

TEMA 4: REALIZACIÓN DE CONSULTAS

REALIZACIÓN DE CONSULTAS	1
1. INTRODUCCIÓN	1
2. LA SENTENCIA SELECT	5
2.1. LA CLAÚSULA SELECT	5
2.2. LA CLAÚSULA FROM	7
2.3. LA CLAÚSULA WHERE	7
2.4. ORDENACIÓN DE REGISTROS. CLAÚSULA ORDER BY	8
3. OPERADORES	10
3.1. OPERADORES DE COMPARACIÓN	10
3.2. OPERADORES ARITMÉTICOS Y DE CONCATENACIÓN	12
3.3. OPERADORES LÓGICOS	13
3.4. PRECEDENCIA	14
4. CONSULTAS CALCULADAS	14
5. FUNCIONES	15
5.1. FUNCIONES NUMÉRICAS	16
5.2. FUNCIONES DE CADENA DE CARACTERES	17
5.3. FUNCIONES DE MANEJO DE FECHAS	19
5.4. FUNCIONES DE CONVERSIÓN	22
5.5. OTRAS FUNCIONES: NVL Y DECODE – IFNULL	23
6. CONSULTAS DE RESUMEN	25
6.1. FUNCIONES DE AGREGADO: SUM Y COUNT	26
6.2. FUNCIONES DE AGREGADO: MIN Y MAX	27
6.3. FUNCIONES DE AGREGADO: AVG, VAR, STDEV Y STDEVP.	28
7. AGRUPAMIENTO DE REGISTROS	29
8. CONSULTAS MULTITABLAS	30
8.1. COMPOSICIONES INTERNAS	32
8.2. COMPOSICIONES EXTERNAS	34
8.3. COMPOSICIONES DE LA VERSION SQL99	35
9. OTRAS CONSULTAS MULTITABLAS: UNIÓN, INTERSECCIÓN Y DIFERENCIA DE CONSULTAS. 37	
10. SUBCONSULTAS	38
11. REALIZANDO UN REPASO MEDIANTE UN EJEMPLO PRÁCTICO	40
12. ENLACES DE REFUERZO Y AMPLIACIÓN	41

REALIZACIÓN DE CONSULTAS



Debes conocer

En esta unidad verás cómo realizar consultas SQL sobre una base de datos MySQL y sobre una base de datos ORACLE. Por ello debes tener en cuenta lo siguiente:

- ✓ Hay aspectos generales del lenguaje SQL que son comunes a ambos sistemas gestores de bases de datos, por tanto cuando no se indique o especifique el SGBD, se entenderá que es información común a ambos sistemas de bases de datos.
- ✓ Para aspectos específicos de un determinado sistema de bases de datos, se indicará explícitamente a qué sistema va referida dicha información.
- ✓ Se utilizarán dos bases de datos para desarrollar los ejemplos de esta unidad. Una base de datos para ORACLE y otra diferente para MySQL.
- ✓ Los ejemplos para cada base de datos vendrán precedidos por el texto destacado en azul **MySQL** o bien **Oracle**.

1. INTRODUCCIÓN

En unidades anteriores has aprendido que SQL es un conjunto de sentencias u órdenes que se necesitan para acceder a los datos. Este lenguaje es utilizado por la mayoría de las aplicaciones donde se trabaja con datos para acceder a ellos. Es decir, es la vía de comunicación entre el usuario y la base de datos.

SQL nació a partir de la publicación "A relational model of data for large shared data banks" de Edgar Frank

Codd. IBM aprovechó el modelo que planteaba Codd para desarrollar un lenguaje acorde con el recién nacido modelo relacional, a este primer lenguaje se le llamó SEQUEL (Structured English QUERy Language). Con el tiempo SEQUEL se convirtió en SQL (Structured Query Language). En 1979, la empresa Relational Software sacó al mercado la primera implementación comercial de SQL. Esa empresa es la que hoy conocemos como Oracle.

Actualmente SQL sigue siendo el estándar en lenguajes de acceso a base de datos relacionales.

En 1992, ANSI e ISO completaron la estandarización de SQL y se definieron las sentencias básicas que debía contemplar SQL para que fuera estándar. A este SQL se le denominó ANSI-SQL o SQL92.

Hoy en día todas las bases de datos comerciales cumplen con este estándar, eso sí, cada fabricante añade sus mejoras al lenguaje SQL.

La primera fase del trabajo con cualquier base de datos comienza con sentencias DDL (en español Lenguaje de Definición de Datos), puesto que antes de poder almacenar y recuperar información debimos definir las estructuras donde agrupar la información: las tablas.

La siguiente fase será manipular los datos, es decir, trabajar con sentencias DML (en español Lenguaje de Manipulación de Datos). Este conjunto de sentencias está orientado a consultas y manejo de datos de los objetos creados. Básicamente consta de cuatro sentencias: **SELECT**, **INSERT**, **DELETE** y **UPDATE**. En esta unidad nos centraremos en una de ellas, que es la sentencia para consultas: **SELECT**.

En Oracle.

Las sentencias SQL que se verán a continuación pueden ser ejecutadas desde el entorno web **Application Express** de Oracle utilizando el botón **SQL Workshop** en la página de inicio o home, y elegir **SQL Commands**.

