por ejemplo: FROM, WHERE, GROUP BY, HAVING, BY

- **Operadores**:
  Permiten crear expresiones complejas. Pueden ser aritméticos (+, -, *, /, ...) o lógicos (< , >, >=, <=, < >, And, Or, not, between, like, in).

- **Funciones**:
  Para conseguir valores complejos. Por ejemplo, la función promedio para obtener la media de un salario.
  Por ejemplo: AVG, COUNT, SUM, MAX, MIN

- **Literales**: Les podemos llamar también constantes y serán valores concretos, como por ejemplo un número, una fecha, un conjunto de caracteres, etc.

Y tendremos que seguir unas normas sencillas pero primordiales:

- Todas las instrucciones terminan con un signo de punto y coma.
- No se distingue entre mayúsculas y minúsculas.
- Cualquier comando puede ser partido con saltos de línea o espacios para facilitar su lectura y comprensión.
- Los comentarios comienzan por /* y terminan con */ (excepto en algunos SGBD).

11.- Lenguaje de descripción de datos.

La primera fase del trabajo con cualquier base de datos comienza con sentencias **DDL**, puesto que antes de poder almacenar y recuperar información debemos definir las estructuras donde almacenar la información. Las estructuras básicas con las que trabaja SQL son las tablas.

Conocer el **Lenguaje de Definición de Datos** (**DDL**) es imprescindible para crear, modificar y eliminar objetos de la base de datos (es decir, los metadatos). En el mercado hay suficientes aplicaciones y asistentes que nos facilitan esta labor, a través de una interfaz visual que nos oculta el lenguaje SQL y en los cuales nos limitamos a poner nombres a los campos, elegir el tipo de datos y activar una serie de propiedades.

Es cierto que estas herramientas nos facilitan el trabajo, pero resulta imprescindible comprender y conocer en profundidad el lenguaje, ya que nos veremos en muchas situaciones donde necesitaremos crear un objeto, modificarlo o eliminarlo sin depender de esas herramientas visuales.

En **Oracle, cada usuario de una base de datos tiene un esquema**, que tendrá el mismo nombre que el usuario con el que se ha accedido y **sirve para almacenar los objetos que posea ese usuario**.

¿De qué objetos estamos hablando? Éstos podrán ser tablas, vistas, índices u otros objetos relacionados con la definición de la base de datos. ¿Y quién puede crear y manipularlos? En principio el usuario propietario (el que los creó) y los administradores de la base de datos. Más adelante veremos que podemos modificar los privilegios de los objetos para permitir el acceso a otros usuarios.

Las **instrucciones DDL** generan acciones que **no se pueden deshacer**, por eso es conveniente usarlas con precaución y tener copias de seguridad cuando manipulamos la base de datos.

11.1.- Creación de bases de datos. Objetos de la base de datos.

Básicamente, la creación de la base de datos consiste en crear las tablas que la componen. Aunque antes de esto tendríamos que definir un espacio de nombres separado para cada conjunto de tablas. Es lo que antes hemos llamado esquemas o usuarios.

Crear una base de datos implica indicar los archivos y ubicaciones que se van a utilizar además de otras indicaciones técnicas y administrativas. Es obvio que todo esto sólo lo puede realizar si se tiene privilegio de Administrador.

Con el estándar de SQL la instrucción a usar sería **Create Database**, pero cada SGBD tiene un procedimiento para crear las bases de datos. Crearíamos una base de datos con el nombre que se indique a continuación.

```
CREATE DATABASE NombredemiBasedeDatos;
```

Hemos estado hablando de objetos de la base de datos, ahora veremos a qué nos referimos.

Según los estándares, **una base de datos es un conjunto de objetos** que nos servirán para gestionar los datos. Estos objetos están **contenidos en esquemas** y éstos a su vez suelen estar **asociados a un usuario**. De ahí que antes dijéramos que **cada base de datos tiene un esquema** que está **asociado a un usuario**.

11.2.- Creación de tablas.

Lo primero que necesitamos en nuestra base de datos será definir los objetos donde vamos a agrupar esos datos. Los **objetos básicos** con los que trabaja SQL **son las tablas**, que como ya sabemos es un conjunto de filas y columnas cuya intersección se llama celda. Es ahí donde se almacenarán los elementos de información, los datos que queremos recoger.

