

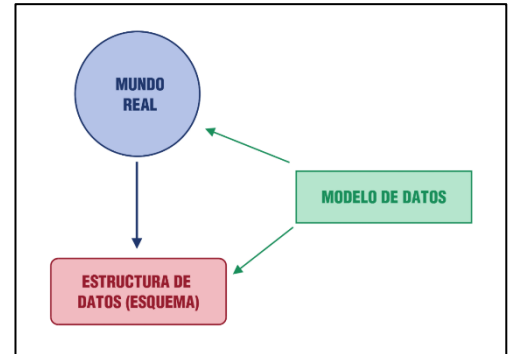
TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

1.	MODELO DE DATOS.....	1
2.	TERMINOLOGÍA DEL MODELO RELACIONAL.....	3
2.1.	RELACIÓN O TABLA. TUPLAS. DOMINIOS.....	3
2.2.	GRADO. CARDINALIDAD.....	5
2.3.	SINÓNIMOS.....	6
3.	RELACIONES.CARACTERÍSTICAS DE UNA RELACIÓN (TABLA).....	6
3.1.	TIPOS DE RELACIONES (TABLAS).....	7
4.	TIPOS DE DATOS.....	7
5.	CLAVES.....	8
5.1.	CLAVE CANDIDATA. CLAVE PRIMARIA. CLAVE ALTERNATIVA.....	9
5.2.	CLAVE EXTERNA, AJENA O SECUNDARIA.....	10
6.	ÍNDICES. CARACTERÍSTICAS.....	11
7.	EL VALOR NULL. OPERACIONES CON ESTE VALOR.....	12
8.	VISTAS.....	13
9.	USUARIOS. ROLES. PRIVILEGIOS.....	14
10.	SQL.....	15
10.1.	ELEMENTOS DEL LENGUAJE. NORMAS DE ESCRITURA.....	16
11.	LENGUAJE DE DESCRIPCIÓN DE DATOS (DDL).....	17
11.1.	CREACIÓN Y BORRADO DE BASES DE DATOS. OBJETOS DE LA BASE DE DATOS.....	18
11.2.	CREACIÓN DE TABLAS.....	19
11.3.	RESTRICCIONES.....	21
11.3.1	Restricción NOT NULL.....	22
11.3.2	Restricción UNIQUE.....	22
11.3.3	Restricción PRIMARY KEY.....	23
11.3.4	Restricción REFERENCES. FOREIGN KEY.....	23
11.3.5	Restricción DEFAULT Y VALIDACIÓN.....	25
11.4.	ELIMINACIÓN DE TABLAS.....	27
11.5.	MODIFICACIÓN DE TABLAS.....	28
11.6.	CREACIÓN Y ELIMINACIÓN DE ÍNDICES.....	30
11.7.	CREACIÓN Y ELIMINACIÓN DE VISTAS.....	32
12.	LENGUAJE DE CONTROL DE DATOS (DCL).....	34
12.1.	PERMISOS.....	36
13.	ENLACES DE REFUERZO Y AMPLIACIÓN.....	40
	ANEXO 1: ELEMENTOS DEL LENGUAJE SQL.....	41

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).



Otras definiciones de modelo de datos serían:

Según *Dittrich (1994)* un **modelo de datos** se puede considerar como un **conjunto de herramientas conceptuales** para describir la representación de la información en términos de datos. Los modelos de datos comprenden aspectos relacionados con: **estructuras y tipos de datos, operaciones y restricciones**.

También según *De Miguel et. al (1999)* un modelo de datos se puede considerar como un conjunto de conceptos, reglas y convecciones que permiten describir y manipular los datos de la parcela de un cierto mundo real que deseamos almacenar en la base 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.

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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.



Para saber más

Principales tipos de Bases de Datos

Los principales tipos de Bases de Datos son:

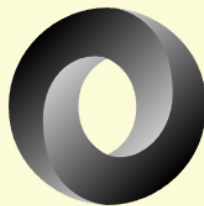
- ✓ **Relacionales:** la información que almacena la Base de Datos está relacionada entre sí. Los datos relacionados (registros o filas) son almacenados en tablas que constan de varios campos (columnas).
- ✓ **No relacionales:** los datos no tienen por qué estar relacionados entre sí y por lo tanto no tienen que almacenarse en estructuras fijas como las tablas del modelo de base de datos relacional.

Las Bases de Datos NoSQL pertenecen al modelo no relacional. Las principales características y ventajas de este tipo son:

- ✓ SQL no es el lenguaje de consulta/modificación de datos principal, aunque sí lo soportan, de ahí el nombre No Sólo SQL.
- ✓ Los datos no tienen que almacenarse en tablas.
- ✓ Generalmente, su arquitectura es distribuida almacenándose la información en más de una máquina del sistema. Por lo tanto, los sistemas que las soportan tienen una mayor escalabilidad horizontal (a mayor número de nodos mayor rendimiento) y también mayor tolerancia ante fallos en los distintos nodos.
- ✓ Son más eficientes en el procesamiento de los datos que las Bases de Datos relacionales
- ✓ Son más eficientes en el procesamiento de los datos que las Bases de Datos relacionales, por eso son la elección para aplicaciones que hacen un uso intensivo de estos ("streaming", etc.).
- ✓ Utilizan lo que se conoce como consistencia eventual que consiste en que los cambios realizados en los datos serán replicados a todos los nodos del sistema, lo cual aumenta el rendimiento de estos sistemas en contraposición a las propiedades ACID de las Bases de Datos relacionales.