También se pueden indicar las sentencias SQL desde el entorno de **SQL*Plus** que ofrece Oracle y que puedes encontrar en **Inicio > Todos los programas > Oracle Database 11g Express Edition > Run SQL Command Line**

Si optas por abrir esa aplicación (**Ejecutar Línea de Comandos SQL**), el primer paso que debe realizarse para manipular los datos de una determinada tabla, es conectarse utilizando un nombre de usuario con los permisos necesarios para hacer ese tipo de operaciones a la tabla deseada. Utiliza para ello la orden **CONNECT** seguida del nombre de usuario. Posteriormente, solicitará la contraseña correspondiente a dicho usuario.

Para ejecutar cualquiera de las sentencias SQL que aprenderás en los siguientes puntos, simplemente debes escribirla completa y pulsar **Intro** para que se inicie su ejecución.

En MySQL

Las **sentencias SQL** que se verán a continuación pueden ser **redactadas y ejecutadas**:

- **Desde cualquier cliente gráfico** como por ejemplo la herramienta **Workbench** de MySQL. En este caso, redactamos la sentencia en una ventana de **SQL** (*Menú File>>New Query Tab*), y la ejecutamos accediendo a *Menú Query>>Execute Current Statement* o mediante el botón '*rayo amarillo con cursor*'.
- **Desde la línea de comandos** (*cliente en modo texto*) de MySQL y que puedes lanzar desde:
 - **Inicio > Todos los programas > MySQL > Command Line Client**. En este caso te pedirá la contraseña del usuario *root* Introduce su contraseña y después *Intro*. El símbolo de sistema *mysql>* te indica que ya estás conectado al servidor de bases de datos y puedes redactar sentencias SQL Una vez redactada la sentencia SQL pulsa la tecla *INTRO* para que se ejecute.

En ambos casos, lo primero que habrá que hacer es iniciar o arrancar el servidor de bases de datos MySQL, si no está iniciado por defecto, y conectar o abrir conexión con el servidor utilizando un nombre de usuario con los permisos necesarios para hacer ese tipo de operaciones sobre la base de datos y tablas deseadas.

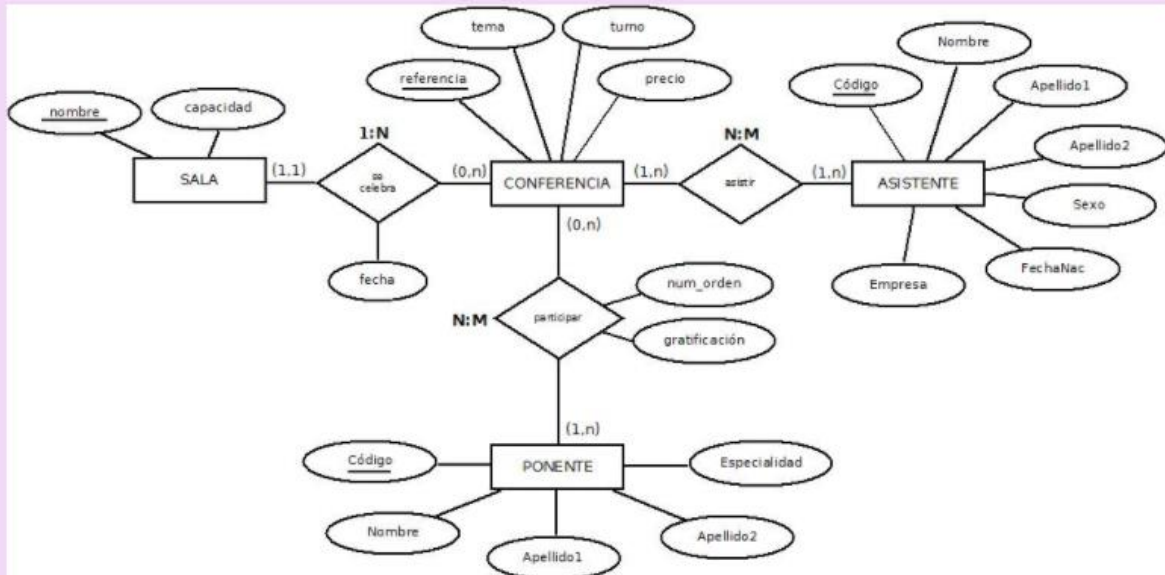
Como usuario, utiliza el usuario **root** con la contraseña que le hayas puesto en la instalación de MySQL.

- **Conexión desde Workbench**. Ejecuta Workbench y pulsa sobre la ventana de conexión a tu servidor como usuario *root*. Si le has puesto contraseña, aparecerá una ventana para que la introduzcas. Una vez introducida la contraseña ya estarás conectado o conectada al servidor MySQL.
- **Conexión desde la línea de comandos**.
 - Como te hemos indicado anteriormente. Una vez que has pulsado sobre **Command Line Client**, te pedirá la contraseña del usuario *root*. Introduce su contraseña y después pulsa *Intro*. El símbolo de sistema *mysql>* te indicará que ya estás conectado al servidor de bases de datos y puedes redactar sentencias SQL.
 - Abriendo una ventana de consola del sistema operativo y ejecutando la orden **mysql -u root -p**, te pedirá la contraseña del usuario *root*. Introduce su contraseña y después pulsa *Intro*. El símbolo de sistema *mysql>* te indicará que ya estás conectado al servidor de bases de datos y puedes redactar sentencias SQL.