Antes de crear la tabla es conveniente planificar algunos detalles:

- Qué **nombre** le vamos a dar **a la tabla**.
- Qué nombre le vamos a dar **a cada una de las columnas**.
- Qué **tipo y tamaño de datos** vamos a almacenar **en cada columna**.
- Qué **restricciones** tenemos sobre los datos.
- Alguna otra **información adicional** que necesitemos.

Y debemos tener en cuenta otras reglas que se deben cumplir para los nombres de las tablas:

- ✓ No podemos tener nombres de tablas duplicados en un mismo esquema (usuario).
- ✓ Deben comenzar por un carácter alfabético.
- ✓ Su longitud máxima es de 30 caracteres.
- ✓ Solo se permiten letras del alfabeto inglés, dígitos o el signo de guión bajo.
- ✓ No puede coincidir con las palabras reservadas de SQL (por ejemplo, no podemos llamar a una tabla WHERE).
- ✓ No se distingue entre mayúsculas y minúsculas.

La sintaxis básica del comando que permite crear una tabla es la siguiente:

```
CREATE TABLE [esquema.] nombredeTabla
(
        columna1 tipo_dato [not null],
        columna2 tipo_dato [not null],
        ...
        columnaN tipo_dato [not null]
)[TABLAESPACE espacio de la tabla];
```

Dónde:

- ✓ columna1, columna2, ..., columnaN son los nombres de las columna que contendrá la tabla.
- ✓ Tipo_Dato indica el tipo de dato de cada columna.
- ✓ TABLESPACE espacio_de_tabla: señala el TABLEESPACE para almacenar la tabla, si no se indica lo hará en el tablaespace por defecto.
- ✓ NOT NULL: especifica si la columna puede contener valores nulos o no.

■ Nombres de las columnas

Para referenciar una columna se puede indicar de las siguientes formas:

- ✓ nombreColumna
- ✓ nombreTabla.nombreColumna
- ✓ esquemaUsuario.nombreTabla.nombreColumna

Siempre que la sentencia afecte a más de una tabla es aconsejable especificar la tabla a la que pertenece cada columna. Si estamos trabajando en un esquema y la tabla pertenece a ese esquema no es necesario indicar el esquemaUsuario.

■ Tipos de Datos.

Es importante conocer los tipos de datos disponibles en un Sistema Gestor de Base de Datos (SGBD), de esta forma podremos optimizar al máximo el uso de recursos de nuestras aplicaciones.

A continuación os indicamos todos los tipos de datos de Oracle más utilizados:

- **CHAR**(longitud)
  - ✓ Almacena cadena de caracteres de longitud fija entre 1 y 255
  - ✓ Las columnas tienen una longitud fija.
  - ✓ Si se introduce una cadena de menor longitud que la definida se rellenará con blancos a la derecha hasta que quede completa.
  - ✓ Si se introduce una cadena de mayor longitud que la fijada Oracle devolverá un error.

- **VARCHAR2** (longitud):
  - ✓ Almacena cadena de caracteres de longitud variable (max. 2000 caracteres)
  - ✓ Las columnas de este tipo tienen una longitud variable.
  - ✓ Si se introduce una cadena de menor longitud que la que está definida se almacenará con esa longitud.
  - ✓ Si se introduce una cadena de mayor longitud a la fijada, Oracle devolverá error.

- **NUMBER** (precisión, escala):
  - ✓ Almacena datos numéricos, tanto enteros como decimales, con o sin signo.
  - ✓ Opcionalmente se le puede indicar la precisión y la escala.
  - ✓ "**precisión**": el número total de dígitos que va a tener el dato que se define: el rango de 1 a 38.
  - ✓ "**escala**": número de dígitos a la derecha del punto decimal: rango de –84 a 127. Una escala negativa indica redondear tantos dígitos a la izquierda del punto decimal.
  - ✓ Si no se indica escala, el valor de esta es 0
  - ✓ Si no se indica precisión ni escala, se almacenará el valor tal y como se introduzca.