¿Por qué JSON en bases de datos relacionales?

Las bases de datos Not only SQL, se basan en un esquema flexible de datos, en los cuales no necesitas declarar o crear primero dicho esquema para comenzar a almacenar información como tampoco es estrictamente necesario el proceso de normalización. Dentro de la industria desde hace años, llegaron a irrumpir las bases de datos de tipo documento tales como MongoDB, las cuales mostraron que al no estar amarradas al esquema tradicional de SQL, podían ofrecer una velocidad de escritura y lectura aún muy superior a lo manejado por las bases de datos relacionales; sin embargo esa realidad se ha vuelto a modificar gracias a los últimos esfuerzos de MySQL gracias a su implementación nativa para guardar, modificar y eliminar datos en formato JSON.



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 Codd¹ en los laboratorios de IBM² 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 $A \times B$.³

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.

Aquí puedes encontrar un vídeo donde se explica de manera gráfica las relaciones matemáticas entre conjuntos. Seguro que te resulta interesante refrescar estos conceptos:

<https://www.youtube.com/watch?v=dvEaYtAja5g&t=1s>

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.

Para saber más: Aquí tienes unos enlaces sobre teoría de conjuntos y lógica de predicados: [Teoría de conjuntos](#). [Lógica de predicados](#).

¹ Científico informático inglés (23 de agosto de 1923 18 de abril de 2003), conocido por sus aportes a la teoría de bases de datos relacionales.

² International Business Machines o IBM es una empresa multinacional estadounidense que fabrica y comercializa herramientas, programas y servicios relacionados con la informática. IBM tiene su sede en Armonk (Nueva York, Estados Unidos) y está constituida como tal desde el 15 de junio de 1911, pero lleva operando desde 1888.

³ En teoría de conjuntos, el producto cartesiano es un producto directo de conjuntos. En particular, el producto cartesiano de dos conjuntos X e Y, denotado por $X \times Y$, es el conjunto de todos los pares ordenados en los que el primer componente pertenece a X y el segundo a Y.

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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.

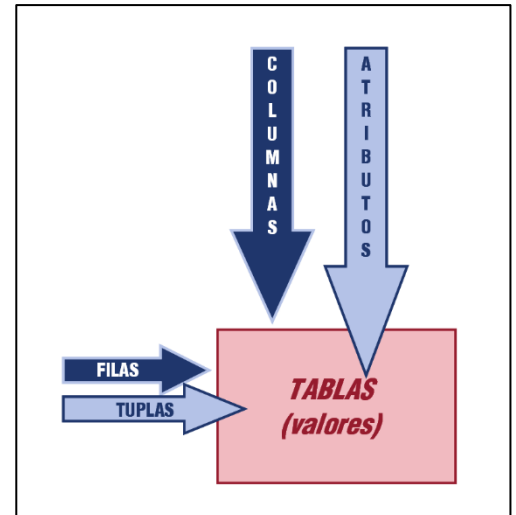
A menudo un dominio se define a través de la declaración de un tipo para el atributo (por ejemplo, diciendo que es un número entero entre 1 y 16), pero también se pueden definir dominios más complejos y precisos. Por ejemplo, para el atributo Sexo de mis usuarios, podemos definir un dominio en el que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



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 $A=\{\text{Carlos, María}\}$, $B=\{\text{Matemáticas, Lengua, Inglés}\}$, $C=\{\text{Aprobado, Suspenso}\}$.

Producto Cartesiano $A \times B \times C$.		
$A=\{\text{Carlos, María}\}$	$B=\{\text{Matemáticas, Lengua}\}$	$C=\{\text{Aprobado, Suspenso}\}$
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

Las posibles relaciones que obtenemos al realizar el producto cartesiano $A \times B \times C$ es el siguiente:

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

Subconjunto del Producto Cartesiano $A \times B \times C$ con cardinalidad 5.		
$A=\{\text{Carlos, María}\}$	$B=\{\text{Matemáticas, Lengua}\}$	$C=\{\text{Aprobado, Suspenso}\}$
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 términos de registros:** FICHEROS - REGISTROS - CAMPOS - NÚMERO CAMPOS - NÚMERO REGISTROS.

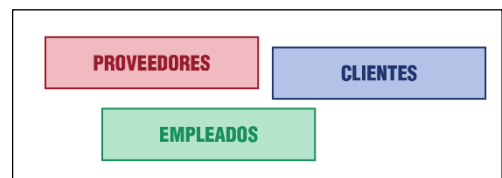
Nomenclatura relacional		Nomenclatura tabla		Nomenclatura ficheros
relación	=	tabla	=	fichero
tupla	=	fila	=	registros
atributo	=	columna	=	campos
grado	=	nº columnas	=	nº campos
cardinalidad	=	nº filas	=	nº registros

3. RELACIONES. CARACTERÍSTICAS DE UNA RELACIÓN (TABLA).

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

Debes saber que:

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

Carlos	Matemáticas	Aprobado
Carlos	Lengua	Suspense
Carlos	Inglés	NOTABLE
María	Matemáticas	Suspense
María	Lengua	Suspense

a	20	=	a	20
b	30		c	70
c	70		b	30

a	20	=	20	a
b	30		70	c
c	70		30	b

3.1. TIPOS DE RELACIONES (TABLAS).

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

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

4. TIPOS DE DATOS.

¿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.

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)
- Debe tener asociado un **Tipo de dato**.