TEMA 4: REALIZACIÓN DE CONSULTAS

ORACLE

El diagrama Entidad - Relación de la base de datos con la que vamos a trabajar en Oracle sería:



Y el modelo relacional (paso a tablas) resultante del diagrama sería:

SALA (nombre, capacidad)

CONFERENCIA (referencia, tema, precio, fecha, turno, nom_sala)

PARTICIPAR (codPonente, refConferencia, num_orden, gratificacion)

PONENTE (código, nombre, apellido1, apellido2, especialidad)

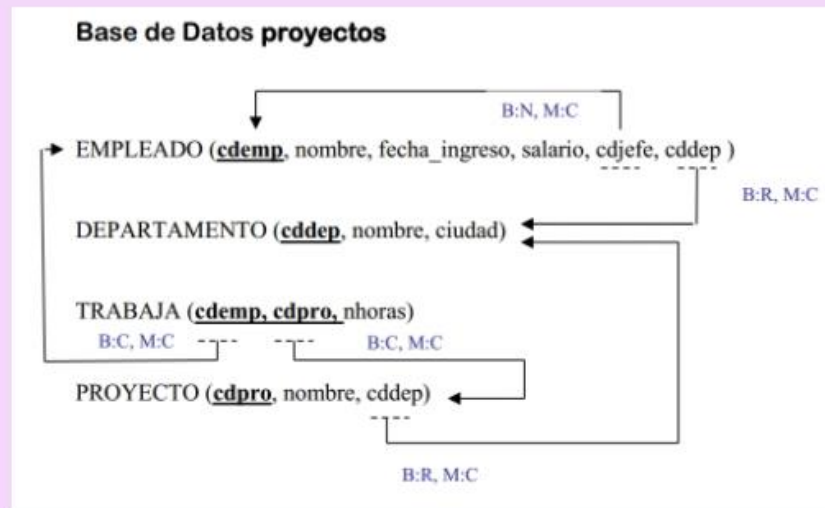
ASISTIR (codAsistente, refConferencia)

ASISTENTE (código, nombre, apellido1, apellido2, sexo, fechaNac, empresa)

TEMA 4: REALIZACIÓN DE CONSULTAS

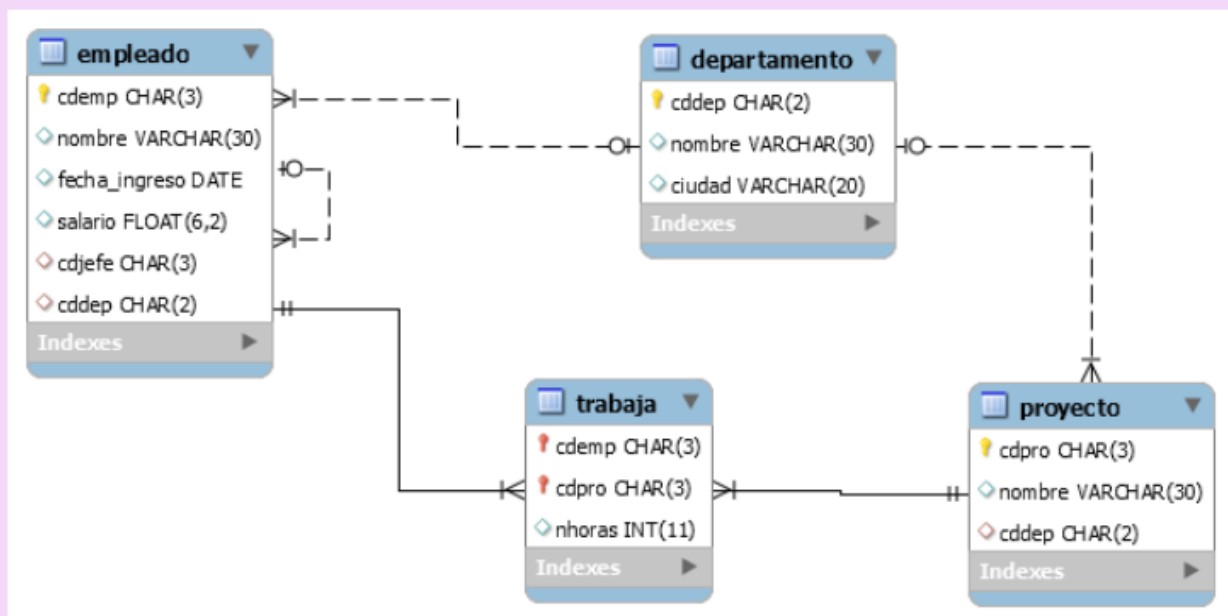
MYSQL

El grafo relacional (paso a tablas), en modo texto, de la base de datos con la que trabajaremos en MySQL es el siguiente:



y su modelo relacional diseñado con el Software Workbench es el siguiente:

y su modelo relacional diseñado con el Software Workbench es el siguiente:



2. LA SENTENCIA SELECT

¿Cómo podemos seleccionar los datos que nos interesen dentro de una base de datos? Para recuperar o seleccionar los datos, de una o varias tablas puedes valerte del lenguaje SQL, para ello utilizarás la sentencia **SELECT**, que consta de cuatro partes básicas:

- **Cláusula SELECT** seguida de la descripción de lo que se desea ver, es decir, de los nombres de las columnas que quieres que se muestren separadas por comas simples (", "). Esta parte es obligatoria.
- **Cláusula FROM** seguida del nombre de las tablas de las que proceden las columnas de arriba, es decir, de donde vas a extraer los datos. Esta parte también es obligatoria.
- **Cláusula WHERE** seguida de un criterio de selección o condición. Esta parte es opcional.
- **Cláusula ORDER BY** seguida por un criterio de ordenación. Esta parte también es opcional.

Por tanto, una primera sintaxis quedaría de la siguiente forma:

```
SELECT [ALL | DISTINCT] columna1, columna2, ... FROM tabla1, tabla2, ... WHERE condición1, condición2, ... ORDER BY ordenación;
```



Recomendación

Las cláusulas **ALL** y **DISTINCT** son opcionales.

- ✓ Si incluyes la cláusula **ALL** después de **SELECT**, indicarás que quieres seleccionar todas las filas estén o no repetidas. Es el valor por defecto y no se suele especificar.
- ✓ Si incluyes la cláusula **DISTINCT** después de **SELECT**, se suprimirán aquellas filas del resultado que tengan igual valor que otras.

2.1. LA CLAÚSULA SELECT

Ya has visto que a continuación de la sentencia **SELECT** debemos especificar cada una de las columnas que queremos seleccionar. Además, debemos tener en cuenta lo siguiente:

- **Se pueden nombrar** a las columnas anteponiendo el nombre de la tabla de la que proceden, pero esto es opcional y quedaría: **NombreTabla.NombreColumna**
- Si queremos **incluir todas las columnas** de una tabla podemos utilizar el comodín **asterisco (*)**. Quedaría así: **SELECT * FROM NombreTabla;**
- También podemos **ponerle alias a los nombres** de las columnas. Cuando se consulta una base de datos, los nombres de las columnas se usan como cabeceras de presentación. Si éste resulta largo, corto o poco descriptivo, podemos usar un alias. Para ello a continuación del nombre de la columna ponemos entre comillas dobles el alias que demos a esa columna. Veamos un ejemplo:

```

-- ORACLE: SELECT nombre "Nombre de la Sala" FROM SALA;
-- MySQL:  SELECT nombre "Nombre empleado" FROM empleado;
           SELECT nombre as "Nombre empleado" FROM empleado;
```

TEMA 4: REALIZACIÓN DE CONSULTAS

- También podemos **sustituir el nombre** de las columnas por constantes, expresiones o funciones SQL.

✚ **ORACLE:** `SELECT 4*3/100 "MiExpresión",capacidad FROM SALA;`

✚ **MySQL:** `SELECT 4*3/100 "MiExpresión", nombre FROM empleado;`

Para saber más

Si quieres conocer algo más sobre esta sentencia y ver algunos ejemplos del uso de **SELECT** aquí tienes el siguiente enlace: [La cláusula SELECT](#).



Ejercicio resuelto

Oracle

Si quieres practicar algunos ejercicios puedes ayudar a **Ana** con algunas consultas. Para ello te facilitamos las tablas que ha creado recientemente para la base de datos con la que actualmente están trabajando. En el siguiente documento tienes los datos que te permitirán realizar algunos ejemplos de esta unidad. Recuerda que los ejemplos irán apareciendo bajo el epígrafe Ejercicio Resuelto.

[Tablas y registros para realizar algunos de los ejemplos.](#)

También tienes algunos datos incluidos para probar las distintas consultas que crees. A partir de ahora nos referiremos a estos datos como tablas de la base de datos de Conferencias.

Por tanto lo primero que tienes que hacer es abrir el editor de SQL, para ello debes ir en el menú de Inicio de Windows a Oracle Database 11g Express Edition y a continuación pulsar en Run SQL Command Line. Aparecerá una pantalla donde tienes que realizar los siguientes pasos:

1. Conectarte a través de un usuario.
2. Ejecutar el archivo que has bajado, para ello debes escribir `@Ruta_donde_se_encuentra_el_archivo/BD04_CONT_R07_02.sql`

En este ejercicio te pedimos que ejecutes el archivo y crees las tablas necesarias para poder realizar ejercicios posteriores.

MySQL

Si quieres practicar algunos ejercicios puedes ayudar a **Ana** con algunas consultas. Para ello te facilitamos las tablas que ha creado recientemente para la base de datos con la que actualmente están trabajando. En el siguiente enlace tienes los datos que te permitirán realizar algunos ejemplos de esta unidad. Recuerda que los ejemplos irán apareciendo bajo el epígrafe Ejercicio Resuelto.

[Tablas y registros para realizar algunos de los ejemplos.](#)

También tienes algunos datos incluidos para probar las distintas consultas que crees. A partir de ahora nos referiremos a estos datos como tablas de la empresa Proyectosx. Te descargarás un archivo comprimido de nombre `bd_proyectosx.zip`.