- **DATE**:
  - ✓ Almacena información de fecha y hora. Para cada tipo DATE almacena la siguiente información:
    Siglo / años / mes / días / horas / minutos / segundos.
  - ✓ Por defecto el valor para el formato de la fecha es una cadena de caracteres del tipo dd/mm/yy.
  - ✓ Se puede modificar con la orden utilizando el parámetro NLS_DATE_FORMAT.

- **DATETIME**:
  - ✓ Combinación de fecha y hora: yy-MM-dd hh-mm-ss con rango desde 1/1/1001 hasta 31/12/9999.

- **TIMESTAMP**:
  - ✓ Igual que el anterior pero desde 1/1/1970 31/12/2037

- **LONG**:
  - ✓ Almacena cadenas de caracteres de longitud variable (hasta 2Gb). Este tipo de datos está obsoleto (en desuso), en su lugar se utilizan los datos de tipo LOB (CLOB, NCLOB). Oracle recomienda que se convierta el tipo de datos LONG a alguno LOB si aún se está utilizando.

    Restricciones:
    - ➢ Sólo se puede definir una columna LONG por tabla
    - ➢ No puede aparecer en restricciones de integridad (CONSTRAINTS).
    - ➢ No sirve para indexar ni utilizar como argumento en funciones

- Una función almacenada no puede devolver un valor LONG.
- No es posible su uso en cláusulas WHERE, GROUP BY, ORDER BY, CONNECT BY, DISTINCT BY, ni con operaciones de UNIÓN, INTERSERC Y MINUS
- No se puede referenciar como subconsulta en la creación de tablas, ni en inserción de filas.
- Las variables o argumentos de bloques PL/SQL no se pueden declarar como tipo LONG.

- **RAW**(varchar2) y **LONG RAW**(long):
  - ✓ Almacena datos binarios
  - ✓ RAW es igual a varchar2 pero maneja cadena de bytes en vez de caracteres Maximo 255 bytes
  - ✓ LONG RAW es igual a LONG y se emplea para almacenar gráficos, sonidos,..
  - ✓ Ya está en desuso y se sustituye por los tipo LOB

- **LOB** (**BLOG, CLOB, NCLOB, BFILE**)
  - ✓ Permiten almacenar y manipular bloques grandes de datos no estructurados (tales como texto, imágenes, videos, sonidos, etc) en formato binario o del carácter.
  - ✓ Admiten hasta 8 terabytes (8000 GB).
  - ✓ Una tabla puede contener varias columnas de tipo LOB.
  - ✓ Soportan acceso aleatorio.
  - ✓ Las tablas con columnas de tipo LOB no pueden ser replicadas

- **BLOG.**
  - ✓ Permite almacenar datos binarios no estructurados
  - ✓ Admiten hasta 8 terabytes

- **CLOB**
  - ✓ Almacena datos de tipo carácter
  - ✓ Admiten hasta 8 terabytes

- **NCLOB**
  - ✓ Almacena datos de tipo carácter
  - ✓ Admiten hasta 8 terabytes.
  - ✓ Guarda los datos según el juego de caracteres Unicode nacional.

- **ROWIND**
  - ✓ Identifica de forma única la dirección de cada fila. Utiliza una representación binaria de la localización física de la fila.

De los tipos anteriores, los comúnmente utilizados son:

- ✓ VARCHAR2 para cadenas de texto no muy grandes
- ✓ DATE para fechas y horas
- ✓ NUMBER para números
- ✓ BLOB para ficheros de tipo word, excel, access, video, sonido, imágenes, etc
- ✓ CLOB para cadenas de texto muy grandes.

Por ejemplo:

```
CREATE TABLE ALUMNOS
(
    num_matricula    NUMBER(6)    NOT NULL,
    dni              VARCHAR2(9) NOT NULL,
    nombre           VARCHAR2(15),
    fecha_nto        DATE,
    direccion        VARCHAR2(30),
    localidad        VARCHAR2(15),
    correo           VARCHAR2(15)
) TABLESPACE users;
```

Los usuarios pueden consultar las tablas creadas por medio de la VISTA USER_TABLES.

```
Select * from user_tables;
```

Existen otras dos VISTAS que permiten obtener información de los objetos que son propiedad del usuario.

- Objetos que son propiedad del usuario

```
Select * from USER_OBJECTS
```

- Tablas, vistas, sinónimos y secuencias propiedad del usuario

```
Select * from USER_CATALOG;
```

11.3.- Restricciones.

