BASES DE DATOS

UNIDAD 2

Bases de datos relacionales

Javier Díaz Garrido

INDICE

Bases de datos relacionales.

Material Complementario

1.- Modelo de datos.

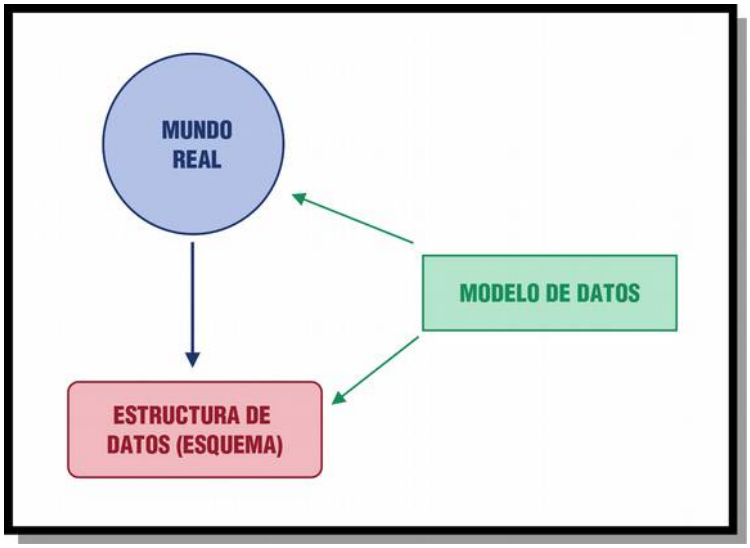
Según el DRAE, un modelo es, entre otras definiciones, el esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja. Podemos decir que es la representación de cualquier aspecto o tema extraído del mundo real. ¿Qué sería entonces un modelo de datos? Aquél que nos permite describir los elementos que intervienen en una realidad o en un problema dado y la forma en que se relacionan dichos elementos entre sí.

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"*

De Miguel, Piattini y Marcos (1999): *"Representación de un determinado mundo real (universo del discurso) en términos de un modelo 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 dos sublenguajes:

- **Lenguaje de Definición de Datos o DDL (Data Definition Language)**, cuya función es describir, de una forma abstracta, las estructuras de datos y las restricciones de integridad.
- **Lenguaje de Manipulación de Datos o DML (Data Manipulation Language)**, que sirven para describir las operaciones de manipulación de los datos.

Autoevaluación

¿Cuáles son los modelos que se centran en las operaciones y se implementan en algún sistema gestor de base de datos?

○ Modelo de datos conceptuales.

◉ Modelo de datos lógico.

○ Modelo de datos físicos.

Efectivamente, un ejemplo sería el modelo relacional.

2.- Terminología del modelo relacional.

¿Sabes que el modelo relacional te va a permitir representar la información del mundo real de una manera intuitiva? Así es, 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. Como hemos visto, se trata de un modelo lógico que establece una estructura sobre los datos, independientemente del modo en que luego los almacenemos. Es como si guardamos nuestra colección de libros, dependiendo del número de habitaciones que tenga en casa, del tamaño y forma de nuestras estanterías, podremos disponer nuestros libros de un modo u otro para facilitarnos el acceso y consulta. Los libros serán los mismos pero puedo disponerlos de distinta forma.

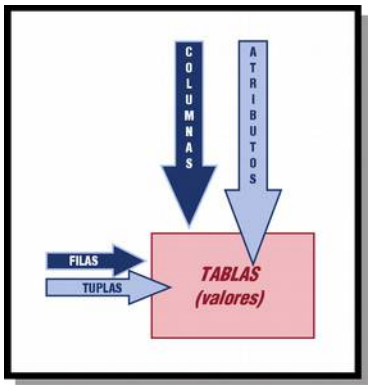
El nombre de modelo relacional viene de la estrecha relación entre el elemento básico de este modelo y el concepto matemático de relación. Si tenemos dos conjuntos A y B, una relación entre estos dos conjuntos sería un subconjunto del producto cartesiano AxB.

El producto cartesiano nos dará la relación de todos los elementos de un conjunto con todos los elementos de los otros conjuntos de ese producto. Al estar trabajando con conjuntos, no puede haber elementos repetidos.

2.1.- Relación o tabla. Tuplas. Dominios.

Pero... ¿qué es eso de “relación”? Hemos dicho que el modelo relacional se basa en el concepto matemático de relación, ya que Codd, que era un experto matemático, utilizó una terminología perteneciente a las matemáticas, en concreto a la teoría de conjuntos y a la lógica de predicados.

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 cumplir que:
  - Cada tupla se debe corresponder con un elemento del mundo real.
  - No puede haber dos tuplas iguales (con todos los valores iguales).

Está claro que 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 de dato 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 y 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 €.

Autoevaluación

¿Cuáles de las siguientes afirmaciones son ciertas sobre las tuplas y los atributos?

☒ Las tuplas deben corresponderse con un elemento del mundo real.

☐ Podríamos tener dos o más tuplas iguales.

☒ Un atributo se define en un dominio de valores.

☒ El nombre de cada dato que se almacena en la relación se denomina Atributo.

2.2.- Grado. Cardinalidad.

Ya 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.

Vamos a verlo con un ejemplo. Relación de grado 3, sobre los dominios A={Carlos, María}, B={Matemáticas, Lengua}, C={Aprobado, Suspenso}. {Nombre, Asignatura, Nota}. Las posibles relaciones que obtenemos al realizar el producto cartesiano AxBxC es el siguiente:

Producto Cartesiano AxBxC.

A={Carlos, María}	B={Matemáticas, Lengua}	C={Aprobado, Suspenso}
CARLOS	MATEMÁTICAS	APROBADO
CARLOS	MATEMÁTICAS	SUSPENSO
CARLOS	LENGUA	APROBADO
CARLOS	LENGUA	SUSPENSO
CARLOS	INGLÉS	APROBADO
CARLOS	INGLÉS	SUSPENSO
MARÍA	MATEMÁTICAS	APROBADO
MARÍA	MATEMÁTICAS	SUSPENSO
MARÍA	LENGUA	APROBADO
MARÍA	LENGUA	SUSPENSO
MARÍA	INGLÉS	APROBADO
MARÍA	INGLÉS	SUSPENSO

Si cogemos un subconjunto de ésta con 5 filas, tendríamos una relación de cardinalidad 5:

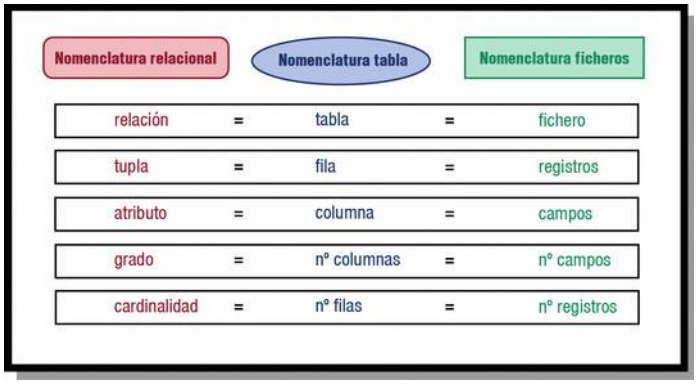
Subconjunto del Producto Cartesiano AxBxC con cardinalidad 5.

A={Carlos, María}	B={Matemáticas, Lengua}	C={Aprobado, Suspenso}
CARLOS	MATEMÁTICAS	APROBADO
CARLOS	LENGUA	APROBADO
CARLOS	INGLÉS	APROBADO
MARÍA	MATEMÁTICAS	APROBADO
MARÍA	INGLÉS	SUSPENSO

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 registros**: FICHEROS-REGISTROS-CAMPOS-NÚM CAMPOS- NÚM REGISTROS.



Autoevaluación

Relaciona cada término del modelo relacional con la terminología de Tablas.

Modelo relacional.	Relación.	Tablas.
RELACIÓN	5	1. COLUMNAS
TUPLA	4	2. NÚM COLUMNAS
ATRIBUTO	1	3. NÚM FILAS
GRADO	2	4. FILA
CARDINALIDAD	3	5. TABLA

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**. Se debe procurar que el nombre sea representativo de los datos que contiene.
- Como hemos visto antes, cada atributo (columna) de la tabla **toma un solo valor** en cada tupla (fila). Diferentes valores de atributo originarán diferentes filas.
- 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**.
- El **orden** de las tuplas (filas) **no importa**.

- El **orden** de los atributos (columnas) **no importa**.
- Todos los datos de un atributo (columna) deben ser del **mismo dominio**.

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

a	20
b	30
c	70

=

a	20
c	70
b	30

a	20
b	30
c	70

=

20	a
70	c
30	b

Autoevaluación

¿Cuál de las siguientes afirmaciones no es cierta en una relación?

○ Todos los atributos deben estar en el mismo dominio.

○ No puede haber dos tuplas completamente iguales.

○ Cada atributo de la tabla toma un único valor en cada tupla.

◉ Podemos tener tablas con el mismo nombre en la misma base de datos.

Correcto, sí es posible en bases de datos distintas.

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. Esta consulta puede ser sobre una sola tabla o sobre una relación entre varias tablas.
  - **Instantáneas**: son vistas (se crean de la misma forma) que sí almacenan los datos que muestran, además de la consulta que la creó. Sólo 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. Dependiendo del SGBD que usemos estas pueden ser de solo de lectura.
- **Temporales**: son tablas que son eliminadas automáticamente por el sistema.

Autoevaluación

Las relaciones que se crean indicando su estructura y sus ejemplares se denominan:

○ Instantáneas.

○ Vistas.

◉ Base.

4.- Tipos de datos.

¿Qué es un DNI? ¿Con qué datos lo representamos? DNI es una información que es susceptible de ser guardada. Normalmente el DNI está formado por dígitos y una letra al final. Si tuviéramos que clasificarlo diríamos que es un conjunto de caracteres alfanuméricos. ¿Y si pensamos en Sueldo? Aquí lo tenemos un poco más claro, evidentemente es un número entero o con decimales. (En ocasiones podremos encontrarnos con valores decimales guardados como números enteros, por ejemplo en un campo de "importe" 53.15 se guardaría como 5315 y 47 como 4700).

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 poseer un **Nombre** (relacionado con los datos que va a contener) y
- debe tener asociado un **Tipo de dato.**

Existen distintas formas de nombrar los tipos de datos dependiendo del lenguaje que utilicemos (C, Java, PHP, MySQL, SQL, Pascal, etc.).

Veamos cuales son los tipos de datos 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 cuales podemos realizar operaciones matemáticas.
- **Fecha/hora**: almacena fechas y horas.
- **Sí/No**: almacena datos que solo tienen dos posibilidades (verdadero/falso). También se denominan lógicos o booleanos.
- **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.

Por ejemplo, en la tabla Usuarios tenemos las siguientes superclaves:

- {Nombre, Apellidos, login, e_mail, F_nacimiento}
- {Nombre, Apellidos, login, e_mail}
- {login, e_mail}
- {login}

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**.
- Claves **alternativas.**
- Claves **ajenas**.

A continuación veremos cada una de ellas.

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.

Siguiendo con nuestro ejemplo, podríamos considerar los atributos Login o E_mail como claves candidatas, ya que sabemos que el Login debe ser único para cada usuario, a E_mail le sucede lo mismo. Pero también cabe la posibilidad de tomar: Nombre, Apellidos y F_nacimiento, las tres juntas como clave candidata.

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.

Si elegimos como clave candidata **Nombre, Apellidos y F_nacimiento**, cumple con la unicidad puesto que es muy difícil encontrarnos con dos personas que tengan el mismo nombre, apellidos y fecha de nacimiento iguales. Es irreducible puesto que sería posible encontrar dos personas con el mismo nombre y apellidos o con el mismo nombre y fecha de nacimiento, por lo que son necesarios los tres atributos (campos) para formar la clave.

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. Por ejemplo, podríamos pensar que Nombre y Apellidos podrían ser una clave candidata, pero ya sabemos que cabe la posibilidad de que dos personas puedan tener el mismo Nombre y Apellidos, así que lo descartamos.

Hasta ahora, seguimos teniendo 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 un 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. (Valores únicos que el sistema de la BBDD asigna automáticamente).

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

Si en nuestra tabla Usuarios escogemos Login como clave primaria, el E_mail o {Nombre, Apellidos, F_Nacimiento} serán nuestras claves alternativas.

Autoevaluación

Rellena los huecos con los conceptos adecuados.

Dentro del conjunto de superclaves, se llaman claves Rellenar huecos: `candidatas` a aquellas que identifican unívocamente a cada una de las Rellenar huecos : `tuplas` . De entre éstas, escogeremos la clave Rellenar huecos): `primaria` . Aquellas que no escogemos se denominarán claves Rellenar huecos: `alternativas` .

5.2.- Clave externa, ajena o secundaria.

Hasta ahora no nos hemos planteado cómo se relacionan unas tablas con otras dentro de una base de datos. Si tenemos las tablas Usuarios y Partidas, necesariamente habrá una "relación" entre ellas. Deben compartir algún dato en común que las relacione. Una partida es jugada por un jugador (Usuarios), por lo que en la tabla Partida deberíamos guardar algún dato del usuario-jugador, pero ¿cuál?

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.

En la tabla Partidas, se recogen datos como **Cod_partida**, **Fecha** y **Hora de creación**, **Nombre de la partida**, etc. ¿Qué campo utilizaremos para relacionarla con la tabla Usuarios? Si nos basamos en la definición, deberíamos utilizar la clave primaria de la tabla Usuarios. Por tanto, el atributo Login que es la clave principal en su tabla aparecerá en la tabla Partidas como clave ajena, externa o secundaria. El Login en Partidas hace referencia a cada jugador que juega esa partida. En lugar de guardar todos los datos de ese jugador en la misma tabla, lo hacemos en otra y lo "referenciamos" por su clave primaria tomándola como ajena. De esta manera evitamos duplicar datos que a la larga pueden provocar incoherencias en la Base de Datos.

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. En nuestro ejemplo, un mismo jugador puede jugar varias partidas.

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 ningún caso existirán claves ajenas que no se correspondan con una clave principal, la clave ajena representaría una referencia o conexión incorrecta.

No podemos tener una partida de un jugador que previamente no se ha registrado. Pero sí podemos tener los datos de una partida y desconocer el jugador de ésta.

Autoevaluación

¿Cuáles de las siguientes afirmaciones sobre las claves ajenas son correctas?

☒ Puede "referenciar" a la clave primaria de la misma tabla donde se encuentra.

☒ Puede "referenciar" a la clave primaria de otra tabla.

☒ Representa relaciones entre datos.

☒ Puede contener valores nulos.

☐ No puede repetirse en la tabla.

Para saber más

De las claves ajenas, foráneas, externas…
El modelo relacional no sería nada sin claves ajenas. *Foreign key*, lo encontraréis traducido de todas las formas posibles: ajena, externa, *aliena*, alienígena…

El modelo relacional no usa el tradicional e imprescindible puntero. Los punteros son para programadores, no para usuarios "normales", y el modelo relacional tiene vocación de "llegar a las masas". El puntero apunta a un dato concreto y sólo a uno, de hecho es una dirección física de memoria. En el modelo relacional, un valor "apunta" a todos los sitios donde se utilice ese mismo valor. La referencia se consigue por comparación, por igualdad. Entonces, ¿para qué queremos las claves ajenas? SQL, por ejemplo, hace como que no conoce ni claves ajenas ni primarias para resolver las consultas, somos nosotros los que de forma explícita debemos comparar los valores de esas columnas. Dos son, realmente, las funciones de las claves ajenas, no necesariamente en este orden:

1. asegurar la integridad referencial y
2. permitirnos modelar nuestro sistema de información a nuestro gusto.

Hablemos de esto último. La palabra clave es "simultaneidad": ¿de cuántas asignaturas puede estar matriculado un alumno en un momento dado? Y el marco temporal en el que responder a esta pregunta es "en cada estado de la base de datos". Si entendemos un estado de base de datos como una fotografía que le hacemos a la base de datos cada vez que se produce un cambio en ésta, estos cambios consisten en que habrá una fila más o una fila menos, o que un dato ha cambiado de valor, pero las columnas de cada tabla son siempre las mismas, y las reglas impuestas a cada una también.