Descomprime el archivo y dentro encontrarás el script SQL `bd_proyectosx.sql`

- 1.- Por tanto lo primero que tienes que hacer es abrir el editor de SQL, para ello debes de tener iniciado el servidor MySQL y desde *Workbench*, iniciar una conexión al servidor como usuario *root*.
- 2.- Desde el menú *File>>Open SQL Script* indica la ruta del archivo `bd_proyectosx.sql` y ejecuta el script.

En este ejercicio te pedimos que ejecutes el archivo y crees las tablas necesarias para poder realizar y probar los ejercicios de esta unidad. Este **script** creará la base de datos **proyectosx**, con la que podrás probar los ejemplos de esta unidad.

Mostrar retroalimentación

El resultado puedes verlo en el **Anexo I.- Creación de tablas y preparación de Oracle y MySQL** al final de los contenidos.

2.2. LA CLAÚSULA FROM

Al realizar la consulta o selección has visto que puedes elegir las columnas que necesites, pero ¿de dónde extraigo la información?

En la sentencia **SELECT** debemos establecer de dónde se obtienen las columnas que vamos a seleccionar, para ello disponemos en la sintaxis de la cláusula **FROM**.

Por tanto, en la cláusula **FROM** se definen los nombres de las tablas de las que proceden las columnas.

Si se utiliza más de una, éstas deben aparecer separadas por comas. A este tipo de consulta se denomina **consulta combinada** o **join**. Más adelante verás que para que la consulta combinada pueda realizarse, necesitaremos aplicar una condición de combinación a través de una cláusula **WHERE**.


En el caso de Oracle, también puedes añadir el nombre del usuario que es **propietario** de esas tablas, indicándolo de la siguiente manera:

USUARIO.TABLA

de este modo podemos distinguir entre las tablas de un usuario y otro (ya que esas tablas pueden tener el mismo nombre).

Tanto en Oracle como en MySQL, puedes asociar un alias a las tablas para abreviar, en este caso **no es necesario que lo encierres entre comillas**.

 **ORACLE:** SELECT * FROM CONFERENCIA C;


 **MySQL:** SELECT * FROM empleado e;

2.3. LA CLAÚSULA WHERE

¿Podríamos desear seleccionar los datos de una tabla que cumplan una determinada condición? Hasta ahora hemos podido ver la sentencia **SELECT** para obtener todas o un subconjunto de columnas de una o varias tablas. Pero esta selección afectaba a todas las filas (registros) de la tabla. Si queremos restringir esta selección a un subconjunto de filas debemos especificar una condición que deben cumplir aquellos registros que queremos seleccionar. Para poder hacer esto vamos a utilizar la cláusula **WHERE**.


A continuación de la palabra **WHERE** será donde pongamos la condición que han de cumplir las filas para salir como resultado de dicha consulta.

El criterio de búsqueda o condición puede ser más o menos sencillo y para crearlo se pueden conjugar operadores de diversos tipos, funciones o expresiones más o menos complejas.

 **Oracle:** Si en nuestra tabla **ASISTENTE**, necesitáramos un listado de los asistentes que sean hombres, bastaría con crear la siguiente consulta:

```
SELECT nombre, apellido1, apellido2
      FROM ASISTENTE
     WHERE sexo = 'H';
```


TEMA 4: REALIZACIÓN DE CONSULTAS

 **MySQL:** Si en nuestra tabla **empleado**, necesitáramos un listado de los empleados con salario inferior a 2000€, bastaría con crear la siguiente consulta:

```
SELECT nombre
FROM empleado
WHERE salario < 2000;
```

Más adelante te mostraremos los operadores con los que podrás crear condiciones de diverso tipo.

Para saber más

Aquí te adelantamos los operadores para que vayas conociéndolos. Con ellos trabajarás cuando hayas adquirido algunos conocimientos más: [Operadores SQL ORACLE](#) | [Operadores SQL MySQL](#)

2.4. ORDENACIÓN DE REGISTROS. CLAÚSULA ORDER BY


En la consulta del ejemplo anterior hemos obtenido una lista de nombres y apellidos de las usuarias de nuestro juego. Sería conveniente que aparecieran ordenadas por apellidos, ya que siempre quedará más profesional además de más práctico. De este modo, si necesitáramos localizar un registro concreto la búsqueda sería más rápida. ¿Cómo lo haremos? Para ello usaremos la cláusula **ORDER BY**.

ORDER BY se utiliza para **especificar el criterio de ordenación** de la respuesta a nuestra consulta. Tendríamos:

```
SELECT [ALL | DISTINCT] columna1, columna2, ...
FROM tabla1, tabla2, ...
WHERE condición1, condición2, ...
ORDER BY columna1 [ASC | DESC], columna2 [ASC | DESC], ..., columnaN [ASC | DESC];
```

Después de cada columna de ordenación se puede incluir el tipo de ordenación (ascendente o descendente) utilizando las palabras reservadas **ASC** o **DESC**. Por defecto, y si no se pone nada, la ordenación es ascendente.

Debes saber que es **posible ordenar por más de una columna**. Es más, puedes ordenar no solo por columnas sino a través de una expresión creada con columnas, una constante (aunque no tendría mucho sentido) o funciones SQL.