Hay veces que necesitamos que un dato se incluya en una tabla de manera obligatoria, otras veces necesitaremos definir uno de los campos como llave primaria o ajena. Todo esto podremos hacerlo cuando definamos la tabla, además de otras opciones.

Antes de hablar de las restricciones veamos el concepto de **integridad**.

- **Integridad:** Hace referencia al hecho de que los datos de la base de datos han de ajustarse a restricciones antes de almacenarse en ella.
- **Restricción de integridad**: será una regla que restringe el rango de valores para una o más columnas de la tabla.
- **Integridad referencial**: garantiza que los valores de las columnas de una tabla dependan de los valores de otra columna de otra tabla.

Una **restricción es una condición** que una o varias columnas **deben cumplir obligatoriamente**.

Cada restricción que creemos llevará un nombre, si no se lo ponemos nosotros lo hará Oracle o el SGBD que estemos utilizando. Es conveniente que le pongamos un nombre que nos ayude a identificarla y que sea único para cada esquema (usuario).

Se aconseja la siguiente regla a la hora de poner nombre a las restricciones:

- Tres letras para el nombre de la tabla.
- Carácter de subrayado.
- Tres letras con la columna afectada por la restricción.
- Carácter de subrayado.
- Dos letras con la abreviatura del tipo de restricción. La abreviatura puede ser:
  - ✓ PK = Primary Key.
  - ✓ FK = Foreign Key.
  - ✓ NN = Not Null.
  - ✓ UK = Unique.
  - ✓ CK = Check (validación).

Para definir las restricciones de utiliza la cláusula **CONSTRAINT** y pueden ser:

- **Restricción de columna**: Restringe una sola columna y se define como parte de la definición de la columna.
- **Restricción de tabla**: Restringe un grupo de columnas y se define al final de la definición de la tabla, una vez definida todas las columnas
  Se puede hacer referencia a varias columnas en una única restricción (por ejemplo, declarando dos columnas como clave primaria o ajena).

Veamos cada una de las posibles restricciones, su significado y uso.

- **Restricción NOT NULL.**

  Con esta restricción obligaremos a que esa columna tenga un valor o lo que es lo mismo, prohíbe los valores nulos para una columna en una determinada tabla.
  Podremos ponerlo cuando creamos o modificamos el campo añadiendo la palabra NOT NULL después de poner el tipo de dato.

  Por ejemplo: Si en la tabla ALUMNOS queremos que el campo **Fecha_Nacimiento**; sea obligatorio ponerlo, nos quedaría así:

```
CREATE TABLE ALUMNOS
(
    Fecha_Nto DATE  CONSTRAINT Alu_Fec_NN  NOT  NULL
);
```

  O bien, de esta otra forma:

```
CREATE TABLE ALUMNOS
(
   Fecha_Nto DATE  NOT  NULL
);
```

- **Restricción UNIQUE.**

  Habrá ocasiones en la que nos interese que no se puedan repetir valores en la columna, en estos casos utilizaremos la restricción **UNIQUE**. Oracle crea un índice automáticamente cuando se habilita esta restricción y lo borra al deshabilitarla.

  También para esta restricción tenemos dos posibles formas de ponerla. Supongamos que el campo **DNI** de nuestra tabla va a ser único. Lo incluiremos en la tabla que estamos creando. Nos quedaría así:

```
CREATE TABLE ALUMNOS
(
    dni  VARCHAR2(9)  CONSTRAINT Alu_Dni_UK UNIQUE
);
```

  Veamos otra forma:

```
CREATE TABLE ALUMNOS
(
    dni  VARCHAR2(9)  UNIQUE
);
```

También podemos poner esta restricción a varios campos a la vez, por ejemplo, si queremos que **dni** y **correo** electrónico sean únicos podemos ponerlo así:

```
CREATE TABLE ALUMNOS
(
    Dni      VARCHAR2 (9),
    Correo   VARCHAR2 (25),
    CONSTRAINT Alumno_UK  UNIQUE (dni, correo)
);
```

Detrás del tipo de datos de **Correo** hay una coma, eso es así porque la restricción es independiente de ese campo y común a varios. Por eso después de UNIQUE hemos puesto entre paréntesis los nombres de los campos a los que afecta la restricción.