La estructura de base de datos —el esquema— que diseñemos determinará la forma en que un usuario va a poder manejar la información. Si he matriculado al alumno A en la asignatura B, ¿puedo matricular a A también en C, o debo desmatricularlo antes de B? ¿Puedo dar de alta a un alumno sin matricularlo de ninguna asignatura? La respuesta a estas preguntas determinará dónde debo colocar la clave ajena —o claves— y decidir si esa clave ajena permite o no nulos y duplicados.

Supongamos la siguiente tabla ALUMNO —se supone que también existe una tabla ASIGNATURAS; en este estado de base de datos, tanto Pepe como Juan y Manolo, están matriculados de una asignatura:

21	Pepe	BD1
34	Juan	BD1
55	Manolo	AESI

En este otro estado de base de datos, aunque Pepe ha cambiado, todos siguen matriculados de una única asignatura:

21	Pepe	**FP2**
34	Juan	BD1
55	Manolo	AESI

Y siempre será así, la clave ajena, que se supone que es la columna con los acrónimos de asignaturas, está colocada ahí, en la propia tabla alumno, y nunca veremos a Pepe —en realidad, a "21″ que no lo hemos dicho, pero es su valor de clave primaria— con dos asignaturas al mismo tiempo.

Pero, ¿y si quiero que mis alumnos se puedan matricular de todas las asignaturas que quiera? Una solución sería una tabla adicional donde irían las claves ajenas.

ALUMNOS

21	Pepe
34	Juan
55	Manolo

MATRICULAS

21	BD1
21	FP2
34	BD1
55	AESI

Pepe, en este estado de base de datos y en cualquier otro, puede matricularse de tantas asignaturas como quiera. Y como él, cualquier alumno. Hemos cambiado el esquema de base de datos, hemos cambiado de sitio las claves ajenas. Tenemos otra base de datos, con otras reglas.

6.- Índices. Características.

Juan considera que es beneficioso crear un índice para la tabla Usuarios. Podría agilizar las búsquedas de usuarios registrados. Le ha contado a Ana que es conveniente tenerlo, aunque también le ha explicado que tener muchos índices no es bueno. Tendrán que hacer una buena elección del número de índices que van a manejar.

Imagina que estás creando un diccionario de términos informáticos. Podrías elegir la opción de escribirlo en una única hoja muy larga (estilo pergamino) o bien distribuirlo por hojas. Está claro que lo mejor sería distribuirlo por páginas. Y si buscamos el término "informática" en nuestro diccionario, podríamos comenzar a buscar en la primera página y continuar una por una hasta llegar a la palabra correspondiente. O bien crear un índice al principio, de manera que podamos consultar a partir de qué página podemos localizar las palabras que comienzan por "i". Esta última opción parece la más lógica.

Pues bien, en las bases de datos, cada tabla se divide internamente en páginas de datos, y se define el índice a través de un campo (o campos) y es a partir de este campo desde donde de busca.

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. Por ejemplo, si consultamos los usuarios cuya fecha de ingreso es anterior a una fecha concreta.

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. Por ejemplo, si la información de nuestra tabla Usuarios se desea consultar por apellidos, tiene sentido indexar por esa columna.

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. Detrás del valor NULO no hay nada, pero detrás del valor ESPACIO EN BLANCO está el valor ascii 32 y CERO al valor ascii 48.

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:

- VERDADERO Y (AND) NULO daría como resultado NULO.
- FALSO Y (AND) NULO daría como resultado FALSO.
- VERDADERO O (OR) NULO daría como resultado VERDADERO.
- FALSO O NULO daría como resultado NULO.
- NO (NOT) NULO daría como resultado 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.

Autoevaluación

¿Cuáles de las siguientes afirmaciones sobre el valor nulo son ciertas?

☒ Designa ausencia de dato.

☐ Es lo mismo que espacio en blanco.

☐ Es lo mismo que cero.

8.- Vistas.

Cuando vimos los distintos tipos de relaciones, aprendimos que, entre otros, estaban las vistas. Ahora ya tenemos más conocimientos para comprender mejor este concepto.

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.

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 tres 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**, como veremos 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.
- **Practicidad:** podemos obtener vistas con resultados que de otro modo sería muy costoso en términos de carga al SGBD.

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. No todos los gestores de bases de datos permiten actualizar vistas, Oracle, por ejemplo, no lo permite, mientras que SQL Server sí.

Autoevaluación

Una vista puede proceder de:

☒ Una tabla.

☒ Varias tablas.

☒ Otras vistas de la misma base de datos.

☒ Otras vistas de otras bases 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**: necesitará el permiso de sistema correspondiente. Por ejemplo para configurar los parámetros del sistemas.
- **Sobre objeto**: necesitará el permiso sobre el objeto en cuestión sobre una tabla para modificarla.

¿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.

Los Roles de los usuarios toman un papel muy impotante en los CMS (Sistemas Gestores de Contenidos) pues, a partir de ellos, se definen todas las interacciones de los usuarios y gestores con el entorno.

Autoevaluación

Al conjunto de permisos que se aplican a una conexión de base de datos, se le llama Rellenar huecos `usuario` . Los permisos dados a usuarios para que realicen ciertas operaciones se les llama Rellenar huecos `privilegios` . Si tengo una agrupación de permisos juntos, tenemos un Rellenar huecos `rol` .

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, inserción, modificación y elilminación de datos,
- 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.

Imagínate que cada programador utilizara sus propias reglas para escribir. Esto sería un caos. Es muy importante establecer los elementos con los que vamos a trabajar y unas normas que seguir.

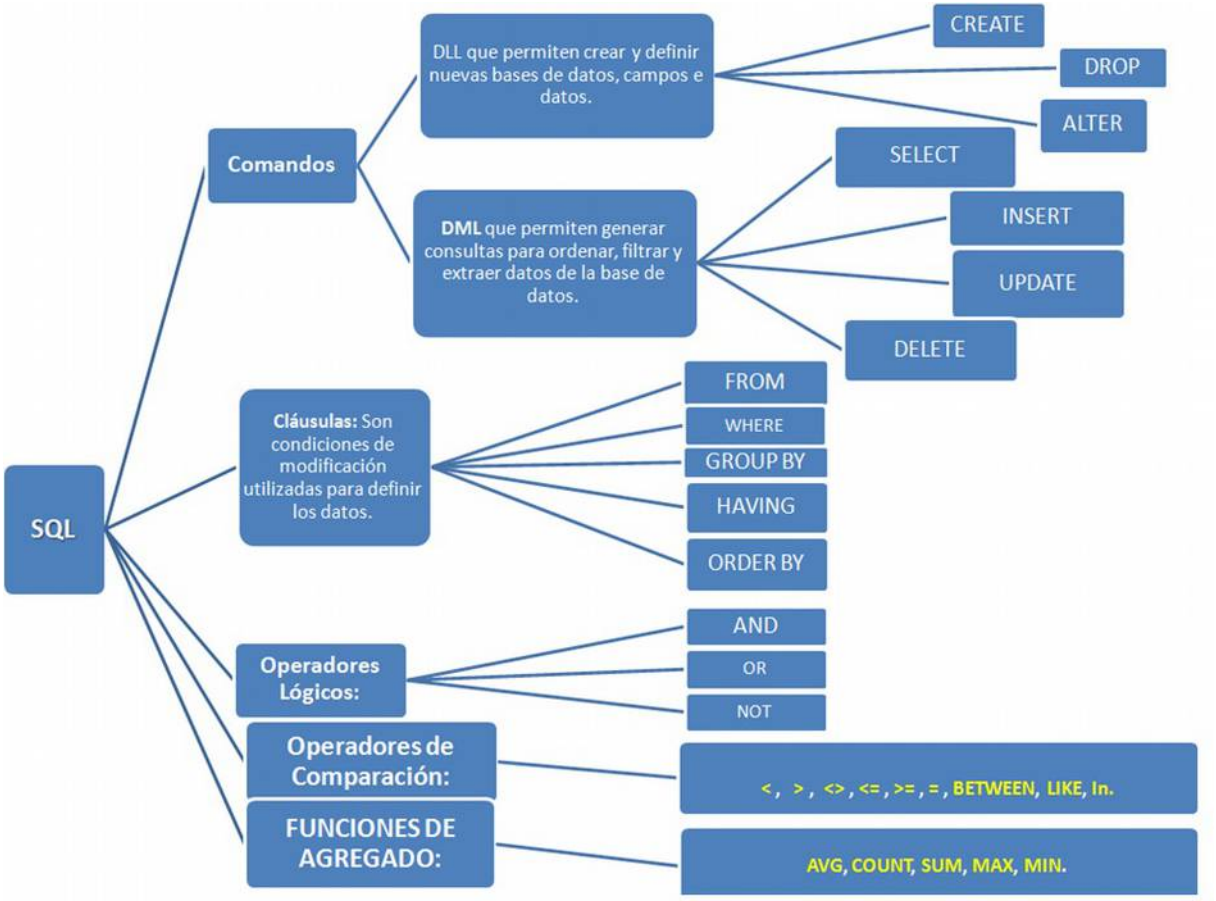
El lenguaje SQL está compuesto por comandos, cláusulas, operadores, funciones y literales . Todos estos elementos se combinan en las instrucciones y se utilizan para crear, actualizar y manipular bases de datos. Estos conceptos son bastante amplios por eso será mejor que vayamos por partes.

- **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, insertar, eliminar, 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.
- **OPERADORES**: Permiten crear expresiones complejas. Pueden ser aritméticos (+, -, *, /, ...) o lógicos (< , >, , < >, And, Or, etc.).
- **FUNCIONES:** Para conseguir valores complejos. Por ejemplo, la función promedio para obtener la media de un salario.
- **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).

Juan le ha dicho a Ana que es hora de ponerse a trabajar con la aplicación. Para aprender mejor le ha pedido permiso a Juan para instalar Oracle en su ordenador y así ir probando todo sobre la marcha para no cometer errores. El SQL estándar y el SQL de Oracle son bastante parecidos, pero con algunas diferencias.



Debes conocer

En el Anexo I encontraras aquellos comandos, cláusulas, operadores y funciones más generales con las que vamos a trabajar a lo largo del curso.

Para trabajar con Oracle tendrás que instalar el programa adecuado.

En el Anexo II puedes encontrar los pasos que debes seguir para la instalación de la aplicación en tu ordenador.

11.- Lenguaje de descripción de datos (DDL).

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 ésto 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;
```

Por ejemplo, a la base de datos que están creando Juan y Ana se le va a llamar RyMjuegos, entonces nos quedaría:

```
CREATE DATABASE RyMjuegos;
```

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.

¿Qué necesitamos para poder guardar los datos? Lo primero 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.

Esta primera etapa de análisis es vital y no debemos escatimar tiempo y recursos en ella pues una mala planificación puede convertirse en un gran problema a largo plazo.

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 deben usar letras del alfabeto inglés, dígitos o el signo de guión bajo, y aunque haya sistemas que lo permitan debemos evitarlos para evitar problemas exportando datos o interactuando con otros sistemas.
- 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.
- En el caso de que el nombre tenga espacios en blanco o caracteres nacionales (permitido sólo en algunas bases de datos), entonces se suele entrecomillar con comillas dobles. En el estándar SQL99 (respetado por Oracle) se pueden utilizar comillas dobles al poner el nombre de la tabla a fin de hacerla sensible a las mayúsculas (se diferenciará entre "USUARIOS"y "Usuarios").

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

```
CREATE TABLE [esquema.] nombredeTabla (
      columna1 Tipo_Dato,
      columna2 Tipo_Dato,
      ...
      columnaN Tipo_Dato );
```

donde:

- columna1, columna2, ..., columnaN son los nombres de las columna que contendrá la tabla.
- Tipo_Dato indica el tipo de dato de cada columna.

Ana va a crear la primera tabla llamada USUARIOS con un solo campo de tipo VARCHAR:

```
CREATE TABLE USUARIOS (Nombre VARCHAR(25));
```

Recuerda que solo podrás crear tablas si posees los permisos necesarios para ello.

Debes conocer

Durante nuestro aprendizaje vamos a tener que crear muchas tablas, para ello necesitaremos manejar los tipos de datos que utiliza Oracle. En el siguiente enlace tienes una relación de estos tipos y su descripción.

TIPO	CARACTERISTICAS	OBSERVACIONES
CHAR	Cadena de caracteres (alfanuméricos) de longitud fija	Entre 1 y 2000 bytes como máximo. Aunque se introduzca un valor más corto que el indicado en el tamaño, se rellenará al tamaño indicado. Es de longitud fija, siempre ocupará lo mismo, independientemente del valor que contenga
VARCHAR2	Cadena de caracteres de longitud variable	Entre 1 y 4000 bytes como máximo. El tamaño del campo dependerá del valor que contenga, es de longitud variable.
VARCHAR	Cadena de caracteres de longitud variable	En desuso, se utiliza VARCHAR2 en su lugar
NCHAR	Cadena de caracteres de longitud fija que sólo almacena caracteres Unicode	Entre 1 y 2000 bytes como máximo. El juego de caracteres del tipo de datos (datatype) NCHAR sólo puede ser AL16UTF16 ó UTF8. El juego de caracteres se especifica cuando se crea la base de datos Oracle
NVARCHAR2	Cadena de caracteres de longitud variable que sólo almacena caracteres Unicode	Entre 1 y 4000 bytes como máximo. El juego de caracteres del tipo de datos (datatype) NCHAR sólo puede ser AL16UTF16 ó UTF8. El juego de caracteres se especifica cuando se crea la base de datos Oracle
LONG	Cadena de caracteres de longitud variable	Como máximo admite hasta 2 GB (2000 MB). Los datos LONG deberán ser convertidos apropiadamente al moverse entre diversos sistemas. 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. No se puede utilizar en claúsulas WHERE, GROUP BY, ORDER BY, CONNECT BY ni DISTINCT Una tabla sólo puede contener una columna de tipo LONG. Sólo soporta acceso secuencial.
LONG RAW	Almacenan cadenas binarias de ancho variable	Hasta 2 GB. En desuso, se sustituye por los tipos LOB.
RAW	Almacenan cadenas binarias de ancho variable	Hasta 32767 bytes. En desuso, se sustituye por los tipos 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.
BLOB	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.

TIPO	CARACTERISTICAS	OBSERVACIONES
BFILE	Almacena datos binarios no estructurados en archivos del sistema operativo, fuera de la base de datos. Una columna BFILE almacena un localizador del archivo a uno externo que contiene los datos	Admiten hasta 8 terabytes. El administrador de la base de datos debe asegurarse de que exista el archivo en disco y de que los procesos de Oracle tengan permisos de lectura para el archivo .
ROWID	Almacenar la dirección única de cada fila de la tabla de la base de datos	ROWID físico almacena la dirección de fila en las tablas, las tablas en clúster, los índices, excepto en las índices-organizados (IOT). ROWID lógico almacena la dirección de fila en tablas de índice-organizado (IOT). Un ejemplo del valor de un campo ROWID podría ser: "AAAIugAAJAAC4AhAAI". El formato es el siguiente: Para "OOOOOOFFFBBBBBBRRR", donde: OOOOOO: segmento de la base de datos (AAAIug en el ejemplo). Todos los objetos que estén en el mismo esquema y en el mismo segmento tendrán el mismo valor. FFF: el número de fichero del tablespace relativo que contiene la fila (fichero AAJ en el ejemplo). BBBBBB: el bloque de datos que contiene a la fila (bloque AAC4Ah en el ejemplo). El número de bloque es relativo a su fichero de datos, no al tablespace. Por lo tanto, dos filas con números de bloque iguales podrían residir en diferentes datafiles del mismo tablespace. RRR: el número de fila en el bloque (fila AAI en el ejemplo). Este tipo de campo no aparece en los SELECT ni se puede modificar en los UPDATE, ni en los INSERT. Tampoco se puede utilizar en los CREATE. Es un tipo de datos utilizado exclusivamente por Oracle. Sólo se puede ver su valor utilizando la palabra reservada ROWID, por ejemplo: select rowid, nombre, apellidos from clientes Ejemplo 2: SELECT ROWID, SUBSTR(ROWID,15,4) "Fichero", SUBSTR(ROWID,1,8) "Bloque", SUBSTR(ROWID,10,4) "Fila" FROM proveedores Ejemplo 3: una forma de saber en cuántos ficheros de datos está alojada una tabla: SELECT COUNT(DISTINCT(SUBSTR(ROWID,7,3))) "Numero ficheros " FROM facturacion
UROWID	ROWID universal	Admite ROWID a tablas que no sean de Oracle, tablas externas. Admite tanto ROWID lógicos como físicos.
NUMBER	Almacena números fijos y en punto flotante	Se admiten hasta 38 dígitos de precisión y son portables a cualquier entre los diversos sistemas en que funcione Oracle. Para declarar un tipo de datos NUMBER en un CREATE ó UPDATE es suficiente con: nombre_columna NUMBER opcionalmente se le puede indicar la precisión (número total de dígitos) y la escala (número de dígitos a la derecha de la coma, decimales, los cogerá de la precisión indicada): nombre_columna NUMBER (precision, escala) Si no se indica la precisión se tomará en función del número a guardar, si no se indica la escala se tomará escala cero. Para no indicar la precisión y sí la escala podemos utilizar: nombre_columna NUMBER (*, escala)

TIPO	CARACTERISTICAS	OBSERVACIONES
		Para introducir números que no estén el el formato estándar de Oracle se puede utilizar la función TO_NUMBER.
FLOAT	Almacena tipos de datos numéricos en punto flotante	Es un tipo NUMBER que sólo almacena números en punto flotante
DATE	Almacena un punto en el tiempo (fecha y hora)	El tipo de datos DATE almacena el año (incluyendo el siglo), el mes, el día, las horas, los minutos y los segundos (después de medianoche). Oracle utiliza su propio formato interno para almacenar fechas. Los tipos de datos DATE se almacenan en campos de longitud fija de siete octetos cada uno, correspondiendo al siglo, año, mes, día, hora, minuto, y al segundo. Para entrada/salida de fechas, Oracle utiliza por defecto el formato DD-MMM-AA. Para cambiar este formato de fecha por defecto se utiliza el parámetro NLS_DATE_FORMAT. Para insertar fechas que no estén en el mismo formato de fecha estándar de Oracle, se puede utilizar la función TO_DATE con una máscara del formato: TO_DATE (el "13 de noviembre de 1992", "DD del MES, YYYY")
TIMESTAMP	Almacena datos de tipo hora, fraccionando los segundos	
TIMESTAMP WITH TIME ZONE	Almacena datos de tipo hora incluyendo la zona horaria (explícita), fraccionando los segundos	
TIMESTAMP WITH LOCAL TIME ZONE	Almacena datos de tipo hora incluyendo la zona horaria local (relativa), franccionando los segundos	Cuando se usa un SELECT para mostrar los datos de este tipo, el valor de la hora será ajustado a la zona horaria de la sesión actual
XMLType	Tipo de datos abstracto. En realidad se trata de un CLOB.	Se asocia a un esquema XML para la definición de su estructura.

De los tipos anteriores, los comunmente utilizados son: VARCHAR2 (cadenas de texto no muy grandes), DATE (fechas, horas), NUMBER (números), BLOB (ficheros de tipo word, excel, access, video, sonido, imágenes, etc) y CLOB (cadenas de texto muy grandes).

Autoevaluación

Señala cuales de las siguientes afirmaciones sobre los nombres de las tablas son ciertas:

☐ Puede haber nombres de tablas duplicados en la misma base de datos.

☒ Su longitud máxima es de 30 caracteres.

☒ La tabla JUEGOS es la misma que la tabla Juegos.

☒ No puede coincidir con las palabras reservadas de SQL.

Si incluimos comillas dobles al nombre de la tabla entonces si diferenciamos entre mayúsculas y minúsculas.

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.

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). Es buena idea incluir de algún modo el nombre de la tabla, los campos involucrados y el tipo de restricción en el nombre de la misma. La sintaxis en SQL estándar es la siguiente:

```
CREATE TABLE NOMBRETABLA (
     Columna1 Tipo_Dato
          [CONSTRAINT nombredelarestricción]
          [NOT NULL]
          [UNIQUE]
          [PRIMARY KEY]
          [FOREIGN KEY]
          [DEFAULT valor]
          [REFERENCES nombreTabla [(columna [, columna ])]
          [ON DELETE CASCADE]]
          [CHECK condición],
     Columna2 Tipo_Dato
          [CONSTRAINT nombredelarestricción]
          [NOT NULL]
          [UNIQUE]
          [PRIMARY KEY]
          [FOREIGN KEY]
          [DEFAULT valor]
          [REFERENCES nombreTabla [(columna [, columna ])]
          [ON DELETE CASCADE]]
          [CHECK condición],...);
```

Veamos un ejemplo:

```
CREATE TABLE USUARIOS (
    Login VARCHAR(15) CONSTRAINT usu_log_PK PRIMARY KEY,
    Password VARCHAR (8) NOT NULL,
    Fecha_Ingreso DATE DEFAULT SYSDATE);
```

Otra opción es definir las columnas de la tabla y después especificar las restricciones, de este modo podrás referir varias columnas en una única restricción.

En los siguientes apartados veremos cada una de las restricciones, su significado y su uso.

Recomendación

Oracle nos 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).

11.3.1.- 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.

Si en la tabla USUARIOS queremos que el campo "F_Nacimiento" sea obligatorio ponerlo, nos quedaría así:

```
CREATE TABLE USUARIOS (
    F_Nacimiento DATE
    CONSTRAINT Usu_Fnac_NN NOT NULL);
```

o bien, de esta otra forma:

```
CREATE TABLE USUARIOS (
    F_Nacimiento DATE NOT NULL);
```

Debemos tener cuidado con los valores nulos en las operaciones, ya que 1*NULL es igual a NULL.

Autoevaluación

Si queremos que un campo no admita valores nulos, al crear la tabla pondremos después del nombre del campo y del tipo de datos:

○ `NULL.`

○ `VARCHAR.`

◉ `NOT NULL.`

11.3.2.- 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. Es interesante de cara a crear claves primarias automaticamente para cada registro.

También para esta restricción tenemos dos posibles formas de ponerla, veámoslo con un ejemplo. Supongamos que el campo Login de nuestra tabla va a ser único. Lo incluiremos en la tabla que estamos creando. Nos quedaría así:

```
CREATE TABLE USUARIOS (
    Login VARCHAR2 (25)
        CONSTRAINT Usu_Log_UK UNIQUE);
```

Veamos otra forma:

```
CREATE TABLE USUARIOS (
    Login VARCHAR2 (25) UNIQUE);
```

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

```
CREATE TABLE USUARIOS (
    Login VARCHAR2 (25),
    Correo VARCHAR2 (25),
        CONSTRAINT Usuario_UK UNIQUE (Login, Correo));
```

Si te fijas, 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.

11.3.3.- 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 como quedaría la si la clave fuese el campo Login:

- Si la clave la forma un único campo:

```
CREATE TABLE USUARIOS (
        Login VARCHAR2 (25) PRIMARY KEY);
```

- O bien poniendo un nombre a la restricción:

```
CREATE TABLE USUARIOS (
             Login VARCHAR2 (25)
                  CONSTRAINT Usu_log_PK PRIMARY KEY);
```

- Si la clave está formada por más de un campo, por ejemplo Nombre, Apellidos y Fecha de Nacimiento:

```
CREATE TABLE USUARIOS (
        Nombre VARCHAR2 (25),
        Apellidos VARCHAR2 (30),
        F_Nacimiento DATE,
        CONSTRAINT Usu_PK PRIMARY KEY(Nombre, Apellidos,
                  F_Nacimiento));
```

11.3.4.- 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 (o incluso con la clave candidata) 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.

En nuestra tabla vamos a tener una clave ajena procedente de la tabla PARTIDAS que será su Cod_Partida, por tanto tendremos que hacer referencia a éste:

```
CREATE TABLE USUARIOS (
   Cod_Partida NUMBER(8)
       CONSTRAINT Cod_Part_FK
       REFERENCES PARTIDAS(Cod_Partida));
```

Si el campo al que hace referencia es clave principal en su tabla no es necesario indicar el nombre del campo:

```
CREATE TABLE USUARIOS (
    Cod_Partida NUMBER(8)
        CONSTRAINT Cod_Part_FK
        REFERENCES PARTIDAS);
```

Si la definición de la clave ajena se pone al final, tendremos que colocar el texto `FOREIGN KEY` para especificar a qué campo se está refiriendo.

Vamos a verlo en el caso en que la clave ajena estuviera formada por Cod_Partida y Fecha de la partida de la tabla PARTIDAS:

```
CREATE TABLE USUARIOS (
    Cod_Partida NUMBER(8),
    F_Partida DATE,
        CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida, F_Partida)
        REFERENCES PARTIDAS);
```

Al relacionar campos necesitamos que el dato del campo que es clave ajena en una tabla (que llamaremos secundaria) previamente haya sido incluido en su tabla de procedencia donde es clave primaria o candidata. En nuestro ejemplo, cualquier código de partida que incluyamos en la tabla USUARIO, debería estar previamente en la tabla de la que procede, es decir, en la tabla PARTIDAS. **A esto se le llama Integridad Referencial.**

Esto puede crear algunos errores, pues puede ocurrir lo siguiente:

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

11.3.5.- Restricción DEFAULT Y VALIDACIÓN.

A veces es muy tedioso insertar siempre lo mismo en un campo. Imagínate que casi todos los jugadores fuesen de España y tenemos un campo País. ¿No sería cómodo asignarle un valor por defecto? Eso es lo que hace la restricción `DEFAULT`.

En nuestro ejemplo vamos a añadir a la tabla USUARIOS el campo País y le daremos por defecto el valor "España".

```
CREATE TABLE USUARIOS (
    Pais VARCHAR2(20) DEFAULT ' España ' );
```

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 USUARIOS (
      Fecha_ingreso DATE DEFAULT SYSDATE);
```

También vamos a necesitar que se compruebe que los valores que se introducen son adecuados para ese campo. Para ello utilizaremos `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 la 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.

Debes conocer

Si queremos obtener una descripción de una tabla, sinonimo, paquete o función, podemos utilizar el comando `DESCRIBE.`

Autoevaluación

Relaciona estos términos utilizados para las restricciones en la creación de tablas con su significado o función:

Términos.	Relación.	Función.
`CHECK`	1	1. Comprueba que los valores que se introducen son los adecuados para un campo.
`DEFAULT`	6	2. Designa a un campo como clave ajena.
`PRIMARY KEY`	5	3. Impide que un campo pueda contener valores nulos
`FOREIGN KEY`	2	4. Impide que se repitan valores para un campo.
`NOT NULL`	3	5. Designa a un campo como clave principal.
`UNIQUE`	4	6. Incluye un valor en un campo de forma predeterminada.

11.4.- Eliminación 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`.

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

Esta instrucción borrará la tabla de la base de datos incluido sus datos (filas). También se borrará toda la información que existiera de esa tabla en el Diccionario de Datos.

La opción `CASCADE CONSTRAINTS` se puede incluir para los casos en que alguna de las columnas sea clave ajena en otra tabla secundaria, lo que impediría su borrado. Al colocar esta opción las restricciones donde es clave ajena se borrarán antes las filas de tablas secundarias que hagan referencia a los valores de la tabla que se va a eliminar y, a continuación, se eliminará la tabla en cuestión.

Vamos a eliminar la tabla con la que hemos estado trabajando:

```
DROP TABLE USUARIOS ;
```

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

Al borrar una tabla:

- Desaparecen todos sus datos
- Cualquier vista asociada a esa tabla seguirá existiendo pero ya no funcionará.

También existe la orden `TRUNCATE TABLE` que te permitirá eliminar los datos (filas) de una tabla sin eliminar su estructura. Es muy útil de cara al manejo de tablas temporales.Y recuerda que solo podrás borrar aquellas tablas sobre las que tengas permiso de borrado.

11.5.- Modificación de tablas (I).

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. ¿Es posible esto? Ahora veremos que sí y en casi todos los casos utilizaremos el comando ALTER TABLE.

- **Si queremos cambiar el nombre de una tabla:**

```
RENAME NombreViejo TO NombreNuevo;
```

- **Si queremos añadir columnas a una tabla: l**as columnas se añadirán al final de la tabla.

```
ALTER TABLE NombreTabla ADD
( ColumnaNueva1 Tipo_Datos [Propiedades]
[, ColumnaNueva2 Tipo_Datos [Propiedades]
... );
```

- **Si queremos eliminar columnas de una tabla:** se eliminará la columna indicada sin poder deshacer esta acción. Además de la definición de la columna, se eliminarán todos los datos que contuviera. No se puede eliminar una columna si es la única que forma la tabla, para ello tendremos que borrar la tabla directamente.

```
ALTER TABLE NombreTabla DROP COLUMN (Columna1 [, Columna2, ...]);
```

- **Si queremos modificar columnas de una tabla:** podemos modificar el tipo de datos y las propiedades de una columna. Todos los cambios son posibles si la tabla no contiene datos. En general, si la tabla no está vacía podremos aumentar la longitud de una columna, aumentar o disminuir en número de posiciones decimales en un tipo NUMBER, reducir la anchura siempre que los datos no ocupen todo el espacio reservado para ellos.

```
 ALTER TABLE NombreTabla MODIFY
(Columna1 TipoDatos [propiedades] [, columna2 TipoDatos
[propiedades] ...] );
```

- **Si queremos renombrar columnas de una tabla:**

```
ALTER TABLE NombreTabla RENAME COLUMN NombreAntiguo TO
NombreNuevo;
```

Tenemos la siguiente tabla creada:

```
CREATE TABLE USUARIOS (
   Credito NUMBER(4) CHECK (Crédito BETWEEN 0 AND 2000));
```

Nos gustaría incluir una nueva columna llamada User que será tipo texto y clave primaria:

```
ALTER TABLE USUARIO ADD
    (User VARCHAR(10) PRIMARY KEY);
```

Nos damos cuenta que ese campo se llamaba Login y no User, vamos a cambiarlo:

```
ALTER TABLE USUARIO RENAME COLUMN User TO Login;
```

Ejercicio resuelto

Tenemos creada la siguiente tabla:

```
CREATE TABLE EMPLEADOS (
    Cod_Cliente VARCHAR(5) PRIMARY KEY,
    Nombre VARCHAR(10),
    Apellidos VARCHAR(25),
    Sueldo NUMBER(2));
```

Ahora queremos poner una restricción a sueldo para que tome valores entre 1000 y 1200:

```
ALTER TABLE EMPLEADOS MODIFY (Sueldo NUMBER(2) CHECK (Sueldo BETWEEN
1000 AND 1200));
```

11.5.1.- Modificación de tablas (II).

Utilizando el comando `ALTER TABLE`, podemos modificar las restricciones o bien eliminarlas:

- **Si queremos borrar restricciones:**

  ```
  ALTER TABLA NombreTabla DROP CONSTRAINT NombreRestriccion;
  ```

- **Si queremos modificar el nombre de las restricciones:**

  ```
  ALTER TABLE NombreTabla RENAME CONSTRAINT NombreViejo TO
      NombreNuevo;
  ```

- **Si queremos activar o desactivar restricciones:**

  A veces es conveniente desactivar temporalmente una restricción para hacer pruebas o porque necesitemos saltarnos esa regla. Para ello usaremos esta sintaxis:

  ```
  ALTER TABLE NombreTabla DISABLE CONSTRAINT NombreRestriccion
  [CASCADE];
  ```

  La opción `CASCADE` desactiva las restricciones que dependan de ésta.

  Para activar de nuevo la restricción:

  ```
  ALTER TABLE NombreTabla ENABLE CONSTRAINT NombreRestriccion
  [CASCADE];
  ```

-

Debes conocer

Puede ocurrir que no hayamos puesto nombre a las restricciones o bien que lo hiciéramos pero no lo recordemos. Sería interesante que se pudiera consultar en algún lado.

Las restricciones o **Constraints** son un mecanismo de seguridad y proteccion en las bases de datos por ejemplo para evitar insertar valores nulos en determinados campos de la tabla o evitar la duplicidad erronea de filas. Si defines una clave primaria en una tabla, se activa la constraint correspondiente de manera que si intentas insertar una nueva fila cuya clave primaria coincide con una ya existente, salta la constraint correspondiente y el consiguiente error de oracle (o la base de datos que estes utilizando) avisandote de ello.