 **Oracle:** En el siguiente ejemplo, ordenamos por apellidos y en caso de empate por nombre:

```
SELECT nombre, apellido1, apellido2
FROM ASISTENTE
ORDER BY apellido1, apellido2, nombre;
```

 **MySQL:** En el siguiente ejemplo, ordenamos por departamento y en caso de empate por nombre:

```
SELECT nombre, cddep
FROM empleado
ORDER BY cddep, nombre;
```

TEMA 4: REALIZACIÓN DE CONSULTAS

Puedes colocar el número de orden del campo por el que quieres que se ordene en lugar de su nombre, es decir, referenciar a los campos por su posición en la lista de selección.

✚ **Oracle:** Por ejemplo, si queremos el resultado del ejemplo anterior ordenado por la Empresa:

```
SELECT nombre, apellido1, apellido2, empresa
FROM ASISTENTE
ORDER BY 4;
```

✚ **MySQL:** Por ejemplo, si queremos el resultado del ejemplo anterior ordenado por salario:

```
SELECT nombre, cddep, salario
FROM empleado
ORDER BY 3;
```

Si colocamos un número mayor a la cantidad de campos de la lista de selección, aparece un mensaje de error y la sentencia no se ejecuta.

¿Se puede utilizar cualquier tipo de datos para ordenar? No todos los tipos de campos te servirán para ordenar, únicamente aquellos de tipo **carácter, número o fecha**.



Ejercicio resuelto

Oracle

Utilizando las tablas y datos descargados anteriormente, vamos a realizar una consulta donde obtengamos el nombre y la capacidad de las salas ordenados de mayor a menor capacidad.

Mostrar retroalimentación

```
SELECT NOMBRE, CAPACIDAD FROM SALA ORDER BY CAPACIDAD DESC;
```

MySQL

Utilizando las tablas y datos de la empresa Proyectosx descargados anteriormente, vamos a realizar una consulta donde obtengamos de la tabla EMPLEADO, el código, nombre y salario de los empleados ordenados por nombre ascendentemente y por salario de manera descendente.

Mostrar retroalimentación

```
SELECT cdemp, nombre, salario
FROM empleado
ORDER BY nombre, salario DESC;
```

3. OPERADORES

Veíamos que en la cláusula **WHERE** podíamos incluir expresiones para filtrar el conjunto de datos que queríamos obtener. Para crear esas expresiones necesitas utilizar distintos operadores de modo que puedas comparar, utilizar la lógica o elegir en función de una suma, resta, etc.

Los operadores son símbolos que permiten realizar operaciones matemáticas, concatenar cadenas o hacer comparaciones.

Oracle reconoce 4 tipos de operadores:

1. Relacionales o de comparación.
2. Aritméticos.
3. De concatenación.
4. Lógicos.

Para saber más

Si quieres conocer un poco más sobre los operadores visita este enlace: [Operadores](#).

3.1. OPERADORES DE COMPARACIÓN

Los puedes conocer con otros nombres como **relacionales**, nos **permitirán comparar expresiones**, que pueden ser valores concretos de campos, variables, etc.

Los operadores de comparación son símbolos que se usan como su nombre indica para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa.

Tenemos los siguientes operadores y su operación:

OPERADOR	SIGNIFICADO
=	Igualdad.
!=, <>, ^=	Desigualdad. (En el caso de MySQL, sólo != y <>)
<	Menor que.
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
IN	Igual que cualquiera de los miembros entre paréntesis. <code>IN (miembro1, miembro2,...)</code>
NOT IN	Distinto que cualquiera de los miembros entre paréntesis. <code>NOT IN (miembro1, miembro2,...)</code>
BETWEEN	Entre. Contenido dentro del rango, incluidos valorMin y valorMax. <code>BETWEEN valorMin AND valorMax</code>
NOT BETWEEN	Fuera del rango. <code>NOT BETWEEN valorMin AND valorMax</code>
LIKE '_abc%'	Se utiliza sobre todo con textos y permite obtener columnas cuyo valor en un campo cumpla una condición textual. Utiliza una cadena que puede contener los símbolos "%" que sustituye a un conjunto de caracteres o "_" que sustituye a un carácter.
IS NULL	Devuelve verdadero si el valor del campo de la fila que examina es nulo.

TEMA 4: REALIZACIÓN DE CONSULTAS

- ✚ **Oracle:** Si queremos obtener el nombre completo de los ponentes cuyo código comienza por ESP ordenados alfabéticamente por el primer apellido sería:

```
SELECT nombre, apellido1, apellido2 FROM PONENTE WHERE codigo LIKE 'ESP%' ORDER BY apellido1;
```

- ✚ **MySQL:** Si queremos obtener aquellos empleados cuyo salario es superior a 1000€ podemos crear la siguiente consulta:

```
SELECT nombre, salario FROM empleado WHERE salario > 1000;
```

/*Ahora queremos aquellos empleados cuyo nombre comienza por A y su sueldo es mayor de 1000€ y ordenados por salario*/

```
SELECT nombre, salario FROM empleado WHERE salario > 1000 AND nombre LIKE 'A%' ORDER BY salario;
```