- **Restricción PRIMARY KEY**

En el modelo relacional las tablas deben tener una clave primaria. Es evidente que cuando creamos la tabla tendremos que indicar a quién corresponde.

Sólo puede haber una clave primaria por tabla pero ésta puede estar formada por varios campos. Dicha clave podrá ser referenciada como clave ajena en otras tablas.

La clave primaria hace que los campos que forman sean NOT NULL y que los valores de los campos sean de tipo UNIQUE.

Veamos cómo quedaría si la clave fuese el campo **num_matricula**:

- Si la clave la forma un único campo:

```
CREATE TABLE ALUMNOS
(
    num_matricula    NUMBER(6)   PRIMARY KEY
);
```

- O bien poniendo un nombre a la restricción:

```
CREATE TABLE ALUMNOS
(
    num_matricula    NUMBER(6)  CONSTRAINT Alu_num_PK  PRIMARY KEY,
    dni              VARCHAR2(9),
    nombre           VARCHAR2(15) NOT NULL,


);
```

- Si la clave está formada por más de un campo, por ejemplo **num_matricula** y **dni**:

```
CREATE TABLE ALUMNOS
(
    num_matricula    NUMBER(6)   NOT NULL,
    dni              VARCHAR2(9)  NOT NULL,
    nombre           VARCHAR2(15),

    CONSTRAINT Alu_PK  PRIMARY KEY (num_matricula, dni)
);
```

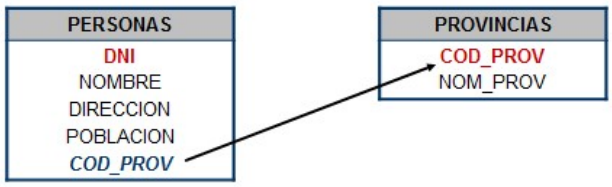
Restricción REFERENCES.FOREIGN KEY.

Ya vimos que las claves ajenas, secundarias o foráneas eran campos de una tabla que se relacionaban con la clave primaria de otra tabla.

Cuando creemos la tabla tendremos que indicar de alguna forma quién es clave ajena. Lo haremos "haciendo referencia" a la tabla y los campos de donde procede.

Puede existir más de una clave ajena en una tabla.

Las claves ajenas deben de tener un valor NULL o un valor igual al de la clave referenciada (REGLA DE INTEGRIDAD REFERENCIAL)



Podemos utilizar cualquiera de los dos formatos siguientes para definir claves ajenas:

FORMATO 1: la clave ajena se define en la restricción de la columna usando **REFERENCES**

```
CREATE TABLE nombre_tabla
(
     colum1 TIPO_DE_DATO
          [CONSTRAINT nombre_restricción]
          REFERENCES nombre_tabla [(columna)]
                    [ON DELETE {CASCADE | SET NULL} ],
     colum2 TIPO_DATO,
     .....
) [TABLESPACE espacio_de_tabla];
```

FORMATO 2: La clave ajena se define al final de todas la columnas usando FOREIGN KEY y REFERENCES.

```
CREATE TABLE nombre_tabla
(
     colum1   TIPO_DE_DATO,
     colum2   TIPO_DE_DATO,
     ..............
     [CONSTRAINT nombre_restricción]
          FOREIGN KEY (columna [, columna])
          REFERENCES nombre_tabla [(columna [, columna])]
          [ON DELETE {CASCADE / SET NULL}],
     ..........
)[TABLESPACE espacio_de_tabla];
```

Por ejemplo si tenemos la tablas

PERSONAS (**dni**, nombre, direccion, población, cod_prov)

PROVINCIAS (**cod_prov**, nom_prov)

Será necesario crear primero la tabla **PROVINCIAS** y después la tabla **PERSONAS** (que referencia a **PROVINCIAS**) para que no se produzca error en ORACLE. Para borrar será primero la tabla que hace la referencia (PERSONAS) y después la referenciada (PROVINCIAS).

```
Create table PROVINCIAS
(
     cod_prov          number(2) PRIMARY KEY,
     nom_prov          varchar2(15)
);

Create table PERSONAS
(
     Dni               number(8) PRIMARY KEY,
     Nombre            varchar2(15),
     direccion         varchar2(25),
     poblacion         varchar2(20),
     cod_prov          number(2) NOT NULL,
        FOREIGN KEY (cod_prov)  REFERENCES   PROVINCIAS
);

También podríamos poner:
cod_prov        number(2) NOT NULL REFERENCES      PROVINCIAS
```

Si hacemos referencia a una tabla que no está creada: Oracle buscará la tabla referenciada y al no encontrarla dará fallo. Esto se soluciona creando en primer lugar las tablas que no tengan claves ajenas.