Existen distintas formas de nombrar los tipos de datos dependiendo del lenguaje que utilicemos(C, Java, PHP, 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 que vamos a realizar operaciones matemáticas.
- **Fecha/hora:** almacena fechas y horas.

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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

Para saber más: Si quieres saber un poco más sobre los tipos de datos puedes ver este enlace de Wikipedia: [Tipo de datos.](#)

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**.

Para saber más

En este enlace tienes más información sobre las superclaves: [Superclaves.](#)

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

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

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

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 caso contrario, 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.

Para saber más: Interesante artículo sobre las claves ajenas y su importancia: [De las claves ajenas, foráneas, externas...](#)

Si necesitas refrescar o simplemente aprender el concepto de clave primaria, en la wikipedia puedes consultarlo: [Claves primarias.](#)

6. ÍNDICES. CARACTERÍSTICAS.

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 se 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.

Para saber más: Si quieres conocer más sobre los índices y MySQL puedes leer este artículo: [Índices](#)

7. EL VALOR NULL. OPERACIONES CON ESTE VALOR.

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

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

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

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

Pero ten en cuenta una cosa, no es lo mismo valor NULO que ESPACIO EN BLANCO.
Tampoco será lo mismo valor NULO que el valor CERO.

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

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

Cuando necesitemos comparar dos campos, si ambos son nulos no podremos obtener ni verdadero ni falso. Necesitaremos definir la lógica con este valor. Veamos los operadores lógicos más comunes y sus resultados utilizando el valor nulo:

- 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.

Para saber más: El uso del valor nulo es un tema que da mucho que hablar, aquí puedes leer sobre ello: [Valores nulos](#)

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 dos razones por las que queramos crear vistas:

- **Seguridad**, nos puede interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.
- **Comodidad**, 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.

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 Server⁴ sí y MySQL Server⁵ también permite vistas actualizables, pero bajo muchas restricciones.

⁴ Sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional.

⁵ MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009— desarrolla MySQL como software libre en un esquema de licenciamiento dual.

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 tablespace⁶ 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 administrador⁷ 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.
- **Sobre objeto:** necesitará el permiso sobre el objeto en cuestión.

¿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.

⁶ Unidad lógica de almacenamiento dentro de una base de datos Oracle

⁷ Es la persona o equipo de personas profesionales responsables del control y manejo del sistema de base de datos, generalmente tiene experiencia en Sistemas Gestores de Base de Datos, diseño de bases de datos, sistemas operativos, comunicación de datos y programación.

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 declarativo⁸ 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 (ASP⁹ o PHP¹⁰) en combinación con cualquier tipo de base de datos (Access, SQL Server, MySQL, Oracle, etc.).

El hecho de que sea estándar¹¹ 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 Generación¹².

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.
- 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.

Para saber más: En este enlace encontrarás de una manera breve, pero interesante, la historia del SQL. [Historia de SQL.](#)

⁸ Tipo de lenguaje de programación basado más en las matemáticas y en la lógica que los lenguajes imperativos, más cercanos estos al razonamiento humano. Los lenguajes declarativos no dicen como hacer una cosa, sino, más bien, qué cosa hacer. A diferencia de los imperativos, no suele haber declaración de variables ni tipos.

⁹ Tecnología de Microsoft para páginas web generadas dinámicamente, ha sido comercializada como un anexo a Internet Information Services (IIS).

¹⁰ Lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas.

¹¹ Modelo a seguir al hacer algo. Son documentos que dan los detalles técnicos y las reglas necesarias para que un producto o tecnología se use correctamente

¹² Lenguajes de programación avanzados en los que el programador no incorpora el procedimiento a seguir, ya que el propio lenguaje es capaz de indicar al ordenador cómo debe ejecutar el programa. Suelen incluir interfaces gráficos y capacidades de gestión avanzadas, pero consumen muchos más recursos del ordenador que la generación de lenguajes previa.

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, 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 separarse con saltos de línea o espacios para facilitar su lectura y comprensión.
- Los comentarios comienzan por /* y terminan con */ (excepto en algunos SGBD).
- Se pueden tabular líneas para facilitar la lectura si fuera necesario.

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 y MySQL en su ordenador y así ir probando todo sobre la marcha para no cometer errores. El SQL estándar y el SQL de Oracle y MySQL son bastante parecidos, pero con algunas diferencias.

Debes conocer: En el siguiente documento encontrarás aquellos comandos, cláusulas, operadores y funciones más generales con las que vamos a trabajar a lo largo del curso: [Elementos del lenguaje](#).

Para saber más: Un Sistema Gestor de Base de Datos muy utilizado en diferentes entornos de desarrollo es MySQL. Sería interesante que lo conocieras y supieras instalarlo, si aún no lo tienes instalado: [Link de descarga de MySQL Manual MySQL](#).

Otra página recomendable donde puedes aprender MySQL desde cero es la siguiente: [MySQL con Clase](#).

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.

Por ejemplo, con Workbench de MySQL podemos:

- Realizar el diseño de la base de datos, mediante su herramienta Data Modeling e implantar mediante ingeniería directa la base de datos en el Servidor (si lo tenemos en marcha) o bien generar el **script SQL** (con las sentencias SQL para la creación de la base de datos y sus tablas) y cargarlo posteriormente en el servidor.
- Trabajar en modo gráfico con esta herramienta, a modo de formularios, y crear la base de datos, sus tablas, etc.

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, por ejemplo, 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.