Ejercicio resuelto

Oracle

Utilizando las tablas y datos descargados anteriormente, vamos a realizar una consulta donde obtengamos el tema y la fecha de las conferencias que tengan turno de tarde y se celebren en la sala Apolo o Zeus

Mostrar retroalimentación

```
SELECT tema, fecha FROM CONFERENCIA WHERE turno='T' AND (sala IN ('Apolo', 'Zeus'));
```

MySQL

Utilizando las tablas y datos de la empresa **Proyectosx** descargados anteriormente, vamos a realizar:

- a) Una consulta donde obtengamos los departamentos de Sevilla o Cádiz.
b) Una consulta para buscar todos los empleados cuyo código tiene como primera letra cualquiera, luego el número 1 y después cualquier conjunto de caracteres.

```
/*a) Una consulta donde obtengamos los departamentos de Sevilla o Cádiz.*/SELECT cddep, nombre, ciudad FROM departamento  
WHERE ciudad IN ('SEVILLA', 'CÁDIZ');
```

Fíjate que buscará aquellas ciudades que coincidan textualmente con las que ponemos entre comillas, que pueden ser simples o dobles. Esa sentencia es equivalente a escribirla con el operador OR:

```
/*a) Una consulta donde obtengamos los departamentos de Sevilla o Cádiz.*/SELECT cddep, nombre, ciudad FROM departamento  
WHERE ciudad = 'SEVILLA' OR ciudad = 'CÁDIZ';
```

NOTA: en Bases de Datos, para este tipo de comparaciones, en vez de utilizar el operador OR, se utiliza mejor el operador IN ().

```
/*b) Una consulta para buscar todos los empleados cuyo código tiene como primera letra cualquiera, luego el número 1 y después cualquier conjunto de caracteres.*/SELECT nombre, cdemp  
FROM empleado WHERE cdemp LIKE '_1%';
```

Para saber más

Puedes ver más operadores que se utilizan en MySQL en el siguiente enlace: [Operadores de comparación en MySQL.](#)

Los operadores para trabajar con cadenas que se utilizan en MySQL puedes verlos en el siguiente enlace. [Operadores con cadenas en MySQL.](#)


Para realizar búsqueda de patrones de caracteres se puede utilizar lo que se denominan Expresiones Regulares o REGEXP. en esta caso, se pueden plantear búsquedas más concretas y complejas. En el siguiente enlace, dispones de ejemplos de uso de expresiones REGEX en MySQL: [Ejemplos de expresiones regulares en MySQL](#)

3.2. OPERADORES ARITMÉTICOS Y DE CONCATENACIÓN


Aprendimos que los operadores son símbolos que permiten realizar distintos tipos de operaciones. Los operadores aritméticos permiten realizar cálculos con valores numéricos. Son los siguientes:

OPERADOR	SIGNIFICADO
+	Suma
-	Resta
*	Multiplicación
/	División

Utilizando expresiones con operadores **es posible obtener** salidas en las cuales **una columna sea el resultado de un cálculo** y no un campo de una tabla.

-  **Oracle:** Mira este ejemplo sobre la tabla CONFERENCIA en el que obtenemos el precio de la conferencia aumentado en un 5 % de aquellas conferencias cuyo precio sea menor a 15 €:

```
SELECT precio*1,05
FROM CONFERENCIA
WHERE precio<15;
```


-  **MySQL:** Mira este ejemplo sobre una tabla TRABAJADORES en el que obtenemos el salario aumentado en un 5 % de aquellos trabajadores que cobran menos de 2000 €:

```
SELECT salario*1.05
FROM empleado
WHERE salario<=2000;
```

Cuando una expresión aritmética se calcula sobre valores **NULL**, el resultado es el propio valor **NULL**.

En Oracle, para concatenar cadenas de caracteres existe el operador de concatenación (" || "). Oracle puede convertir automáticamente valores numéricos a cadenas para una concatenación.


En MySQL se utiliza la función **CONCAT(cadena1, cadena2, ...)**

-  **Oracle:** En la tabla PONENTE tenemos separados en dos campos el primer y segundo apellido de los ponentes, si necesitáramos mostrarlos juntos podríamos crear la siguiente consulta:

```
SELECT nombre, apellido1 || apellido2
FROM PONENTE;
```

Si queremos dejar un espacio entre un apellido y otro, debemos concatenar también el espacio en blanco de la siguiente manera:

```
SELECT nombre, apellido1 || ' ' || apellido2
FROM PONENTE;
```

-  **MySQL:** Para obtener en una sola columna el código de empleado y su nombre, separados ambos por un espacio en blanco, utilizamos la función CONCAT() de la siguiente forma:

```
SELECT CONCAT(cdemp, " ", nombre)
FROM empleado;
```


Para saber más

Los operadores que se utilizan en MySQL puedes verlos en el siguiente enlace: [Operadores aritméticos en MySQL](#) | [La función de concatenación CONCAT\(\) de MySQL](#)


3.3. OPERADORES LÓGICOS

Habr  ocasiones en las que tengas que evaluar m s de una expresi n y necesites verificar que se cumple una  nica condici n, otras veces comprobar si se cumple una u otra o ninguna de ellas. Para poder hacer esto utilizaremos los operadores l gicos.