Si queremos borrar las tablas tendremos que proceder al contrario, borraremos las tablas que tengan claves ajenas antes.

Tenemos otras soluciones y es añadir tras la cláusula REFERENCE:

- ✓ **ON DELETE CASCADE**: te permitirá borrar todos los registros cuya clave ajena sea igual a la clave del registro borrado.

```
Create table PERSONAS
(
      dni                 number(8) PRIMARY KEY,
      nombre              varchar2(15),
      direccion           varchar2(25),
      poblacion           varchar2(20),
      cod_prov            number(2) NOT NULL,
            FOREIGN KEY (cod_prov)  REFERENCES  PROVINCIAS
                  ON DELETE  CASCADE
);
```

Se utiliza cuando al borrar las filas asociadas con claves primarias (PROVINCIAS), deseamos que se borren automáticamente las correspondientes filas con clave ajena, dependientes de otras tablas (PERSONAS).

- ✓ **ON DELETE SET NULL**: colocará el valor NULL en todas las claves ajenas relacionadas con la borrada.

```
Create table PERSONAS
(
      dni                 number(8) PRIMARY KEY,
      nombre              varchar2(15),
      direccion           varchar2(25),
      poblacion           varchar2(20),
      cod_prov            number(2) NOT NULL,
          FOREIGN KEY (cod_prov)  REFERENCES  PROVINCIAS
               ON DELETE SET NULL
);
```

Se utiliza cuando al borrar las filas asociadas con claves primarias (PROVINCIAS), pero no queremos que se borren las correspondientes filas con clave ajena, dependientes de otras tablas (PERSONAS), en este caso pone a NULL dicho campo en PERSONAS

- **Restricción DEFAULT**.

A veces es muy tedioso insertar siempre lo mismo en un campo. Imagínate que casi todos las personas de nuestra tablan fuesen de Avilés y tenemos un campo Población. ¿No sería cómodo asignarle un valor por defecto? Eso es lo que hace la restricción **DEFAULT.**

```
Create table PERSONAS
(
      dni                 number(8) PRIMARY KEY,
      nombre              varchar2(15),
      direccion           varchar2(25),
      poblacion           varchar2(20) DEFAULT 'Aviles',
      cod_prov            number(2) NOT NULL,
          FOREIGN KEY (cod_prov)  REFERENCES  PROVINCIAS
               ON DELETE SET NULL
);
```

En las especificaciones de DEFAULT vamos a poder añadir distintas expresiones: constantes, funciones SQL y variables.

Si queremos incluir en un campo la fecha actual, independientemente del día en el que estemos, podremos utilizar la función SYSDATE como valor por defecto:

```
Create table PERSONAS
(
      ……….
      Fecha_ingreso DATE DEFAULT SYSDATE
);
```

- **Restricción Validación CHECK.**

Esta restricción comprueba que se cumpla una condición determinada al rellenar una columna. Dicha condición se puede construir con columnas de esa misma tabla.

Si en una tabla **USUARIOS** tenemos el campo Crédito y éste sólo puede estar entre 0 y 2000, lo especificaríamos así:

```
Create table USUARIOS
(
     Credito   NUMBER(4)  CHECK  (Crédito BETWEEN 0 AND 2000)
);
```

Una misma columna puede tener varios **CHECK** asociados a ella, para ello ponemos varios **CONSTRAINT** seguidos y separados por comas.

11.4.- Eliminacion de tablas.

Cuando una tabla ya no es útil y no la necesitamos es mejor borrarla, de este modo no ocupará espacio y podremos utilizar su nombre en otra ocasión.