Para saber más: Si quieres saber un poco más sobre el Lenguaje de Definición de Datos, puedes visitar la Wikipedia, aquí tienes el enlace: [Lenguaje de Definición de Datos](#)

11.1. CREACIÓN Y BORRADO DE BASES DE DATOS. OBJETOS DE LA BASE DE DATOS.

Básicamente, la creación de la base de datos consiste en crear las tablas que la componen. Aunque antes de esto, dependiendo del SGBD, 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;
```

En el caso de **MySQL** la sintaxis a utilizar sería la siguiente:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] NombredemiBaseDatos
```

- Obligatoriamente se debe poner bien **SCHEMA** o bien **DATABASE**,
- Opcionalmente se puede incluir **IF NOT EXISTS**. Si lo incluimos, al crear la base de datos de nombre **NombredemiBaseDatos**, si no existe una base dedatos con ese nombre, la crea, si ya existe, no intentará crearla ni dará error.

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;
```

Por otro lado, la instrucción más utilizada para **borrar** bases de datos es **Drop Database**. En MySQL el comando se acompaña con el nombre de la base de datos a eliminar. En el caso de querer eliminar la base de datos RyMjuegos pondríamos:

```
DROP 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.

Para saber más: Si quieres aprender a crear bases de datos con MySQL, aquí puedes aprender: [MySQL: creación de bases de datos.](#)

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.

Código	Nombre	Cod. Juego	
1	PARTIDA01	3	
2	PARTIDA02	2	
3	PARTIDA03	4	
4	PARTIDA04	3	
5	PARTIDA05	1	

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

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

Si nos centramos en **MySQL**, la sintaxis para crear una tabla sería la siguiente:

```
CREATE TABLE [IF NOT EXISTS] [esquema].nombretabla(
    Columna1 TipoDato [restricciones de columna]
    Columna2 TipoDato [restricciones de columna]
    .....
    [restricciones de tabla] )
[ (ENGINE | TYPE) = Tipo_Tabla];
```

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

donde:

- **nombretabla:** nombre de la tabla (identificador válido según las reglas soportadas por el gestor de bases de datos).
- **IF NOT EXISTS:** la tabla se creará si no existe ya una con ese nombre.
- **ColumnaN:** nombre de la columna (identificador válido según las reglas soportadas por el gestor de bases de datos).
- **TipoDato:** Tipo de dato para la columna (por ejemplo: **INT**, **VARCHAR**, **CHAR**, **DATE**, etc.)
- **ENGINE:** indica el tipo de almacenamiento para la tabla mediante **Tipo_Tabla**. En la misma BD puede haber tablas con diferente tipo de almacenamiento, como por ejemplo son las tablas MyISAM y las tablas InnoDB. (Los tipos de almacenamiento de tablas permitidos en nuestra SGBD los podemos ver con la sentencia: **SHOW ENGINES;**)

¿Qué son las restricciones?

Una **restricción** es una condición que una o varias columnas deben cumplir obligatoriamente. (En el siguiente apartado las veremos en detalle)

- **Restricciones de columna:** Afectan solo a esa columna (clave primaria, no nulo, etc.)
- **Restricciones de tabla:** se indican después de especificar todas las columnas, se les puede asignar un nombre y pueden afectar a varias columnas.

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 el SGBD.

- Para Oracle, en el siguiente enlace tienes una relación de estos tipos y su descripción. [Tipos de datos en Oracle.](#)
- Para MySQL, en el siguiente enlace tienes una relación de estos tipos y su descripción. [Tipos de datos en MySQL.](#)

En el caso de MySQL, se permiten varios tipos o motores de almacenamiento. Los dos tipos de almacenamiento más populares son: MyISAM e InnoDB. En el siguiente enlace puedes consultar en qué consiste cada uno de ellos. [Tipos MySAM e InnoDB](#)

Para saber más: Si quieres ver cómo se puede hacer un tipo de datos AUTOINCREMENT en Oracle puedes verlo en este enlace. [Secuencias en Oracle.](#)

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 creamos llevará un nombre, si no se lo ponemos nosotros lo hará Oracle, MySQL 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, para indicar las **restricciones a nivel de columna**.

Veamos un ejemplo en **Oracle**:

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


El mismo ejemplo en **MySQL**, sería:

```
CREATE TABLE USUARIOS2 (
  Login VARCHAR(15) PRIMARY KEY,
  Password VARCHAR(8) NOT NULL,
  Fecha_Ingreso TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

Otra restricción a nivel de columna es **[AUTO_INCREMENT]**.

- Con **[AUTO_INCREMENT]** MySQL permite asignar un identificador único a cada fila, generando secuencias de números de forma automática, comenzando usualmente desde el 1.

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



Recomendación

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

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

Para saber más: En el siguiente enlace puedes ver ejemplos del uso de la restricción u atributo **AUTO_INCREMENT** en MySQL. [Uso de AUTO INCREMENT en MySQL](#)

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 NOT NULL);
```

En **Oracle** también se podría nombrar la restricción. Podría ser:

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

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

11.3.2 Restricción UNIQUE.

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

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

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

En **Oracle** podría ser:

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

Y tanto en **Oracle** como en **MySQL** podría ser:

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 VARCHAR(25),
  Correo VARCHAR(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.

Reflexiona: Entonces, la restricción puesta de manera conjunta al Login y Correo, como restricción de Tabla en el ejemplo anterior

CONSTRAINT Usuario_UK UNIQUE (Login, Correo)

¿Es lo mismo que poner esa restricción de forma individual a cada columna?