Si tienes un modelo de datos muy grande, que maneje muchas constraints es posible que en la depuracion de un programa exista un error de base de datos y conozcas el nombre de la constraint, pero no su regla (si es clave primaria, si es clave foranea, si la longitud debe ser mayor que un cierto valor, si el campo no debe ser null, etc).
Para facilitarte la busqueda de su significado tal vez te sea de interes la siguiente vista del diccionario de datos:

```
SELECT * FROM all_constraints
```

Apareceran todas las constraints del modelo (las tablas accesibles) y ya podras filtrar por nombre de la tabla (donde aplica la restriccion) o nombre de la constraint. *Bueno, bonito y barato…*

11.6.- Creación y eliminación de índices

Sabemos que crear índices ayuda a la localización más rápida de la información contenida en las tablas. Ahora aprenderemos a crearlos y eliminarlos:

```
CREATE INDEX NombreIndice ON NombreTabla (Columna1 [, Columna2 ...]);
```

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 indices 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.

Para eliminar un índice es suficiente con poner la instrucción:

```
DROP INDEX NombreIndice;
```

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

Ejercicio resuelto

Tenemos creada la siguiente tabla:

```
CREATE TABLE EMPLEADOS (
    Cod_Cliente VARCHAR(5) PRIMARY KEY,
    Nombre VARCHAR(10),
    Apellidos VARCHAR(25),
    Sueldo NUMBER(2));
```

Crea un índice con el campo Apellidos, luego elimínalo.

```
CREATE INDEX miIndice ON EMPLEADOS (Apellidos);
DROP INDEX miIndice;
```

12.- Lenguaje de control de datos (DCL).

Ya hemos visto que necesitamos una cuenta de usuario para acceder a los datos de una base de datos. Las claves de acceso se establecen cuando se crea el usuario y pueden ser modificados por el Administrador o por el propietario de dicha clave. La Base de Datos almacena encriptadas las claves en una tabla del diccionario llamada DBA_USERS.

¿Cómo se crean los usuarios? La sintaxis es:

```
CREATE USER NombreUsuario
IDENTIFIED BY ClaveAcceso
[DEFAULT TABLESPACE tablespace ]
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {K | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil];
```

donde:

- `CREATE USER`: crea un nombre de usuario que será identificado por el sistema.
- `IDENTIFIED BY`: permite dar una clave de acceso al usuario creado.
- `DEFAULT TABLESPACE`: asigna a un usuario el Tablespace por defecto para almacena los objetos que cree. Si no se asigna ninguna, será `SYSTEM`.
- `TEMPORARY TABLESPACE`: especifica el nombre del Tablespace para trabajos temporales. Por defecto será `SYSTEM`.
- `QUOTA`: asigna un espacio en Megabytes o Kilobytes en el Tablespace asignado. Si no se especifica el usuario no tendrá espacio y no podrá crear objetos.
- `PROFILE`: asigna un perfil al usuario. Si no se especifica se asigna el perfil por defecto.

Recuerda que para crear usuarios debes tener una cuenta con privilegios de Administrador.

Para ver todos los usuarios creados utilizamos las vistas ALL_USERS y DBA_USERS. Y para ver en mi sesión los usuarios que existen pondría: `DESC SYS.ALL_USERS;`

Practiquemos un poco con este comando. Creemos una cuenta de usuario limitado, que no tenga derecho ni a guardar datos ni a crear objetos, más tarde le daremos permisos:

```
CREATE USER UsuarioLimitado IDENTIFIED BY passworddemiusuariolimitado ;
```

Podemos modificar usuarios mediante el comando ALTER USER, cuya sintaxis es la siguiente:

```
ALTER USER NombreUsuario
IDENTIFIED BY clave_acceso
[DEFAULT TABLESPACE tablespace ]
[TEMPORARY TABLESPACE tablespace]
[QUOTA int {K | M} ON tablespace]
[QUOTA UNLIMITED ON tablespace]
[PROFILE perfil];
```

Un usuario sin privilegios de Administrador únicamente podrá cambiar su clave de acceso. Debemos de tener en cuenta que a la hora de modificar una contraseña, si el campo esta está encriptado, el contenido de este campo ha de insertarse ya encriptado en la base de datos.

Para eliminar o borrar un usuario utilizamos el comando `DROP USER` con la siguiente sintaxis:

```
DROP USER NombreUsuario [CASCADE];
```

La opción `CASCADE` borra todos los objetos del usuario antes de borrarlo. Sin esta opción no nos dejaría eliminar al usuario si éste tuviera tablas creadas.

12.1.- Permisos (I).

Ningún usuario puede llevar a cabo una operación si antes no se le ha concedido el permiso para ello. En el apartado anterior hemos creado un usuario para iniciar sesión, pero si con él intentáramos crear una tabla veríamos que no tenemos permisos suficientes para ello.

Para poder acceder a los objetos de una base de datos necesitas tener privilegios (permisos). Éstos se pueden agrupar formando **roles**, lo que simplificará la administración. Los roles pueden activarse, desactivarse o protegerse con una clave. Mediante los roles podemos gestionar los comandos que pueden utilizar los usuarios. Un permiso se puede asignar a un usuario o a un rol.

Básicamente los diferentes permisos que podemos asignar en una base de datos a la hora de trabajar sobre registros son:

- `SELECT:` Permisos de selección, permite seleccionar registros de una tabla y visualizarlos
- `INSERT:` Permisos de inserción, permite insertar o añadir registros a una tabla
- `UPDATE:` Permisos de actualización, permite modificar registros de una tabla
- `DELETE:` Permisos de borrado, permite eliminar registros de una tabla

También podremos adjudicar permisos sobre objetos, con los cuales se podrán crear , modificar y eliminar tablas, vistas , informes…

Un privilegio o permiso se especifica con el comando `GRANT` (conceder).

Si se dan privilegios **sobre los objetos**:

```
GRANT {privilegio_objeto [, privilegio_objeto]...|ALL|[PRIVILEGES]}
ON [usuario.]objeto
FROM {usuario1|rol1|PUBLIC} [,{usuario2|rol2|PUBLIC] ...
[WITH GRANT OPTION];
```

- `ON` especifica el objeto sobre el que se conceden los privilegios.
- `TO` señala a los usuarios o roles a los que se conceden privilegios.
- `ALL` concede todos los privilegios sobre el objeto especificado.
- `[WITH GRANT OPTION]` permite que el receptor del privilegio se lo asigne a otros.
- `PUBLIC` hace que un privilegio esté disponible para todos los usuarios.

En el siguiente ejemplo Juan ha accedido a la base de datos y ejecuta los siguientes comandos:

- `GRANT INSERT TO Usuarios TO Ana;` (permitirá a Ana insertar datos en la tabla Usuarios)
- `GRANT ALL ON Partidas TO Ana;` (Juan concede todos los privilegios sobre la tabla Partidas a Ana)

Los privilegios **de sistema** son los que dan derecho a ejecutar comandos SQL o acciones sobre objetos de un tipo especificado. Existen gran cantidad de privilegios distintos.

La sintaxis para dar este tipo de privilegios la tienes aquí:

```
GRANT {Privilegio1 | rol1 } [, privilegio2 | rol2}, ...]
TO {usuario1 | rol1| PUBLIC} [, usuario2 | rol2 | PUBLIC} ... ]
[WITH ADMIN OPTION];
```

- `TO` señala a los usuarios o roles a los que se conceden privilegios.
- `WITH ADMIN OPTION` es una opción que permite al receptor de esos privilegios que pueda conceder esos mismos privilegios a otros usuarios o roles.
- `PUBLIC` hace que un privilegio esté disponible para todos los usuarios.

Veamos algunos ejemplos:

```
GRANT CONNECT TO Ana;
```

Concede a Ana el rol de `CONNECT` con todos los privilegios que éste tiene asociados.

```
GRANT DROP USER TO Ana WITH ADMIN OPTION;
```

Concede a Ana el privilegio de borrar usuarios y que ésta puede conceder el mismo privilegio de borrar usuarios a otros.

Para saber más

Si quieres conocer más sobre permisos y objetos sobre los que se conceden privilegios, lee lo siguiente:

Conceptos de gestión de privilegios y recursos	
Concepto	**Significado**
Privilegio	Permiso para realizar una acción, asignable a un usuario o un rol
Rol	Conjunto de privilegios, asignables a un usuario o un rol
Usuario	Colección de objetos y privilegios identificado con un nombre y password
Perfil	Conjunto de restricciones relativas al uso de recursos, y asignable a usuarios. Un usuario sólo puede tener un perfil
Recurso	Uso susceptible de ser restringido, asignable a un perfil
Roles Predefinidos por Oracle (select * from dba_roles;)	
CONNECT	
RESOURCE	
DBA	
EXP_FULL_DATABASE	
IMP_FULL_DATABASE	
DELETE_CATALOG_ROLE	
EXECUTE_CATALOG_ROLE	
SELECT_CATALOG_ROLE	
Recursos en Oracle (select * from user_resource_limits;)	
COMPOSITE_LIMIT	
SESSIONS_PER_USER	
CPU_PER_SESSION	
CPU_PER_CALL	
LOGICAL_READS_PER_SESSION	
LOGICAL_READS_PER_CALL	
IDLE_TIME	
CONNECT_TIME	
PRIVATE_SGA	
Limites en uso del espacio en disco (select * from dba_ts_quotas;)	
ALTER USER SCOTT QUOTA UNLIMITED ON USER_DATA;	
ALTER USER SCOTT QUOTA 5M ON TEMPORARY_DATA;	

ALTER USER SCOTT QUOTA 0 ON SYSTEM;

Para la gestión de	se utilizan los comandos
Privilegios	GRANT (conceder un privilegio a un usuario o a un rol) REVOKE (denegar un privilegio a un usuario o a un rol)
Roles	CREATE ROLE (crear) ALTER ROLE (modificar) DROP ROLE (borrar) SET ROLE (activar, desactivar) GRANT (conceder un permiso o un rol a un rol) REVOKE (denegar un permiso o un rol a un rol)
Usuarios	CREATE USER (crear) ALTER USER (modificar) DROP USER (borrar) GRANT (conceder un permiso o un rol a un usuario) REVOKE (denegar un permiso o un rol a un usuario)
Perfiles y Recursos	CREATE PROFILE (crear) ALTER PROFILE (modificar) DROP PROFILE (borrar) ALTER USER ... PROFILE (asignar a un usuario) CREATE USER ... PROFILE (asignar a un usuario)

Activación de perfiles

Estado de la Base de Datos	Acción
Base de Datos Parada	RESOURCE_LIMIT = TRUE en c:\orant\database\initorcl.ora
Base de Datos Arrancada	ALTER SYSTEM SET RESOURCE_LIMIT = TRUE;

12.1.1.- Permisos (II).

Hasta ahora hemos aprendido a conceder permisos o privilegios. Será importante aprender a retirarlos:

Con el comando `REVOKE` se retiran los privilegios:

- Sobre **objetos**:

```
REVOKE {privilegio_objeto [, privilegio_objeto]...|ALL|[PRIVILEGES]}
ON [usuario.]objeto
FROM {usuario|rol|PUBLIC} [,{usuario|rol|PUBLIC] ...;
REVOKE CREATETAB ON DATABASE FROM USER6
```

- **Del sistema o roles a usuarios**:

```
REVOKE {privilegio_stma | rol} [, {privilegio_stma | rol}]...|ALL|
[PRIVILEGES]}
ON [usuario.]objeto
FROM {usuario|rol|PUBLIC} [,{usuario|rol|PUBLIC] ...;
REVOKE UPDATE
ON EMPLEADOS
FROM PUBLIC
```

Juan va a quitar el permiso de seleccionar y de actualizar sobre la tabla Usuarios a Ana:

```
REVOKE SELECT, UPDATE ON Usuarios FROM Ana;
```

y va a quitarle el permiso de eliminar usuarios:

```
REVOKE DROP USER FROM Ana;
```

Autoevaluación

Usuarios y permisos.

Comando	Relación	Función
CREATE USER	3	1. Se utiliza para dar permisos a los usuarios o roles.
DROP USER	2	2. Se utiliza para eliminar usuarios.
GRANT	1	3. Se utiliza para crear usuarios.
REVOKE	4	4. Se utiliza para quitar permisos.

Anexo I.- Elementos del lenguaje SQL.

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.

- **COMANDOS:**

Comandos DDL. Lenguaje de Definición de Datos.

Comando:	Descripción:

CREATE	Se utiliza para crear nuevas tablas, campos e índices.
	`CREATE TABLE tabla1 (`
	`col1 INTEGER CONSTRAINT pk PRIMARY KEY,`
	`col2 CHAR(25) NOT NULL,`
	`col3 CHAR(10) CONSTRAINT uni1 UNIQUE,`
	`col4 INTEGER,`
	`col5 INT CONSTRAINT fk5 REFERENCES tab2 );`

DROP	Se utiliza para eliminar tablas e índices. `DROP TABLE tabla 1`
ALTER	Se utiliza para modificar tablas. `ALTER TABLE tabla1 ADD columna3 integer`

Comandos DML. Lenguaje de Manipulación de Datos.

Comando:	Descripción:

SELECT	Se utiliza para consultar filas que satisfagan un criterio determinado.
	`SELECT numemp, nombre`
	`FROM tabla_empleados`
	`WHERE ventas > cuota`

INSERT	Se utiliza para cargar datos en una única operación. `INSERT INTO tabla_empleados VALUES (200, 'Juan López', 30, NULL, 'dep ventas')`

UPDATE	Se utiliza para modificar valores de campos y filas específicos.
	`UPDATE tabla_empleados SET ventas_mes = 0`
	`WHERE departamento=`dep-ventas’`

DELETE	Se utiliza para eliminar filas de una tabla. `DELETE * FROM tabla_empleados`

Comandos DCL. Lenguaje de Control de Datos.

Comando:	Descripción:
GRANT	Permite dar permisos a uno o varios usuarios o roles para realizar tareas determinadas. `GRANT CONNECT ON DATABASE TO USER WINKEN, USER BLINKEN, USER NOD`
REVOKE	Permite eliminar permisos que previamente se han concedido con GRANT. `REVOKE CREATETAB ON DATABASE FROM USER6`

- **CLÁUSULAS:**

  Llamadas también condiciones o criterios, son palabras especiales que permiten modificar el funcionamiento de un comando.

Cláusulas:	Descripción:
FROM	Se utiliza para especificar la tabla de la que se van a seleccionar las filas. `SELECT * FROM tabla_empleados`
WHERE	Se utiliza para especificar las condiciones que deben reunir las filas que se van a seleccionar. `Select * FROM tabla_empleados WHERE alta=’true’`

GROUP BY	Se utiliza para separar las filas seleccionadas en grupos específicos.
	`SELECT provincia`
	`FROM tabla_empleados`
	`GROUP BY provincia;`

HAVING	Se utiliza para expresar la condición que debe satisfacer cada grupo. `SELECT * FROM tabla_empleados HAVING COUNT(ventas)>5`
ORDER BY	Se utiliza para ordenar las filas seleccionadas de acuerdo a un orden específico. `Select nombre, apellidos, ventas FROM tabla_empleados ORDER BY ventas`

- **OPERADORES:**

  Permiten crear expresiones complejas. Pueden ser aritméticos (+, -, *, /, ...) o lógicos (< , >, , < >, And, Or, …).

  El operador * representa a todos los campos de la Base de Datos

Operadores lógicos.

Operador:	Descripción:
AND	Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas. `SELECT nombre, apellidos FROM tabla_empleados WHERE sexo='hombre' AND edad>60`
OR	Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta. `SELECT nombre, apellidos FROM tabla_empleados WHERE edad<30 OR edad>60`
NOT	Devuelve el valor contrario de la expresión. `SELECT nombre, apellidos FROM tabla_empleados WHERE NOMBRE NOT LIKE 'JUAN*'`

Operadores de comparación.

Operadores:	Descripción:
<	Menor que.
>	Mayor que.
< >	Distinto de.
< =	Menor o igual.
> =	Mayor o igual.
=	Igual.
BETWEEN	Se utiliza para especificar un intervalo de valores.
LIKE	Se utiliza para comparar.
IN	Se utiliza para especificar filas de una base de datos.

- **FUNCIONES:**

  Para conseguir valores complejos. Por ejemplo, la función promedio para obtener la media de un salario. Existen muchas funciones, aquí tienes la descripción de algunas.