Para eliminar una tabla utilizaremos el comando **DROP TABLE.**
El formato es:

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

La opción **CASCADE CONSTRAINTS** permite suprimir todas las restricciones de integridad referencial que se refieran a claves de la tabla borrada

Un usuario solo puede borrar sus propias tablas

Solo el administrador o algún usuario al que se le otorgara privilegios puede borrar las tablas de otro usuario

Al borrar una tabla:

- ✓ Desaparecen todos sus datos.
- ✓ Se suprimen los índices y privilegios asociados a ella.
- ✓ Las vistas y sinónimos asociados a una tabla dejan de funcionar **pero no desaparecen**, por lo que habría que eliminarlos.

Ejemplo: tabla PROVINCIAS y PERSONAS (con clave ajena a PROVINCIAS)

```
drop table PROVINCIAS;
```

Produciría un error debido ya que existen tablas (en este caso PERSONAS) que tienen alguna columna que es clave ajena y que referencian a la tabla PROVINCIAS, por lo que si eliminamos la tabla PROVINCIAS, la columna PERSONAS.COD_PROV no tendría sentido al referenciar a una tabla que no existe.
Para evitar esto sería necesario borrar la tabla PROVINCIAS, pero indicando que elimine también todas las referencias de otras tablas a PROVINCIAS

```
drop table PROVINCIAS cascade constraints;
```

Ten cuidado al utilizar este comando, el borrado de una tabla es irreversible y no hay una petición de confirmación antes de ejecutarse.

11.5.- Limpieza de tablas.

Oracle dispone de la orden **TRUNCATE TABLE** que te permitirá eliminar los datos (filas) de una tabla sin eliminar su estructura.
Esta orden libera el espacio ocupado para poder utilizarlo en otros usos.

Una instrucción **TRUNCATE** no permite **ROLLBACK**

Formato:

```
TRUNCATE TABLE [usuario.] nom_tabla [{DROP | REUSE} STORAGE];
```

Donde:

DROP STORAGE: opción por defecto, libera el espacio de las filas borradas

REUSE STORAGE: borra las filas, pero mantiene el espacio reservado para nuevas filas.

No se puede truncar una tabla cuya clave primaria sea clave ajena de otra tabla y exista integridad referencial definida. Habrá que desactivar la restricción.

Ej. TRUNCATE TABLE PROVINCIAS (producirá error hasta que se desactive la restricción, en este caso suponiendo que otra tabla p.e. PERSONAS tengan clave ajena a PROVINCIAS)

Ten cuidado al utilizar este comando, el borrado de una tabla es irreversible y no hay una petición de confirmación antes de ejecutarse.

11.6.- Modificación de tablas.

Es posible que después de crear una tabla nos demos cuenta que se nos ha olvidado añadir algún campo o restricción, quizás alguna de las restricciones que añadimos ya no es necesaria o tal vez queramos cambiar el nombre de alguno de los campos. En estos casos utilizaremos el comando **ALTER TABLE.**

Las tablas se pueden modificar:

- ✓ Cambiando la definición de una columna.
- ✓ Añadiendo/borrando una columna.

Formato:

```
ALTER TABLE nombre_tabla
{
      [ADD          (columna  tipo)]
      [DROP         COLUMN (columna [, columna ]...)]
      [MODIFY       (columna tipo [, columna tipo]...)]
      [RENAME       COLUMN NombreAntiguo TO NombreNuevo]
      [ADD          CONSTRAINT restricción]
      [DROP         CONSTRAINT restricción]
      [DISABLED     CONSTRAINT restricción]
      [ENABLE       CONSTRAINT restricción]
};
```