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 si la clave fuese el campo Login:

- Si la clave la forma un único campo: (**Oracle y MySQL**)
- O bien poniendo un nombre a la restricción: (**Oracle**)

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

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

- Si la clave está formada por más de un campo, por ejemplo Nombre, Apellidos y Fecha de Nacimiento, entonces, obligatoriamente hay que ponerlo como restricción de tabla, después de haber declarado las columnas correspondientes: (**Oracle y MySQL**)

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

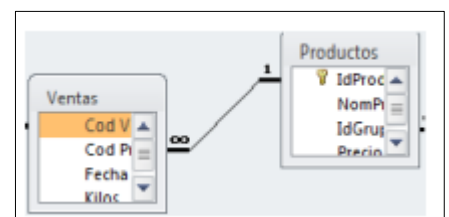
11.3.4 Restricción REFERENCES. FOREIGN KEY.

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 creamos 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. En **Oracle** se puede poner la restricción junto al campo que va a ser la clave ajena:

```
CREATE TABLE USUARIOS (
  Login VARCHAR (25) PRIMARY KEY,
  Cod_Partida INTEGER(8)
  CONSTRAINT Cod_Part_FK
  REFERENCES PARTIDAS(Cod_Partida));
```



TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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

```
CREATE TABLE USUARIOS (  
    Login VARCHAR (25) PRIMARY KEY,  
    Cod_Partida INTEGER(8)  
    CONSTRAINT Cod_Part_FK  
    REFERENCES PARTIDAS);
```

En **MySQL** al igual que con las restricciones anteriores la clave ajena se pone al final. En este caso tendremos que colocar el texto **FOREIGN KEY** para especificar a qué campo se está refiriendo. (También se podría hacer así en Oracle)

```
CREATE TABLE USUARIOS (  
    Login VARCHAR (25) PRIMARY KEY,  
    Cod_Partida INTEGER(8),  
    F_Partida DATE,  
    CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida)  
    REFERENCES PARTIDAS(Cod_Partida));
```

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 (  
    Login VARCHAR (25) PRIMARY KEY,  
    Cod_Partida INTEGER(8),  
    F_Partida DATE,  
    CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida, F_Partida)  
    REFERENCES PARTIDAS(Cod_Partida, F_Partida));
```

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: tanto Oracle como MySQL buscarán 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.
- **ON DELETE RESTRICT**: (igual que **NO ACTION** en MySQL) no te permitirá borrar el registro principal, si hay registros asociados con ese valor en su clave ajena.

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

Otras cláusulas, para mantener la Integridad Referencial, que van tras **REFERENCE** y permiten cambiar de forma automática el valor de las claves ajenas en función de la modificación hecha en el valor de la clave principal son:

- **ON UPDATE CASCADE:** te permitirá modificar el valor de la clave ajena de todos los registros cuya clave ajena sea igual a la clave del registro modificado.
- **ON UPDATE SET NULL:** colocará el valor **NULL** en todas las claves ajenas relacionadas con la modificada.
- **ON UPDATE RESTRICT:** (igual que **NO ACTION** en MySQL) no te permitirá modificar la clave primaria el registro principal, si hay registros asociados con ese valor en su clave ajena.

Por defecto, en **MySQL**, si no se indica nada, las opciones son:

ON DELETE RESTRICT

ON UPDATE RESTRICT

Por ejemplo, si queremos que al borrar una partida de la tabla PARTIDAS se borren todos los usuarios de la tabla USUARIOS que hayan jugado esa partida. Pondríamos:

```
CREATE TABLE USUARIOS (  
  Login VARCHAR (25) PRIMARY KEY,  
  Cod_Partida INTEGER(8),  
  F_Partida DATE,  
  CONSTRAINT Partida_Cod_F_FK FOREIGN KEY (Cod_Partida)  
  REFERENCES PARTIDAS(Cod_Partida) ON DELETE CASCADE);
```

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 (  
  Login VARCHAR (25) PRIMARY KEY,  
  Pais VARCHAR (20) DEFAULT 'España' );
```

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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, en caso de **Oracle**:

```
CREATE TABLE USUARIOS (  
    Login VARCHAR (25) PRIMARY KEY,  
    Fecha_ingreso DATE DEFAULT SYSDATE);
```

En **MySQL** no se pueden incluir expresiones o funciones SQL, excepto para el tipo de dato **TIMESTAMP**, que permite que se le asocie por defecto el valor **CURRENT_TIMESTAMP**, que es el momento actual en el que estemos (fecha y hora).

```
CREATE TABLE USUARIOS (  
    Login VARCHAR (25) PRIMARY KEY,  
    Fecha_ingreso TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

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 (Login VARCHAR (25) PRIMARY KEY),  
    Credito FLOAT(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.

Además, en **MySQL** también existe el tipo de datos **ENUM** que se puede utilizar para crear enumeraciones, es decir, campos que admiten sólo valores fijos. Por ejemplo:

```
CREATE TABLE USUARIOS (  
    Login VARCHAR (25) PRIMARY KEY,  
    Pais ENUM ('España', 'Francia', 'Bélgica'));
```

Debes conocer: Si queremos obtener una descripción de una tabla, sinonimo, paquete o función, podemos utilizar el comando **DESCRIBE**.

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 en **MySQL** utilizaremos el comando **DROP TABLE**.