Funciones de agregado.

Función:	Descripción:
AVG	Calcula el promedio de los valores de un campo determinado. `SELECT nombre FROM tabla_empleados WHERE VENTAS > AVG(ventas)`
COUNT	Devuelve el número de filas de la selección. `SELECT COUNT (dni) FROM tabla_empleados WHERE PROVINCIA=´ÁVILA´`
SUM	Devuelve la suma de todos los valores de un campo determinado. `SELECT SUM (importe_ventas) FROM tabla_empleados WHERE DNI='11222333T'`

Función:	Descripción:
MAX	Devuelve el valor más alto de un campo determinado. `SELECT MAX(importe_ventas) FROM tabla_empleados`
MIN	Devuelve el valor mínimo de un campo determinado. `SELECT MAX(importe_ventas) FROM tabla_empleados`

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

Literales:	Descripción:
23/03/97	Literal fecha.
María	Literal caracteres.
5	Literal número.

Anexo II.- Instalación de Oracle Database Express Edition 11g.

A partir de ahora vamos a trabajar con este Sistema Gestor de Base de Datos. Para ello debes ir a la página oficial de Oracle.

1. En ella podemos elegir entre varios tipos de descargas según sea nuestro Sistema Operativo

2. La página de Oracle solicitará nuestro **registro** para realizar la descarga, pues tenemos que ser usuarios registrados para poder bajarlo.

3. Una vez bajado el archivo, deberemos de descomprimirlo en una carpeta y ejecutar el archivo **setup.exe**.

4. Al hacerlo, aparecerá una ventana donde podrás ver que está preparándose para la instalación y tras unos segundos aparecerá la página de bienvenida donde pulsaremos en siguiente.

5. A continuación, se nos muestra otra ventana donde tenemos que aceptar los términos del acuerdo para poder continuar con la instalación:

6. Tras aceptar y pulsar en **siguiente** aparecerá otra ventana donde podemos **elegir donde instalar el programa**, en principio es mejor dejar lo que aparece por defecto.

7. Pulsamos en **siguiente**. En la ventana a continuación nos solicitará **que introduzcamos una contraseña** para los usuarios **SYS** y**SYSTEM** (será la misma para los dos).

8. Pulsamos en **siguiente** y aparecerá otra ventana donde nos informa entre otras cosa, que la instalación se realizará para conectar mediante HTTP en el **puerto 8080**. Ahora el botón que tenemos que pulsar es **Instalar**.

El asistente se pondrá a instalar. Transcurrido un tiempo aparecerá otra ventana que **indicará que el proceso a terminado** y que pulsemos en **Terminar**. Tras estos pasos, Oracle se abrirá automáticamente para que puedas comenzar a trabajar.

También se habrá creado un **icono de acceso:**

También podrás ver que tienes un acceso en el **botón de inicio.**

Si accedemos a **Get Started,** debe aparecer una ventana donde podremos acceder a la base de datos que trae por defecto.

Aquí usaremos las claves que pusimos antes. Ya estamos listos para trabajar.

MATERIAL COMPLEMENTARIO

Ejemplos DCL

Vamos a ejecutar sentencias SQL desde la línea de comandos. Utilizar la cónsola es habitual,por ejemplo, si tenemos un servidor con una base de datos que atiende a clientes, no es necesario que éste sea gráfico, no gasta cpu para ello. Y tiene una ventaja fundamental: es independiente de la plataforma.

Para habilitar la cónsola de SQL de Oracle, ejecutaremos el comando sqlplus (en windows será sqlplus.exe y en linux es un ejecutable sqlplus o un .sh). Normalmente en entornos gráficos nos crea un acceso directo, o en el menú de programas un enlace llamado “Run SQL Command Line”. En Linux en modo cónsola teclearemos sqlplus /nolog

NO estamos trabajando en DOS, ni en cónsola de Linux, es el intérprete de SQL de Oracle, VÁLIDO para TODAS LAS PLATAFORMAS en que funciona Oracle.

Primero debemos conectarnos, con el usuario/esquema administrador y el comando: connect. Escribimos

```
connect system
```

y pulsamos intro. Nos solicitará la clave y si la introducimos correctamente nos dará el mensaje de “Connected” y pasaremos a dar órdenes a nuestra base de datos.

Opcionalmente también podemos escribir el comando así:

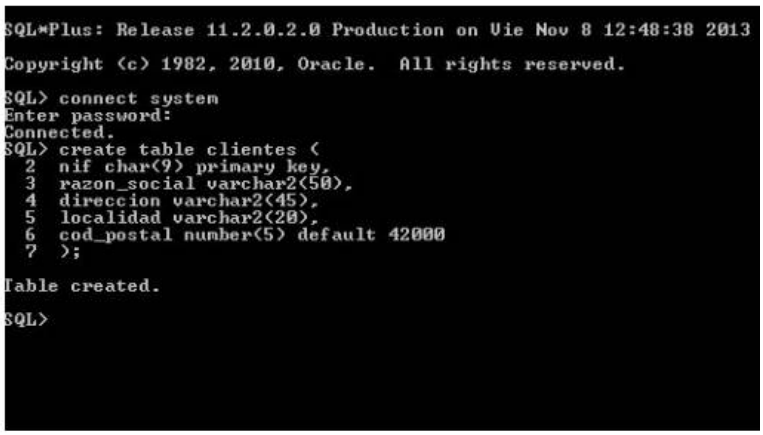
```
connect system/clave_que_tengamos_en _el_usuario_system
```

Tanto si los escribimos de una forma u otra, si tiene éxito podremos gestionar nuestra base de datos.

Por ejemplo vamos a crear una tabla de clientes con los siguientes campos:

Tabla clientes

- nif char(9) clave primaria
- razon_social alfanumérico de 50 caracteres
- direccion alfanumérico de 45 caracteres
- localidad alfanumérico de 20 caracteres
- cod_postal numérico de 5 posiciones enteras por defecto con valor 42000



Una orden en SQL puede tener varias líneas, Para terminar una orden y ejecutarla escribimos un punto y coma al final de la misma y pulsamos intro. Si no queremos ejecutar la orden (por no ser necesario, por equivocación, etc) simplemente pulsamos intro.

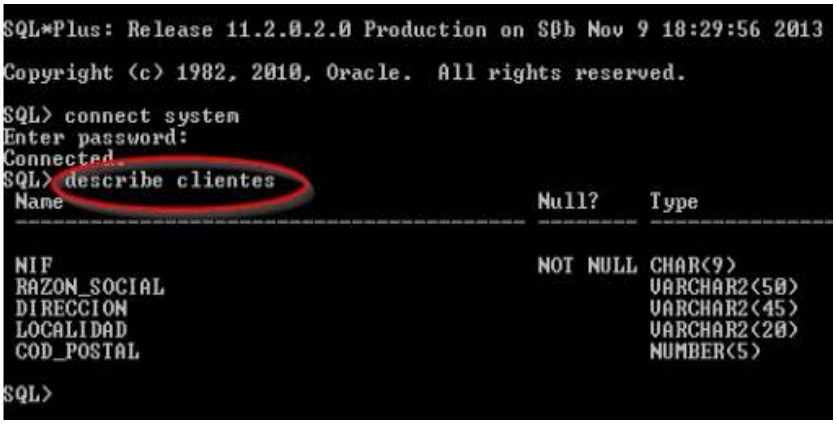
También podemos ejecutar una orden sin escribir el punto y coma y una vez que hayamos pulsado intro, escribimos RUN.

Cuando escribimos una orden que ocupa varias líneas aparecen automáticamente unos números que indican el número de línea que ocupa la orden, NO LOS TENEMOS QUE ESCRIBIR, son parte de un editor de textos de cónsola.

Para ver las características de la tabla que hemos creado usaremos el comando:

```
describe
nombre_de_tabla;
```

En la imagen vemos el resultado de ejecutar el comando



Ahora vamos a crear un usuario nuevo llamado javier con clave homs, lo haremos mediante el comando create user:

```
create user javier
identified by homs;
```

, y una vez creado nos queremos conectar con él.

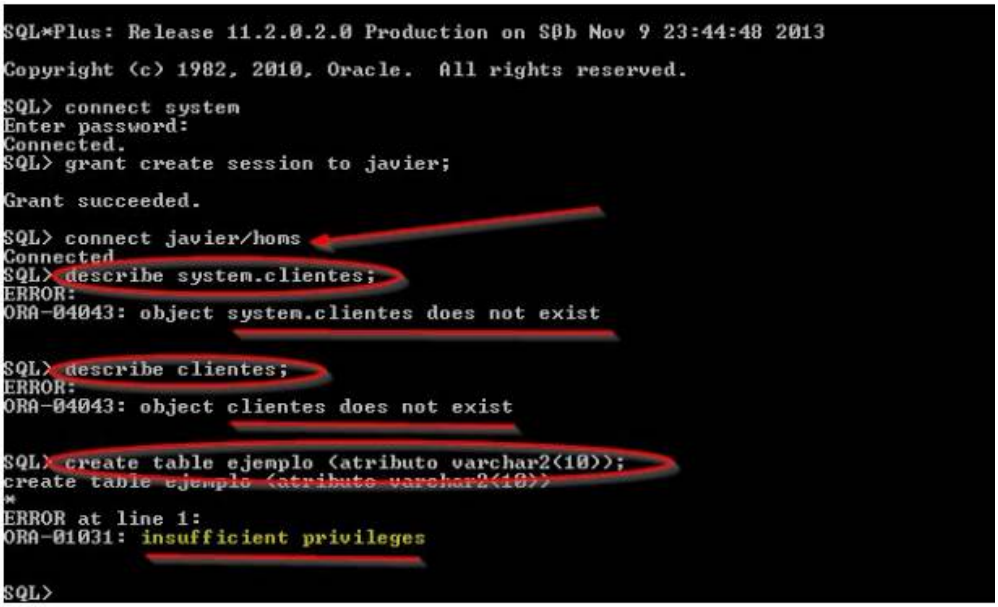
Pero no basta, crear un usuario no supone que pueda se pueda operar con él. Hay que asignarle privilegios y/o roles.

Como la conexión ha fallado, estamos fuera del sistema y es necesario volver a identificarnos con un usuario que tenga suficientes privilegios.

Por lo tanto le daremos al usuario javier, la posibilidad de conectarse, asignándole el privilegio de conexión, mediante:

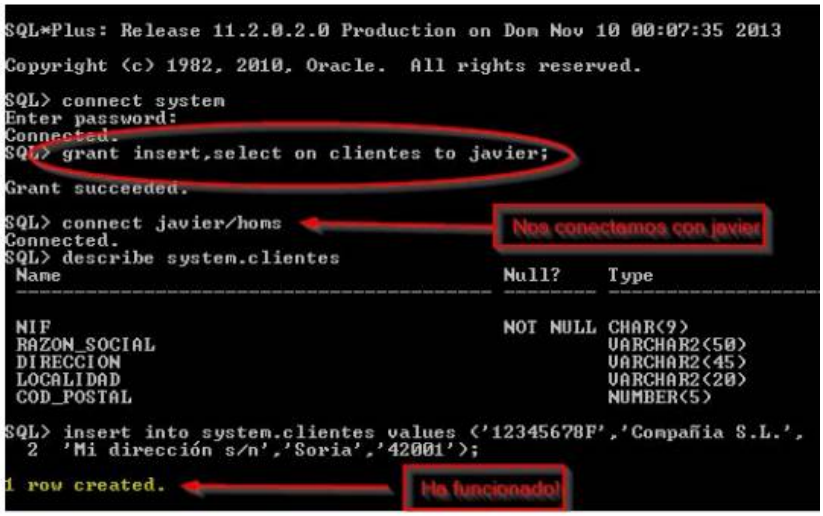
```
grant create session to javier;
```

El privilegio create session permite usar el comando connect, pero no permite nada más. En la imagen vemos que lo que ocurre después de conectarnos con javier al intentar realizar alguna operación:



nota: clientes no es una tabla de javier, por tanto para referenciarla escribimos delante de ella al usuario al que pertenece: system.clientes

Le concederemos algún privilegio a javier para que pueda trabajar: Primeramente nos conectamos con algún usuario que tenga suficientes privilegios y aplicaremos la sentencia grant sobre la tabla clientes, por ejemplo: le daremos privilegios de consulta e inserción de datos.



En el ejemplo vemos que el usuario javier puede ver las características de la tabla clientes e insertar un registro (más adelante se verá la sentencia insert).

Para retirar los privilegios el usuario que se los ha dado o un usuario administrador ejecutaría la orden:

```
revoke select, insert
on clientes from
javier;
```

En Oracle hay dos tipos de privilegios:

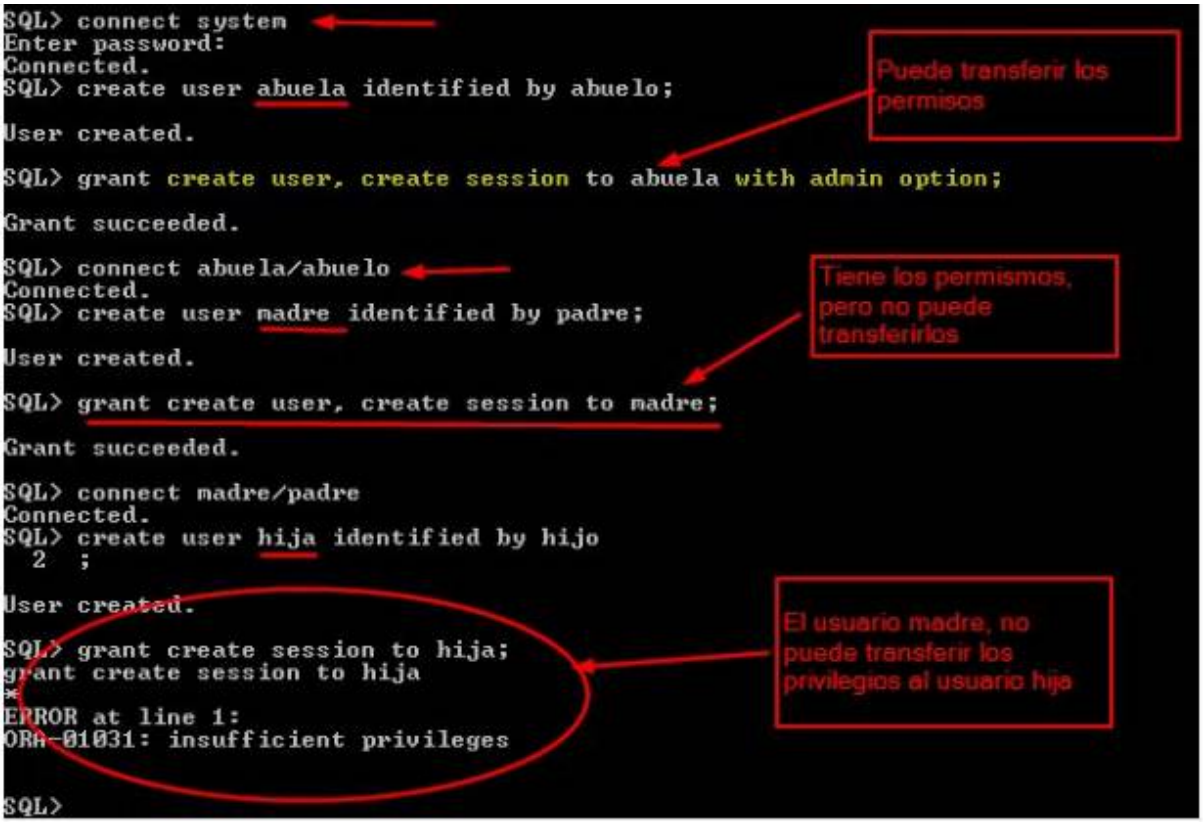
● De Sistema: los permisos de sistema, permiten ejecutar comandos del tipo DDL (Data definition Language), como CREATE, ALTER y DROP o del tipo DML (Data Manipulation Language). Oracle tiene más de 150 privilegios de sistema. Por ejemplo el privilegio create session es de este tipo. El formato más simple es:

```
GRANT privilegios TO usuario [WITH GRANT OPTION]
```