- **[ADD (columna tipo)]**: Añade columnas a una tabla
  - ✓ Si la columna no está definida como NOT NULL, se puede añadir en cualquier momento.
  - ✓ Si la columna está definida como NOT NULL:
    - Se define primero la columna sin especificar NOT NUL
    - A continuación se le asigna un valor para cada fila
    - Se modifica la columna a NOT NULL
- **[DROP COLUMN (columna [, columna]...)]:** Borra una columna de una tabla.
  - ✓ Recordar que no se pueden borrar todas las columnas de una tabla
  - ✓ Tampoco se pueden eliminar las claves primarias referenciadas por claves ajenas
- **[MODIFY (columna tipo [, columna tipo].....)]**: Modifica una o más columnas existentes. Habrá que tener en cuenta:
  - ✓ Se puede aumentar la longitud de la columna en cualquier momento
  - ✓ Al disminuir la longitud de una columna con datos no se puede dar menor tamaño que el máximo valor almacenado
  - ✓ Si la columna es NULL en todas las filas de la tabla, se puede disminuir y modificar el tipo de dato
  - ✓ La opción MODIFY … NOT NULL sólo será posible cuando la tabla no contenga ninguna fila con valor nulo en la columna
- **[ADD CONSTRAINT restricción]:** Permita añadir una restricción
- **[DROP CONSTRAINT restricción]:** Permite borrar una restricción
- **[DISABLE CONSTRAINT restricción]:** Permita desactivar una restricción
- **[ENABLE CONSTRAINT restricción]:** Permite activar una restricción

11.7.- Renombrado de Tablas.

Permite cambiar el nombre a una tabla, vista o un sinónimo.

Oracle invalida todos los objetos que dependan del objeto renombrado, como las vistas, los sinónimos que hacen referencia a la tabla renombrada.

Formato:

```
RENAME nombre_old TO nombre_new;
```

Ejemplo:

```
create synonym ALUM from ALUMNOS;
  //sinónimo para tabla ALUMNOS

rename ALUMNOS to TALUMNOS;
  // al renombrar la tabla, el sinónimo ALUM deja de funcionar
```

11.8.- Índices.

Sabemos que crear índices ayuda a la localización más rápida de la información contenida en las tablas.

Es un objeto de la base de datos que se asocia a una tabla y al que se asocia una o varias columnas de la tabla.

- **Formato para crear un índice:**

```
CREATE INDEX nombre_indice ON nombre_tabla
(
    columna [,columna]...
);
```

No es aconsejable que utilices campos de tablas pequeñas o que se actualicen con mucha frecuencia. Tampoco es conveniente si esos campos no se usan en consultas de manera frecuente o en expresiones.

El diseño de índices es un tema bastante complejo para los Administradores de Bases de Datos, ya que una mala elección ocasiona ineficiencia y tiempos de espera elevados. Un uso excesivo de ellos puede dejar a la Base de Datos colgada simplemente con insertar alguna fila.

La mayoría de los índices se crean de manera implícita cuando ponemos las restricciones PRIMARY KEY, FOREIGN KEY o UNIQUE.

- **Formato para eliminar un índice:**

```
DROP INDEX NombreIndice;
```

GLOSARIO

Administrador

Es la persona o equipo de personas profesionales responsables del control y manejo del sistema de base de datos, generalmente tiene experiencia en Sistemas Gestores de Base de Datos, diseño de bases de datos, sistemas operativos, comunicación de datos y programación.

ASP

Tecnología de Microsoft para páginas web generadas dinámicamente, ha sido comercializada como un anexo a Internet Information Services (IIS).

Edgar Frank Cood

Científico informático inglés (23 de agosto de 1923 18 de abril de 2003), conocido por sus aportes a la teoría de bases de datos relacionales.

Estándar

Modelo a seguir al hacer algo. Son documentos que dan los detalles técnicos y las reglas necesarias para que un producto o tecnología se use correctamente

Lenguaje declarativo

Tipo de lenguaje de programación basado más en las matemáticas y en la lógica que los lenguajes imperativos, más cercanos estos al razonamiento humano. Los lenguajes declarativos no dicen cómo hacer una cosa, sino, más bien, qué cosa hacer. A diferencia de los imperativos, no suele haber declaración de variables ni tipos.

Lenguaje de cuarta generación

Lenguajes de programación avanzados en los que el programador no incorpora el procedimiento a seguir, ya que el propio lenguaje es capaz de indicar al ordenador cómo debe ejecutar el programa. Suelen incluir interfaces gráficos y capacidades de gestión avanzadas, pero consumen muchos más recursos del ordenador que la generación de lenguajes previa.

Lógica booleana

Es una lógica de conjuntos y nos sirve, principalmente, para definir formas de intersección entre conjuntos

PHP

Lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

Tablespace

Unidad lógica de almacenamiento dentro de una base de datos Oracle

Versátil

Adaptable a muchas cosas o que tiene varias aplicaciones.