```
DROP TABLE Nombre_tabla [Nombre_tabla] ...
```

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.

En **Oracle**, cambia un poco, pero es muy similar:

```
DROP TABLE nombre_tabla [ CASCADE CONSTRAINT]
```

No se permite el borrado de varias tablas en la misma sentencia y 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 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.

Oracle y MySQL disponen de la orden **TRUNCATE TABLE** que te permitirá eliminar los datos (filas) de una tabla sin eliminar su estructura.

Y recuerda que solo podrás borrar aquellas tablas sobre las que tengas permiso de borrado.

11.5. MODIFICACIÓN DE TABLAS

Es posible que después de crear una tabla nos demos cuenta que se nos ha olvidado añadir algún campo o restricción, quizás alguna de las restricciones que añadimos ya no es necesaria o tal vez queramos cambiar el nombre de alguno de los campos. ¿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 TABLE NombreViejo TO NombreNuevo;
```

- Si queremos añadir columnas a una tabla: las 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 (en Oracle) y FLOAT(en MySQL), 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:

- En Oracle:

```
ALTER TABLE NombreTabla CHANGE COLUMN Nombre.Antiguo NombreNuevo definiciónColumna;
```

- En MySQL:

```
ALTER TABLE NombreTabla CHANGE COLUMN Nombre.Antiguo NombreNuevo definiciónColumna;
```

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

Tenemos la siguiente tabla creada:

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

Por ejemplo, nos gustaría incluir una nueva columna llamada User que será tipo texto y clave primaria:

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

Nos damos cuenta que ese campo se llamaba Login y no User, vamos a cambiarlo (**en Oracle**):

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

Y en **MySQL**

```
ALTER TABLE USUARIOS CHANGE COLUMN User Login VARCHAR(10);
```

En Oracle, utilizando el comando **ALTER TABLE**, podemos modificar las restricciones o bien eliminarlas:

- **Si queremos borrar restricciones:** ALTER TABLE 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];

En MySQL:

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

- **Si queremos añadir restricciones, por ejemplo, la restricción UNIQUE:** ALTER TABLE NombreTabla ADD [CONSTRAINT [nombrecons]] UNIQUE [INDEX|KEY] [nombre_indice];
- **Si queremos eliminar una clave primaria:** ALTER TABLE NombreTabla DROP PRIMARY KEY;
- **Si queremos añadir una restricción de clave ajena:** ALTER TABLE NombreTabla ADD [CONSTRAINT [nombrecons]] FOREIGN KEY [nombre_indice] (nombre_columna_indice,...);



Ejercicio resuelto

Tenemos creada la siguiente tabla:

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

Ahora queremos poner una restricción a sueldo para que tome valores entre 1000 y 1200, ¿cómo lo harías?

Mostrar retroalimentación

```
ALTER TABLE EMPLEADOS ADD CONSTRAINT emp_sue_CK CHECK (Sueldo BETWEEN 1000 AND 1200);
```

Para saber más: Es interesante que veas más ejemplos de uso del comando ALTER en MySQL. [Ejemplos de ALTER TABLE](#)

Y dependiendo del Sistema de bases de Datos, las restricciones se manejan de diferentes maneras. [Manejo de CONSTRAINT en diferentes Sistemas de Bases de datos](#)

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

Además de con la sentencia CREATE INDEX, en MySQL también podemos crear índices de las siguientes maneras:

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

1. Al mismo tiempo que creamos la tabla con el uso de la opción **INDEX**.

```
CREATE TABLE nombreTabla(  
    campo1 tipoDato,  
    campo2 tipoDato,...  
    INDEX [nombreIndice] (campos);
```

2. Con la sentencia **ALTER TABLE** si es que la tabla ya existe, mediante **ADD INDEX**.

```
ALTER TABLE nombreTabla  
ADD INDEX [nombreIndice] (campos);
```

Las dos sentencias piden el nombre del índice, sin embargo, con la sentencia **CREATE INDEX** el nombre es obligatorio.

Para eliminar un índice es suficiente con poner la instrucción (**tanto para Oracle como MySQL**):

```
DROP INDEX NombreIndice ON NombreTabla;
```

Pero ¿cómo averiguo el nombre de los índices de una tabla? Mediante la sentencia:

```
SHOW INDEX FROM database_nombre.table_nombre;
```

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 FLOAT);
```

Crea un índice con el campo Apellidos, luego elimínalo.

Mostrar retroalimentación

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


11.7. CREACIÓN Y ELIMINACIÓN DE VISTAS.

Como ya se ha comentado anteriormente en el apartado 8, 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.

Para crear una vista en una base de datos utilizaremos la siguiente sintaxis:

```
CREATE [OR REPLACE] VIEW NombreVista AS Sentencia de Consulta;
```

Las palabras clave **CREATE VIEW NombreVista** creará la vista bajo un nombre específico con el cual podremos referirnos a ella. Y para que la vista tenga un sentido completo debemos utilizar al menos una tabla y algunos campos ya creados previamente para que podamos almacenar su estructura. Es decir, estaríamos creando una "tabla virtual" utilizando la estructura de una o más tablas con los campos que necesitamos. Para ello es necesario utilizar las sentencias de consulta.

Aunque todavía no se ha trabajado con la sentencia **SELECT** en SQL simplemente daremos unas pinceladas básicas para poder utilizar y crear vistas.

La sentencia **SELECT** tiene la siguiente sintaxis:

```
SELECT [nombreCampos] FROM nombreTabla [WHERE condiciones] [ORDER BY campo ASC|DESC];
```

Para eliminar una vista utilizamos la palabra reservada **DROP** seguido del elemento que deseamos eliminar, en este caso, **VIEW** e indicando el nombre que le hemos proporcionado.