● De Objeto: Este tipo de permiso le permite al usuario realizar ciertas acciones en objetos de la BD, como una Tabla, Vista, un Procedure o Función, etc. Los privilegios otorgados a javier sobre clientes son un ejemplo de ellos. Su formato básico es:

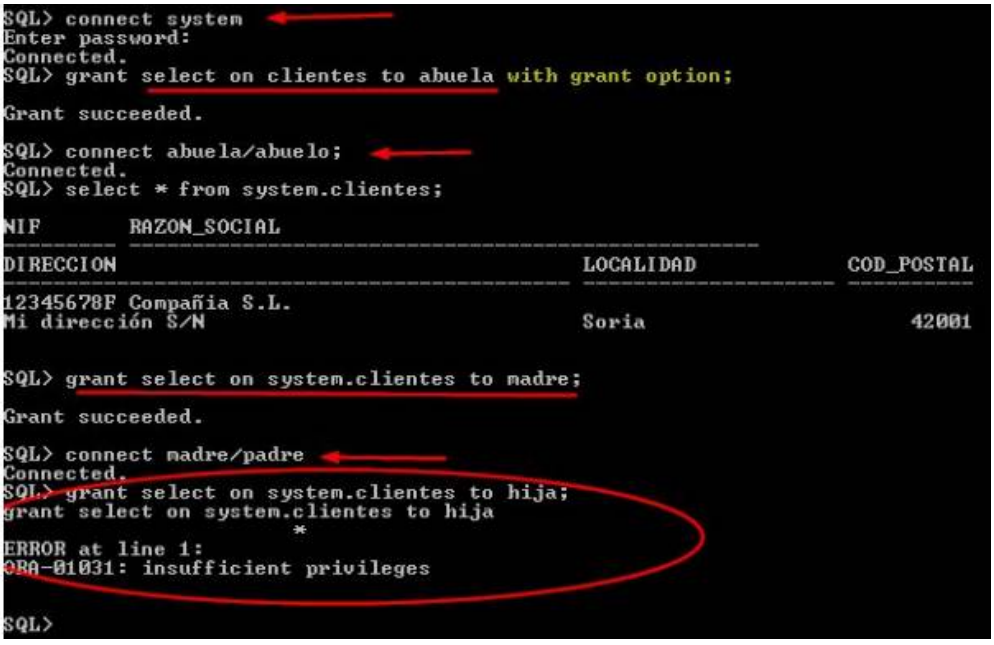
```
GRANT privilegios ON objeto TO usuario [WITH ADMIN OPTION]
```

Con la cláusula WITH ADMIN OPTION podemos otorgar los privilegios de SISTEMA a los usuarios que creemos:



En la figura el usuario administrador crea el usuario abuela, con los permisos de crear usuario y sesión con la opción “with admin option”. Éste se los transfiere al usuario madre, pero sin dicha opción, con lo cual no puede transmitirlos al usuario hija.

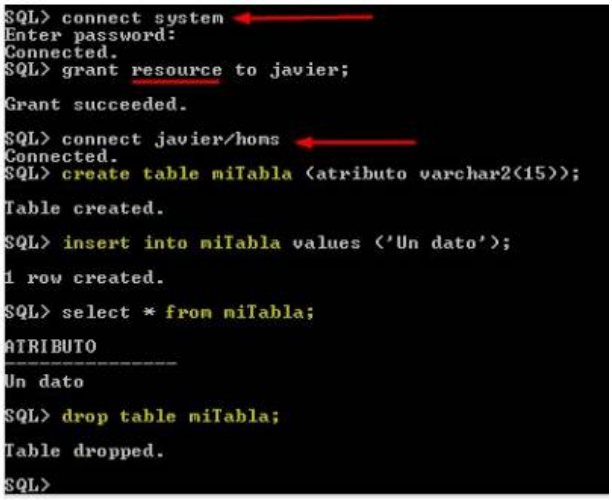
Si añadimos la cláusula WITH GRANT OPTION, significa que autorizamos a dar los mismos permisos que tenemos SOBRE LOS OBJETOS (tablas, vistas, índices...) a cualquier usuario que creemos. En la figura siguiente vemos como la clase abuela se le dan permisos de consulta sobre la tabla y el poder de transferirlo a los usuarios que produzca.



Con este permiso, el usuario abuela, puede ejecutar la sentencia de consulta sobre las tablas select (más adelante se estudiará). Y la otorga al usuario hija creado por ella en el ejemplo anterior. El usuario "madre" podrá utilizar la sentencia "select", pero al no tener la opción "with grant option" no puede trasladar ésta, al usuario hija.

Las cláusulas "with admin option" y "with grant option" han de manejarse con sumo cuidado, sobre todo en organizaciones grandes, pues podemos perder el control de quien es el dueño de los datos de la base de datos.

En cambio un rol es un **conjunto** de privilegios (del tipo que sea) que se asignan a un usuario o a otro rol. Es más fácil trabajar con conjunto de privilegios que uno a uno. Oracle posee un rol para otorgar a un usuario la capacidad de crear/modificar/borrar todos los objetos (tablas, índices, vistas, funciones, disparadores...) para manejar su "propia" base de datos. Este rol se denomina **resource.**



En la imagen vemos como al atribuir el rol de resource al usuario javier, éste puede crear, consultar y borrar una tabla. Por supuesto resource tiene más permisos que los vistos en el ejemplo, pero por ahora es suficiente saber que permite tratar su propia base de datos.

Por último es muy aconsejable terminar la sesión con EXIT o QUIT (son sinónimos), para que los cambios se guarden en la base de datos.

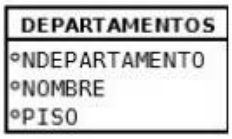
Ejemplos crear tablas

Una empresa tiene los siguientes datos sobre sus empleados y departamentos:

Los datos obligatorios son:

En EMPLEADOS el DNI, nombre, apellidos y ndepartamento. Esté último atributo indica el departamento al que pertenece el trabajador

En DEPARTAMENTOS el ndepartamento y el nombre.



Crear las tablas:

Comencemos por DEPARTAMENTOS:

Esta tabla contiene 3 atributos, primeramente analicemos cual puede ser CLAVE.

NDEPARTAMENTO es el número de departamento, lógicamente será único para cada departamento y siempre tendrá un valor, por lo tanto es una buena CLAVE CANDIDATA

NOMBRE es descriptivo sobre los valores que tiene y siempre ha de contener alguno, además es distinto para cada departamento, por lo tanto también es una buena CLAVE CANDIDATA

PISO en cambio, si no nos dicen nada, puede repetirse para varios departamentos (varios pueden estar en el mismo piso) y además según el enunciado no tiene que ser un valor conocido obligatoriamente. Por lo tanto será un atributo.

A la hora de definir quien es clave aplicaremos dos “normas”

- El que se guarde en el menor número de bytes
- El que se pueda repetir en otras tablas

La clave primaria es un valor que se va a poder repetir, si nos fijamos NDEPARTAMENTO indica el número de departamento en la tabla DEPARTAMENTOS y el número de departamento al que pertenece un trabajador en la tabla EMPLEADOS.

Viendo estas dos reglas nos quedaría que la CLAVE PRIMARIA sería NDEPARTAMENTO y NOMBRE sería una CLAVE ALTERNATIVA y por lo tanto podríamos asignar la restricción de UNIQUE.

Veamos ahora los tipos de datos:

NDEPARTAMENTO es un valor numérico, el tamaño dependerá del número de departamentos que tengamos o de los que podamos llegar a tener.

NOMBRE de departamento será alfanumérico y su longitud dependerá del negocio que tratemos

PISO es numérico y es el mismo caso que NDEPARTAMENTO.
Por tanto la tabla quedaría:

```
create table departamentos (
ndepartamento numeric(3) primary key,
nombre varchar2(25) unique not null,
piso numeric(2)
);
```

Veamos la tabla de EMPLEADOS

Primeramente cual puede ser la clave:

DNI es único para cada empleado y obligatorio, por tanto es CLAVE CANDIDATA

NOMBRE no es único para cada empleado aunque es obligatorio

APELLIDOS no es único para cada empleado aunque es obligatorio

FECHA_NAC no es único para cada empleado y puede ser nula

DIRECCIÓN puede no contener valores según el enunciado

NDEPARTAMENTO contiene valores, pero más de un empleado puede estar en dicho departamento.

Una posible combinación de nombre y apellidos podría ser clave?. Es difícil que dos empleados se llamen igual pero puede ocurrir por lo tanto no es una buena combinación.

La combinación de nombre, apellidos y ndepartamento tiene una probabilidad mucho menor de coincidir en 2 trabajadores, pero es una posibilidad.

Ahora los tipos:

DNI es alfanumérico y de 9 caracteres

NOMBRE alfanumérico, alrededor de 20

APELLIDOS alfanumérico alrededor de 45

FECHA_NAC de tipo fecha

DIRECCIÓN alfanumérico alrededor de 50

NDEPARTAMENTO tiene que coincidir con su homólogo de la tabla DEPARTAMENTOS, por tanto no solo tiene que coincidir sino que además debe ser CLAVE AJENA (para que sea admitido un registro de un trabajador en la tabla EMPLEADOS es necesario que NDEPARTAMENTO sea nulo o que el valor del número de departamento exista en la tabla DEPARTAMETOS, como en este caso el enunciado nos indica que el valor no puede ser nulo esto significa que un empleado ha de PERTENECER SIEMPRE a un departamento de nuestra empresa)

Por tanto quedaría:

```
create table empleados (
dni char(9) primary key,
nombre varchar2(20) not null,
apellidos varchar2(45) not null,
fecha_nac date,
direccion varchar2(50),
ndepartamento numeric(3) not null,
foreign key (ndepartamento) references departamentos (ndepartamento)
);
```

Ejemplos modificar tablas

Tabla compradores

Name	Null?	Type
CIF_COMPRADOR	NOT NULL	VARCHAR(11)
NOMBRE_SOCIAL		VARCHAR(30)
DOMICILIO_SOCIAL		VARCHAR(30)
LOCALIDAD		VARCHAR(30)
C_POSTAL		VARCHAR(5)
TELEFONO		CHAR(9)

Tabla artículos	Null?	Type
referencia_articulo	NOT NULL	VARCHAR(6)
descripcion_articulo		VARCHAR(30)
precio_unidad		NUMBER(6,0)
iva		NUMBER(2,0)
existencias_actuales		NUMBER(5,0)
stock_minimo		NUMBER(4,0)

Crear las siguientes tablas

Una vez creadas modificar las tablas con la sentencia ALTER TABLE

Convertir el campo: cif_comprador en clave primaria:

```
alter table compradores add
constraint clave_primaria_pk
primary key (cif_comprador);
```

Desde este momento la tabla solo permitirá añadir un cif_comprador diferente para cada registro de la tabla, si la tabla hubiese tenido datos ya introducidos con anterioridad, y alguno de ellos tuviera el cif_comprador repetido en distintos registros, habría dado error al crear la clave primaria.

También se podría crear la clave primaria sin ponerle un nombre a la restricción:

```
alter table compradores add primary key (cif_comprador);
```

La diferencia estriba en que en principio no se podría borrar la restricción ya que no tiene nombre, pero la realidad es que Oracle le asigna uno automáticamente.

Añadir la columna fax a la tabla de compradores con la misma definición que el teléfono.

```
alter table compradores add (fax varchar2(9));
```

Si hubiera que añadir más atributos a la vez, se separarían entre sí con una coma;

Por ejemplo:

```
alter table compradores add (fax varchar2(9), fecha date)
```

Añadir en la tabla artículos la restricción de comprobar que la columna precio_unidad contenga un valor NOT NULL y mayor que 0.

```
alter table articulos add constraint control_precio_ck check
(precio_unidad>0);
alter table articulos modify precio_unidad number(6,0) not null;
```

Para añadir la restricción check se ha añadido un constraint, pero no se puede realizar constraint para añadir la restricción de not null, es necesario modificar la definición del atributo. Cuando se realiza la definición de un atributo hay que tener mucho cuidado con la definición, podemos tener pérdida de caracteres o de definición de la cantidad numérica si se cambia la precisión del atributo, y Oracle no dejaría realizarlo.

Modificar la tabla compradores para poner 42000 en la columna c_postal como valor por defecto

```
alter table compradores modify c_postal number(5) default 42000;
```

Este es otro ejemplo típico, en el que default no puede declararse con un constraint, es necesario modificar el campo.

Modificar la tabla compradores para decrementar de 30 a 20 caracteres los tamaños de las columnas domicilio_social y localidad.

```
alter table compradores modify localidad varchar2(20);
alter table compradores modify domicilio_social varchar2(20);
```

Si alguno de los valores que hubiera antes de cambiar el tamaño fuera mayor que 20, Oracle por defecto no deja que haya dicha modificación, es decir, defiende el dato.

Modificar la tabla artículos para que tenga dos decimales el precio unidad.

```
alter table articulos modify precio_unidad number(6,2);
```

Pero para poder hacerlo, Oracle precisa que la columna precio_unidad debe estar vacía, es decir, contener todo null, sino la operación es rechazada

Cambiar el tipo de datos de cif_comprador de varchar2 a char

```
alter table compradores modify cif_comprador char(11)
```

Normalmente el paso de char a varchar2 y viceversa no tiene ningún problema, excepto que los datos que contienen no quepan.

Vamos a suponer que únicamente vendemos a cada comprador un artículo distinto, por tanto vamos a poner la clave de compradores en artículos. De esa manera sabemos inmediatamente el artículo que se ha comprado..

```
alter table articulos add (cif_comprador char(11));
```

Pero no es suficiente, debemos añadir integridad referencial, es decir, el valor que introduzcamos de cif_comprador en la tabla artículos debe existir en la tabla compradores o ser nulo.

```
alter table articulos add constraint clave_ajena_fk foreign key
(cif_comprador) references compradores(cif_comprador);
```

Si tenemos datos introducidos, todos los artículos que haya en la tabla tendrán el cíf_comprador nulo, pero a la hora de modificar un dato o introducir uno nuevo es necesario que exista el valor en la tabla compradores o sea nulo.

Eliminar la columna stock_minimo de la tabla artículos

```
alter table articulos drop (stock_minimo);
```

Automáticamente la columna desaparece, si hubiera datos, estos se han perdido.