```
DROP VIEW NombreVista;
```



Ejercicio Resuelto

Imagina que tenemos creada la tabla **EMPLEADOS** con los campos **DNI, nombre, apellidos, ciudad y salario**

Podríamos realizar sentencias de consulta sobre esa tabla como las siguientes:

CONSULTA 1: Selecciona todos los campos de la tabla EMPLEADOS

```
SELECT * FROM EMPLEADOS;
```

CONSULTA 2: Selecciona los campos DNI, Nombre y apellidos de la tabla EMPLEADOS.

```
SELECT DNI, nombre, apellidos FROM EMPLEADOS;
```

CONSULTA 3: Selecciona los nombres y salarios de aquellos empleados que cobran más de 1000€.

```
SELECT Nombre, Salario FROM EMPLEADOS WHERE salario>1000;
```

¿Qué sentencia utilizarías para crear una vista que se llame OficinaAlmería y en la que sólo se vean los nombres y apellidos de los empleados que viven en la ciudad de Almería?

Mostrar retroalimentación

```
CREATE VIEW OficinaAlmeria AS SELECT nombre, apellidos FROM EMPLEADOS WHERE ciudad='Almería';
```

Para saber más: Aunque en la unidad siguiente del módulo se verá en profundidad y se trabajará con la sentencia SELECT, si quieres aprender un poco más puedes acceder a los siguientes enlaces: [Ejemplos de sentencias SELECT utilizando MySQL](#). [Sintaxis de la sentencia SELECT \(página oficial MySQL\)](#)

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, en el caso de Oracle, y en el caso de MySQL están almacenadas y encriptadas en la base de datos 'mysql' y tabla 'user'.

¿Cómo se crean los usuarios? La sintaxis es:

En Oracle:

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 almacenar 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 ;
```

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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.

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.

En MySQL:

```
CREATE USER NombreUsuario@'equipo'  
IDENTIFIED BY 'ClaveAcceso';
```

La sintaxis es:

Donde:

- **CREATE USER:** crea un nombre de usuario que será identificado por el sistema.
- **@:** tras este símbolo aparece el nombre del equipo desde el cual el usuario puede conectarse.
 - Puede ser 'localhost' u otro nombre
 - Puede ser una IP.
 - Puede ser '%' que significa cualquier máquina.
- **IDENTIFIED BY:** permite dar una clave de acceso al usuario creado.

Recuerda que para crear usuarios debes tener una cuenta con privilegios de Administrador.
Para ver todos los usuarios creados, puedes consultar la tabla **mysql.user**
`select * from mysql.user;`

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 con la orden **GRANT** que veremos después.

```
CREATE USER usuario1@localhost IDENTIFIED BY 'limitado';
```

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

```
DROP USER NombreUsuario@'equipo';
```

12.1. PERMISOS.

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.

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

En Oracle:

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

Donde:

```
GRANT {privilegio_objeto [, privilegio_objeto]...[ALL|[PRIVILEGES]]
ON [usuario.]objeto
TO {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 ON 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];
```

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

Donde

- **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.

En MySQL:

Para poder acceder a los objetos de una base de datos necesitas tener privilegios (permisos).

- Los **privilegios sobre objetos** permiten acceder y realizar cambios en las tablas de los esquemas de la base de datos. Por ejemplo, **INSERT**, **SELECT**, **UPDATE** y **DELETE**, son privilegios sobre objetos.
- Los **privilegios de sistema** son los que dan derecho a ejecutar un tipo de comando SQL o a realizar alguna acción sobre objetos de un tipo especificado, por ejemplo, **CREATE USER**, **ALTER**, **CREATE o CREATE VIEW**.

Un privilegio o permiso se otorga con el comando **GRANT** (conceder) una vez creado el usuario con **CREATE USER**

Nota. En versiones de MySQL anteriores a la versión 8.x, el comando GRANT también permitía crear a un usuario en el momento de concederle permisos.

```
GRANT {permiso[(listacolumnas)] [, permiso[(listacolumnas)]...[ALL]}  
ON [tipoobjeto]{nombretabla | *.* | basedatos.*}  
TO {usuario1 [IDENTIFIED BY [PASSWORD] contraseña] [,usuario1 [IDENTIFIED BY [PASSWORD] contraseña] .  
[WITH GRANT OPTION];
```

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

Donde:

- **permiso:** privilegio sobre objeto o sistema.
- **listacolumnas:** columnas sobre las que se concede el permiso.
- **tipoobjeto** =TABLE | FUNCTION | PROCEDURE
- **ON** especifica el objeto sobre el que se conceden los privilegios.
- **TO** señala a los usuarios a los que se conceden privilegios.
- **ALL** concede todos los privilegios sobre el objeto especificado.
- **usuarioN: (nombreUsuario@ubicación)** usuario al que se le concede los permisos. Se especifica indicando un nombre, luego @ y después la ubicación de acceso. La ubicación % hace referncia a cualquier equipo, 'localhost' al equipo local, pero también se puede especificar la dirección IP o nombre del equipo dentro de una red, entre otros.
- **contraseña:** password del usuario.
- **WITH GRANT OPTION:** permite al usuario otorgar permisos a otros usuarios, para ello, el usuario debe tener esos permisos sobre los objetos correspondientes.

Con GRANT también se pueden asignar permisos a Roles. Los Roles están permitidos en MySQL desde la versión 8.x.

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

- Concede al usuario1 con conexión desde localhost, ([usuario1@localhost](#)) insertar datos en la tabla usuarios:
 - **GRANT INSERT ON usuarios TO [usuario1@localhost](#);**
- Concede todos los privilegios sobre la tabla partidas al usuario1 con conexión desde localhost ([usuario1@localhost](#))
 - **GRANT ALL ON partidas TO [usuario1@localhost](#);**

Para saber más

En MySQL, te recomendamos que si quieres conocer más sobre permisos y objetos sobre los que se conceden privilegios, así como los diferentes privilegios que se pueden conceder en MySQL, visita este enlace: [Privilegios en MySQL](#)

Y para ver ejemplos, visita el siguiente enlace: [Gestión de privilegios.](#)

Y para saber cómo gestionar Roles en MySQL 8.x visita el siguiente enlace: [Roles en MySQL 8.0](#)

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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

Con el comando **REVOKE** se retiran los privilegios:

En Oracle:

- Sobre **objetos**:

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

- 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} ...];
```

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;
```

En MySQL:

Con el comando **REVOKE** se retiran los privilegios:

- Sobre **objetos**:

```
REVOKE {permiso[(listacolumnas)] [, permiso]...|ALL}  
ON [tipoobjeto] {nombretabla*.*|basedatos.*}  
FROM {usuario1} [,usuario2] ...;
```

root va a quitar el permiso de consultar y de actualizar sobre la tabla usuarios a **usuario1@localhost**:

```
REVOKE SELECT, UPDATE ON usuarios FROM usuario1@localhost;
```

y va a quitarle el permiso de eliminar usuarios, a usuario1 cuando se conecta desde cualquier ubicación:

```
REVOKE DROP USER FROM usuario1@'%';
```

Debes conocer

Videotutorial resumen de todo lo visto en la Unidad 3: <https://www.youtube.com/watch?v=q0-yEuzHdnA&t=1s>

13. ENLACES DE REFUERZO Y AMPLIACIÓN.

- Tipos de datos en MySQL: <http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=423>
- Documentación oficial del SGBD MySQL: http://docs.oracle.com/cd/E17952_01/index.html
- Ejemplo de creación de una base de datos y una tabla en MySQL con Workbench: <https://jumipe.wordpress.com/2013/06/16/crear-base-de-datos-en-mysql-con-workbenchcreate-data-base-mysql-with-workbench/>
- Ejemplo de conexión mediante consola de comandos en Windows: <http://mysql.conclase.net/curso/?cap=006#>
- Ejemplos de uso del comando ALTER TABLE con MySQL: http://mysql.conclase.net/curso/?sqlsen=ALTER_TABLE
- Documentación oficial del SGBD ORACLE: <https://docs.oracle.com/en/database/database.html>
- Tipos de datos en Oracle: <http://www.tuinformaticafacil.com/oracle-11g/tipos-de-datos-en-oracle-11g>
- Concepto de TABLESPACE en ORACLE: <http://edwinsaldanaabd.blogspot.com.es/2010/04/que-es-un-tablespace-oracle.html>
- Ejemplo de creación de un "TABLESPACE" y asociarle un archivo (DATAFILE) en Oracle: <http://dba.stackexchange.com/questions/1989/how-do-i-create-tablespace-in-oracle-11g>
- Ejemplo de creación de un usuario en ORACLE, asignarle los "TABLESPACES por defecto: user y temp" y otorgar todos lo permisos: <http://jeanmazuelos.com/es/blog/20131102/oracle-database-server-express-edition-11g> (mirar el pto 3.1. Creación de un usuario oracle).
- Documentación oficial del SGBD POSTGRESQL: <http://www.postgresql.org/docs/>
- Uso de la herramienta Query Tool de PgAdminIII para ejecutar Scripts: <http://www.pgadmin.org/docs/dev/query.html>
- Tipos de datos en PostgreSQL: <http://www.postgresql.org.ar/trac/wiki/datatype.html>

ANEXO 1: 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.
DROP	Se utiliza para eliminar tablas e índices.
ALTER	Se utiliza para modificar tablas.

Comandos DML. Lenguaje de Manipulación de Datos.	
Comando:	Descripción:
SELECT	Se utiliza para consultar filas que satisfagan un criterio determinado.
INSERT	Se utiliza para cargar datos en una única operación.
UPDATE	Se utiliza para modificar valores de campos y filas específicos.
DELETE	Se utiliza para eliminar filas de una tabla.

Comandos DCL. Lenguaje de Control de Datos.	
Comando:	Descripción:
GRANT	Permite dar permisos a uno o varios usuarios o roles para realizar tareas determinadas.
REVOKE	Permite eliminar permisos que previamente se han concedido con GRANT.

- 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.
WHERE	Se utiliza para especificar las condiciones que deben reunir las filas que se van a seleccionar.
GROUP BY	Se utiliza para separar las filas seleccionadas en grupos específicos.
HAVING	Se utiliza para expresar la condición que debe satisfacer cada grupo.
ORDER BY	Se utiliza para ordenar las filas seleccionadas de acuerdo a un orden específico.

TEMA 3: IMPLANTACIÓN DE BASES DE DATOS RELACIONALES

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

Operadores:	Descripción:
AND	Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR	Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT	Devuelve el valor contrario de la expresión.

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.

Función:	Descripción:
AVG	Calcula el promedio de los valores de un campo determinado.
COUNT	Devuelve el número de filas de la selección.
SUM	Devuelve la suma de todos los valores de un campo determinado.
MAX	Devuelve el valor más alto de un campo determinado.
MIN	Devuelve el valor mínimo de un campo determinado.

- **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.