Crear un índice en la tabla compradores en el atributo nombre social

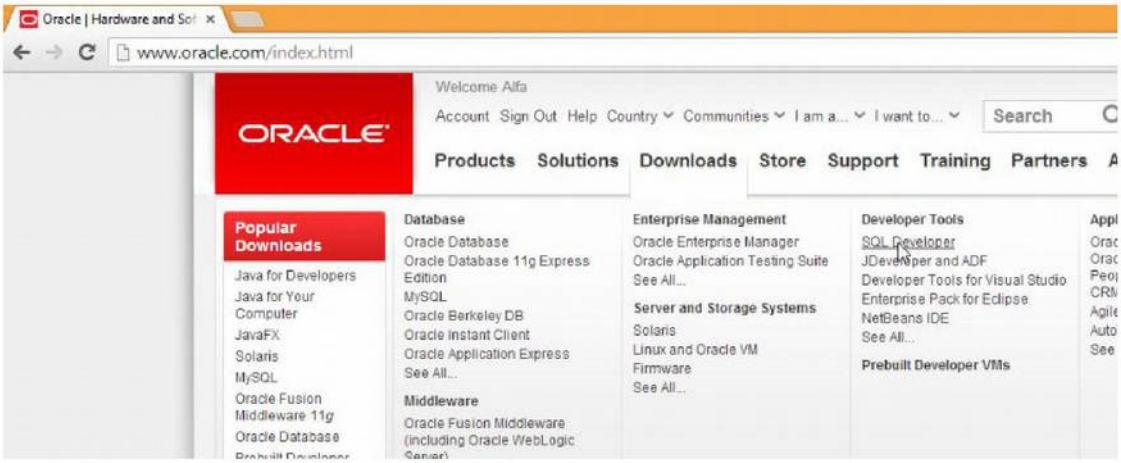
```
create index nombre on compradores (nombre_social);
```

Crear un índice es una decisión compleja, es recomendable si se van hacer búsquedas sobre ese campo, o listados ordenados. Pero si una tabla sufre muchas modificaciones y/o borrados también se actualiza el fichero índice, y es una operación costosa

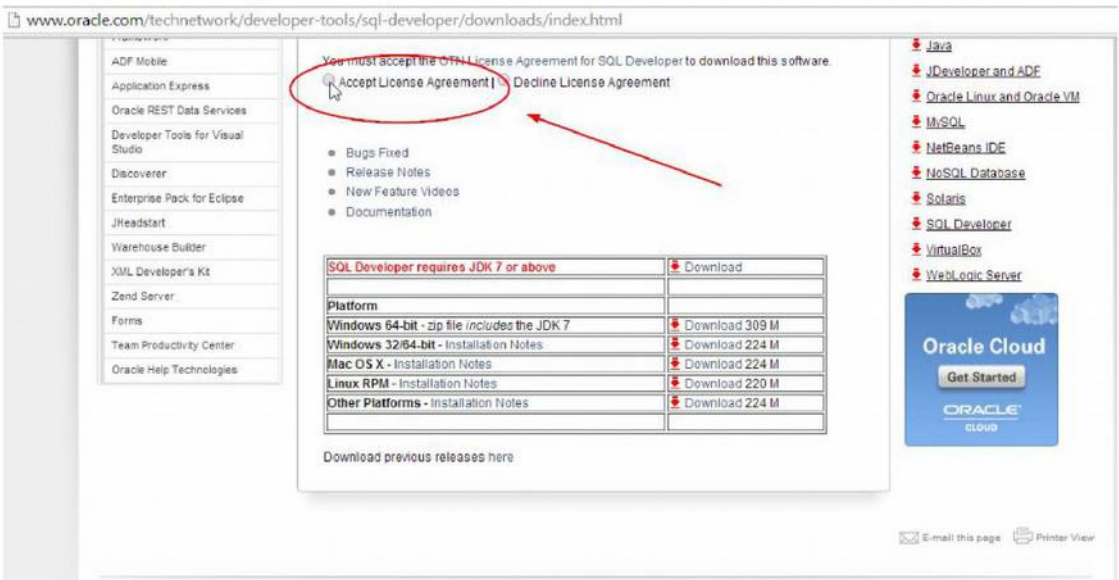
SQL Developer

Vamos a utilizar un entorno gráfico profesional para acceder a Oracle. Existe la herramienta SQL Developer realizada en JAVA por Oracle. Permite conectar mediante TCP/IP a un servidor Oracle. El entorno APEX está pensado para generar aplicaciones de mantenimiento web para las bases de datos, pero no para realizar muchas de las operaciones que habitualmente se usan para trabajar en Bases de datos.

Primeramente descargamos SQLDeveloper desde http://www.oracle.com. Pulsamos sobre **Downloads** y en la opción de **Developer tools** presionar sobre **SQL Developer**:



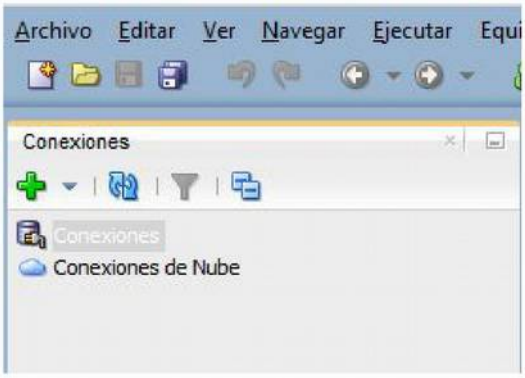
Aceptamos la licencia:



La versión a descargar dependerá del sistema operativo empleado. SQLDeveloper es un programa en JAVA, por lo tanto necesitamos instalar el JDK de JAVA, (o nos podemos bajar la opción para windows 64 que la tienen incluida). En este caso lo normal es que ya tengamos instalado el JDK (lo usamos en otros módulos del ciclo).

En linux tenemos un paquete rpm de instalación, pero en los demás sistemas descomprimimos el zip y ejecutamos el fichero “sqlDeveloper” (no es necesaria instalación).

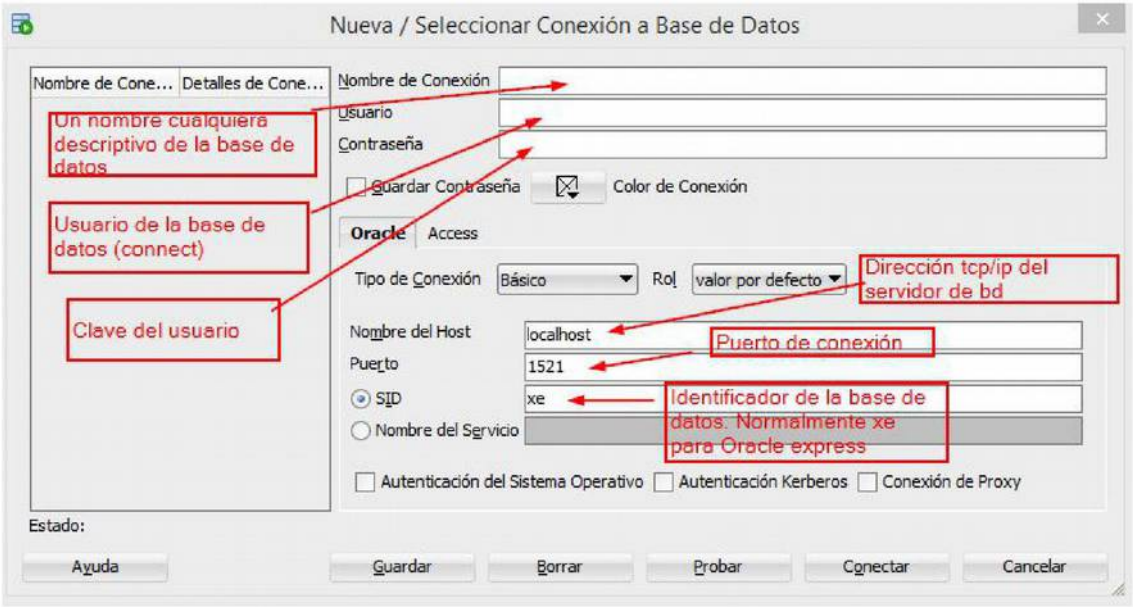
Lo habitual es que solicite la primera vez donde está instalado el JDK



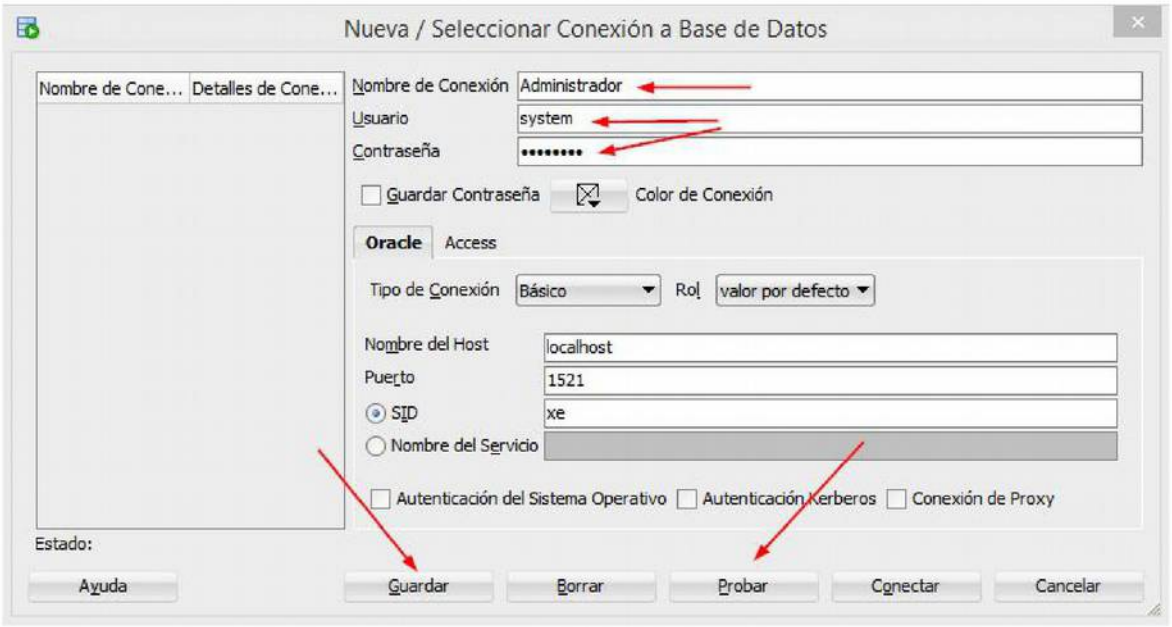
En la parte izquierda de la pantalla tenemos las conexiones con los distintos servidores de base de datos que podamos tener:

Para realizar una nueva conexión pulsamos sobre el símbolo + .

Se abre una ventana donde introducimos los datos del usuario de la base de datos que deseemos manejar. (Aquí en Oracle se suele confundir el concepto de base de datos con el de usuario de base de datos).

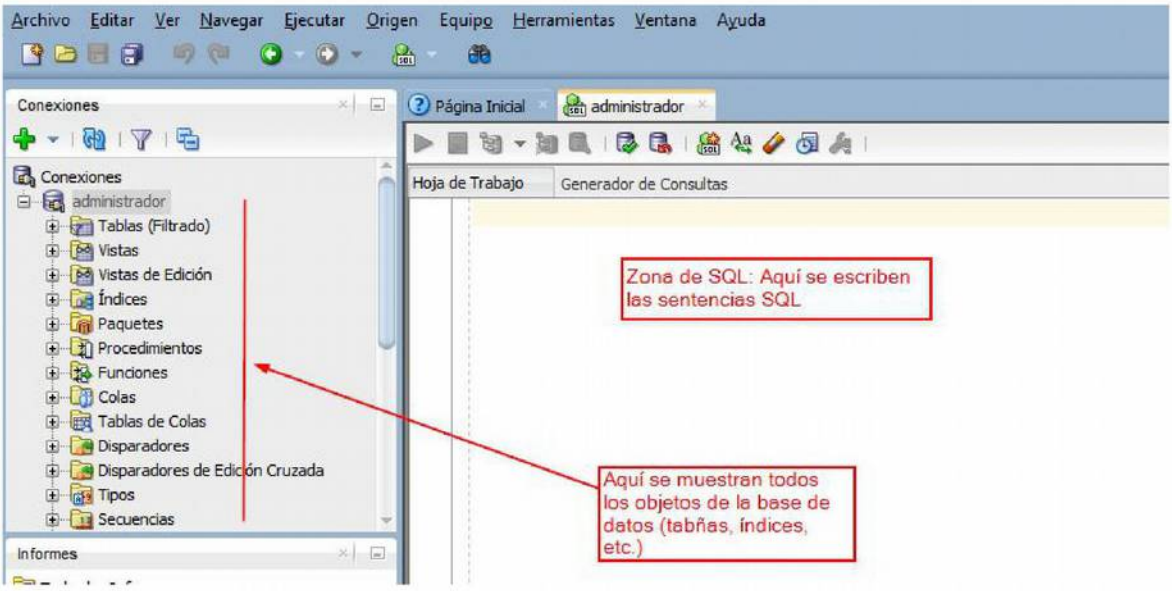


Una de las primeras conexiones a realizar será con el usuario system, para poder crear y manejar las demás conexiones:



Una vez puesto los datos pulsamos sobre el botón “Probar” y si no nos devuelve ningún mensaje de que no ha podido conectar, guardamos la conexión.

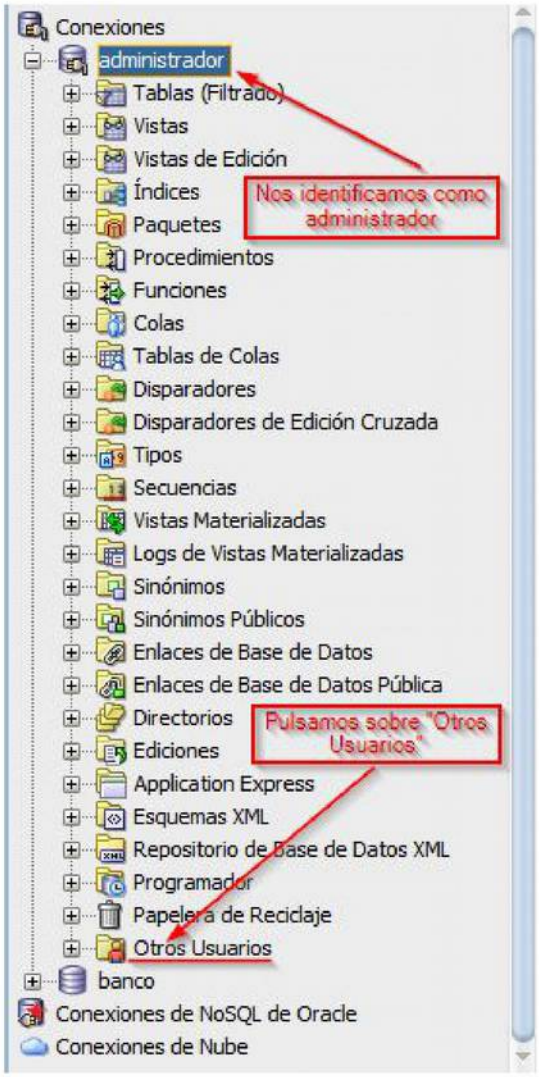
Para usar una conexión solo es necesario realizar doble click sobre ella, si hemos guardado la clave entraremos directamente (en un sistema de producción de datos real no hay que almacenar la contraseña NUNCA), en caso contrario nos la solicitará.



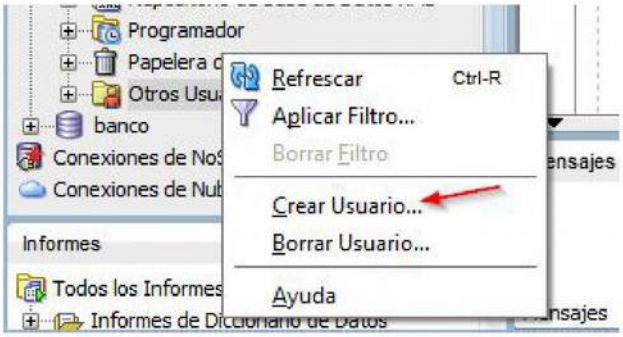
Lo primero que hay que hacer es crear una conexión como administradores de nuestra bases de datos.

Ejemplo 1:Veamos ahora como crear un usuario de bases de datos y asignarle permisos de uso:

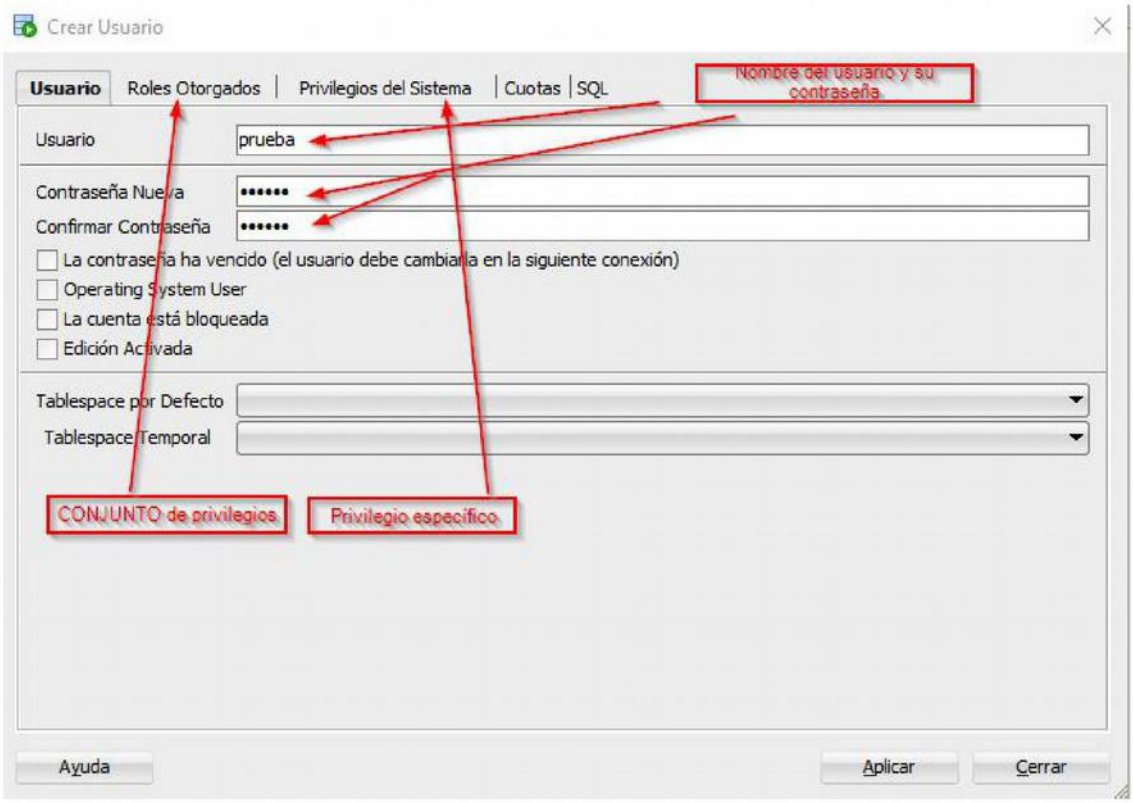
Primeramente nos identificamos como Administrador del sistema (system) y pulsamos con el BOTÓN DERECHO (si no sois zurdos y tenéis configurado el izquierdo) sobre otros



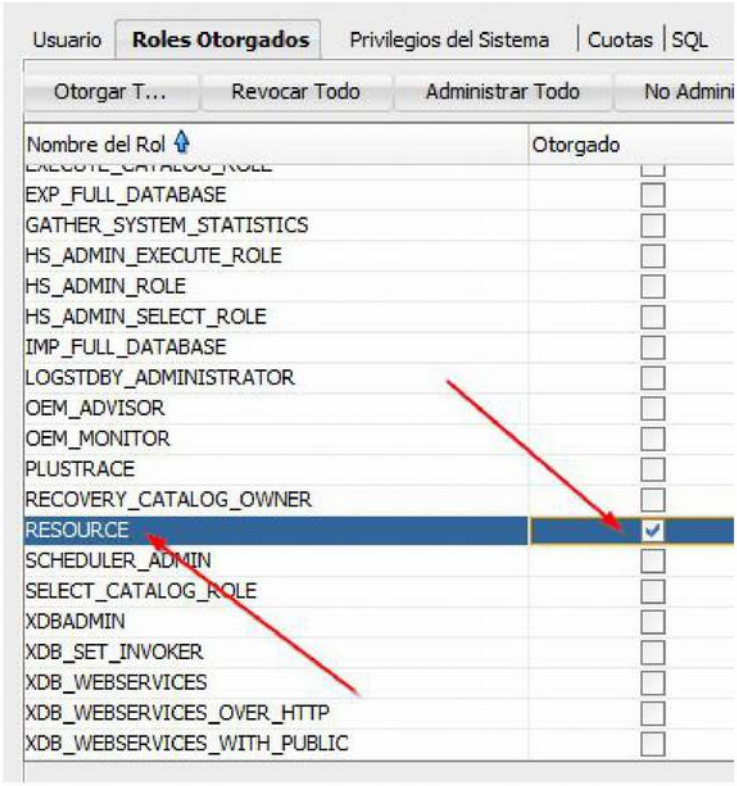
usuarios
Aparece un submenú donde pulsaremos sobre “Crear usuario”



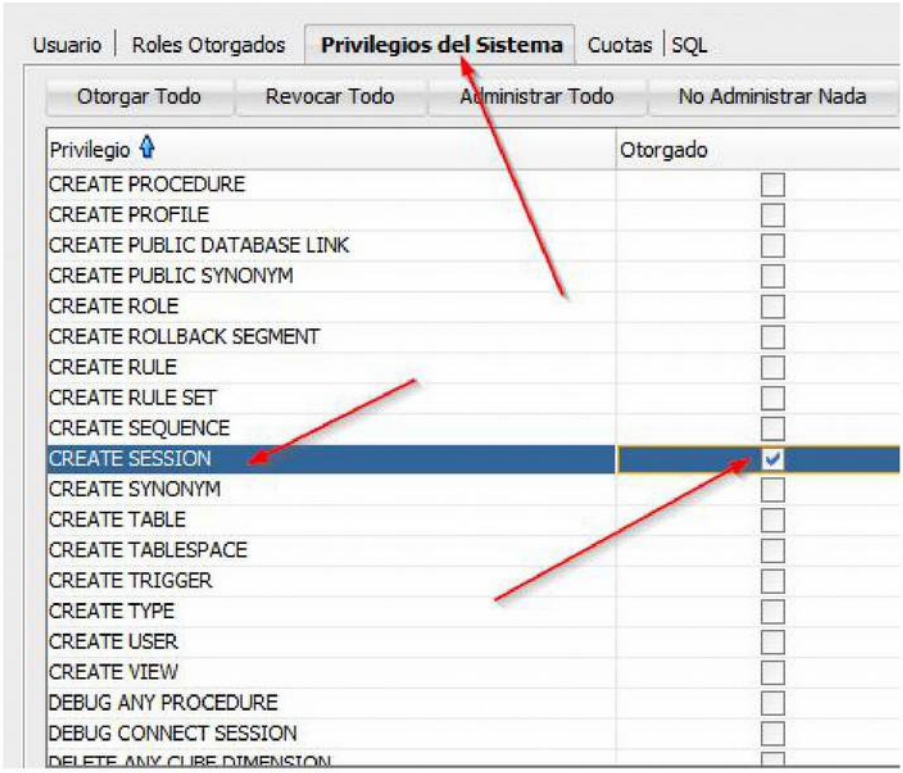
Y nos sale una ventana donde incluiremos el nombre del usuario, la contraseña y nos fijamos en las pestañas de roles, y privilegios del sistema.



Pulsamos sobre la ventana de roles (conjunto de privilegios que se asignan) y buscamos el rol de “resource” (está ordenado alfabéticamente) y lo marcamos. Este rol permite gestionar (altas,bajas, modificaciones, etc.) las tablas, índices, procedimientos, etc. al usuario. Es decir, sin este permiso no podría hacer nada.

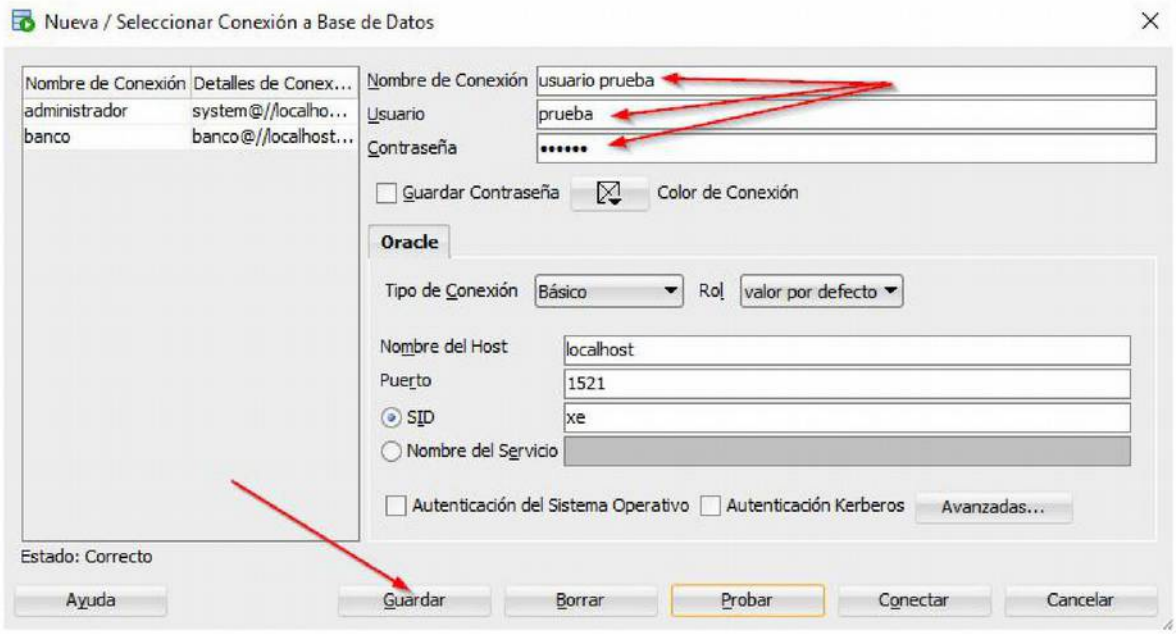


Seguidamente pulsamos sobre la pestaña de "Privilegios del sistema" y buscamos el privilegio (permiso único a diferencia de un rol) de "create session" que permite al usuario "prueba" poder conectarse al sistema, es decir, crear una conexión:



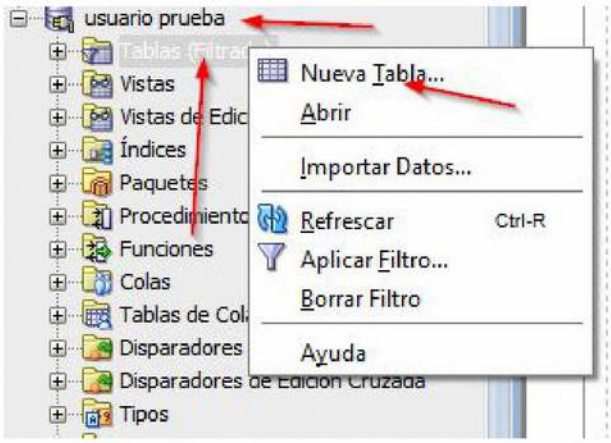
Ahora ya podemos crear una CONEXIÓN, tal y como se ha explicado para el usuario

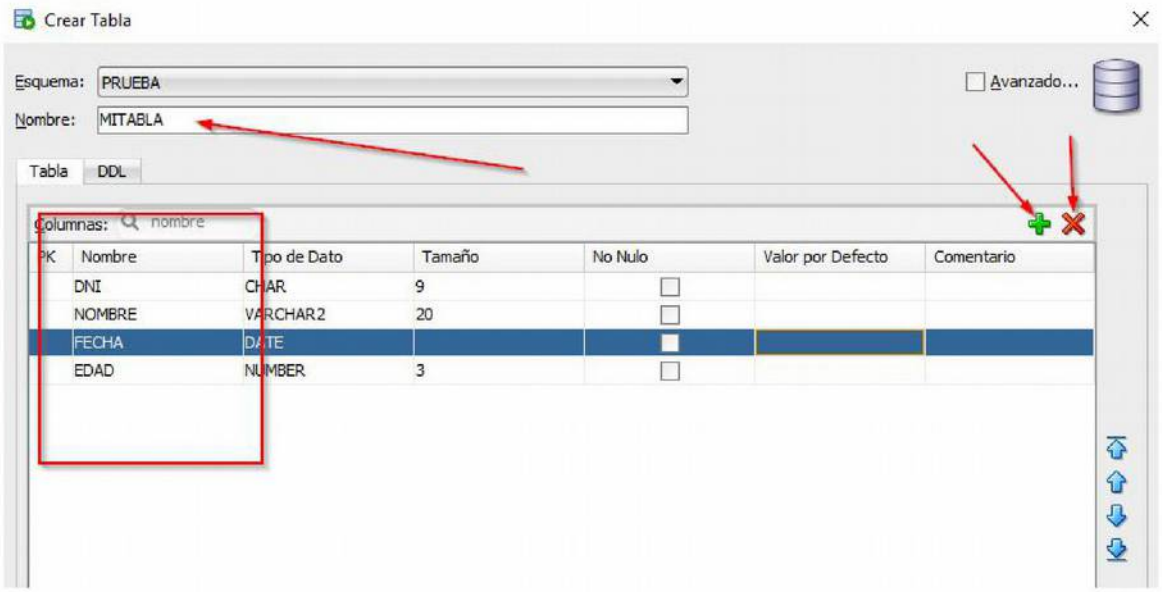
“system”



Una vez conectados ya podemos trabajar con el usuario “prueba”

Si hacemos “doble click” sobre la conexión nos conectaremos y podremos ver los distintos objetos que posee. Si pulsamos el botón derecho del ratón sobre tablas, veremos que podemos crear una tabla:





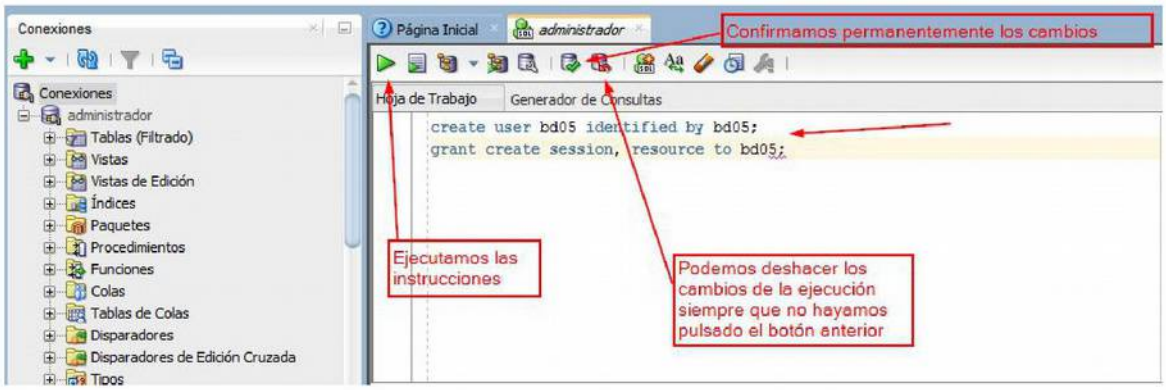
Incluimos el nombre de la tabla y con los signos + y x podemos añadir o borrar los campos.

Al aceptar se creará la tabla.

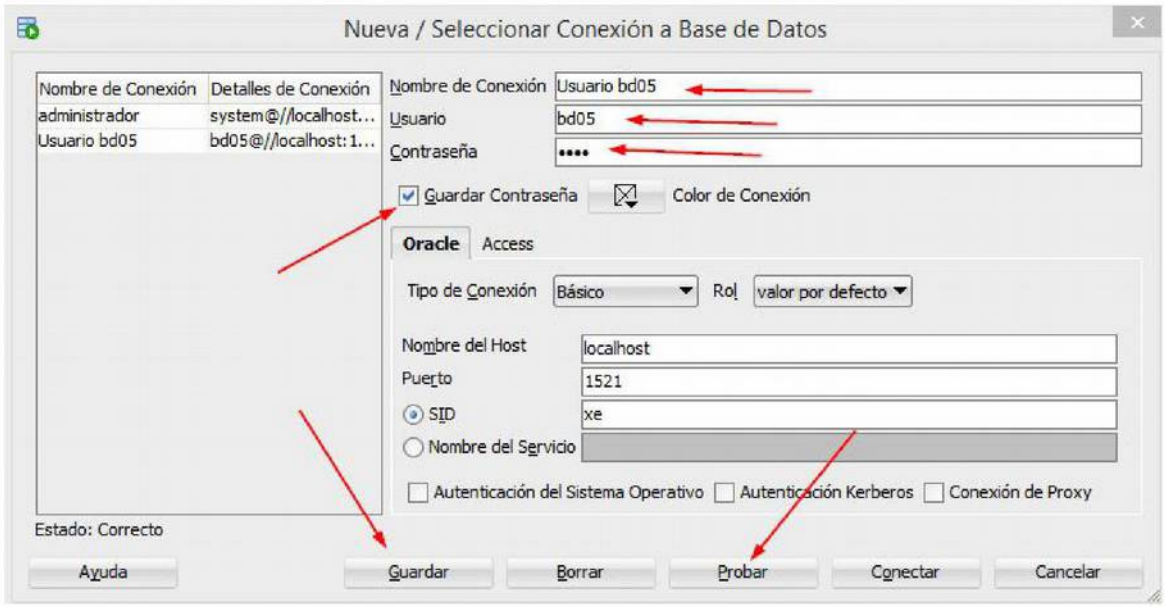
Ejemplo 2: Crear el usuario bd05 con clave bd05

Primeramente nos identificamos como administrador, pulsamos el botón:

Se abrirá una pestaña donde podremos escribir nuestras sentencias donde creamos el usuario, y le damos permisos (grant create session, resource to bd05)



Una vez creada la base de datos creamos la conexión:



Desde ese momento nos conectamos al usuario y podremos utilizarlo.