OPERADOR	SIGNIFICADO
AND	Devuelve verdadero si sus expresiones a derecha e izquierda son ambas verdaderas.
OR	Devuelve verdadero si alguna de sus expresiones a derecha o izquierda son verdaderas.
NOT	Invierte la l�gica de la expresi�n que le precede, si la expresi�n es verdadera devuelve falsa y si es falsa devuelve verdadera.

 **Oracle:** Si queremos obtener aquellas salas cuya capacidad est  entre 100 y 300 personas ser :

```
SELECT nombre, capacidad
FROM SALA
WHERE capacidad>=100 AND capacidad<=300;
```

 **MySQL:** Si queremos obtener aquellos empleados en cuyo salario sea menor o igual a 2000  o superior a 2500 :

```
SELECT nombre, salario
FROM empleado
WHERE salario <=2000 OR salario>2500;
```



Ejercicio resuelto

Oracle

Utilizando las tablas y datos descargados anteriormente, vamos a realizar una consulta donde obtengamos el primer apellido y el nombre de todos los asistentes que no trabajan en "BK Programaci n"

Mostrar retroalimentaci n

```
SELECT apellido1, nombre FROM ASISTENTE WHERE empresa is NULL or empresa NOT IN('BK Programaci n');
```

MySQL

Utilizando las tablas y datos de la empresa Proyectosx descargados anteriormente, vamos a realizar una consulta donde obtengamos todos los nombres de departamentos menos los de 'Sevilla' o 'Granada'.

Mostrar retroalimentaci n

```
SELECT nombre, ciudad
FROM departamento
WHERE ciudad NOT IN ('Sevilla', 'Granada');
```

3.4. PRECEDENCIA

Con frecuencia utilizaremos la sentencia **SELECT** acompañada de expresiones muy extensas y resultará difícil saber que parte de dicha expresión se evaluará primero, por ello es conveniente conocer el orden de precedencia que tiene Oracle:

1. Se evalúa la multiplicación (*) y la división (/) al mismo nivel.
2. A continuación sumas (+) y restas (-).
3. Concatenación (||). (**en el caso de Oracle**)
4. Todas las comparaciones (<, >, ...).
5. Después evaluaremos los operadores **IS NULL, IN NOT NULL, LIKE, BETWEEN**.
6. **NOT**.
7. **AND**.
8. **OR**.

Si quisiéramos variar este orden necesitaríamos utilizar paréntesis.

Para saber más

Aquí puedes comprobar el orden de precedencia en MySQL: [Precedencia en MySQL](#).

4. CONSULTAS CALCULADAS

En algunas ocasiones es interesante realizar operaciones con algunos campos para obtener información derivada de éstos. Si tuviéramos un campo Precio, podría interesarnos calcular el precio incluyendo el IVA o si tuviéramos los campos Sueldo y Paga Extra, podríamos necesitar obtener la suma de los dos campos. Estos son dos ejemplos simples pero podemos construir expresiones mucho más complejas. Para ello haremos uso de la creación de campos calculados.

Los operadores aritméticos se pueden utilizar para hacer cálculos en las consultas.

Estos **campos calculados** se obtienen a través de la sentencia **SELECT** poniendo a continuación la expresión que queramos. Esta consulta **no modificará los valores originales** de las columnas ni de la tabla de la que se está obteniendo dicha consulta, únicamente mostrará una columna nueva con los valores calculados. Por ejemplo:

 **Oracle:**

```
SELECT tema, precio, precio + 10
FROM CONFERENCIA;
```

Con esta consulta hemos creado un campo que tendrá como nombre la expresión utilizada. Podemos ponerle un alias a la columna creada añadiéndolo detrás de la expresión junto con la palabra **AS**. En nuestro ejemplo quedaría de la siguiente forma:

```
SELECT tema, precio, precio + 10 AS PrecioNuevo
FROM CONFERENCIA;
```


MySQL:

```
SELECT nombre, salario, salario + 25  
FROM empleado;
```

Con esta consulta hemos creado un campo que tendrá como nombre la expresión utilizada. Podemos ponerle un alias a la columna creada añadiéndolo detrás de la expresión junto con la palabra **AS**.

En nuestro ejemplo quedaría de la siguiente forma:

```
SELECT nombre, salario, salario + 25 AS SalarioNuevo  
FROM empleado;
```

5. FUNCIONES

¿Has pensado en todas las operaciones que puedes realizar con los datos que guardas en una base de datos? Seguro que son muchísimas. Pues bien, en casi todos los Sistemas Gestores de Base de Datos existen funciones ya creadas que facilitan la creación de consultas más complejas. Dichas funciones varían según el SGBD, veremos aquí las que utiliza Oracle.

Las funciones son realmente operaciones que se realizan sobre los datos y que realizan un determinado cálculo. Para ello necesitan unos datos de entrada llamados parámetros o argumentos y en función de éstos, se realizará el cálculo de la función que se esté utilizando. Normalmente los parámetros se especifican entre paréntesis.

Las funciones se pueden incluir en las cláusulas **SELECT**, **WHERE** y **ORDER BY**.

Las funciones se especifican de la siguiente manera:

```
NombreFunción [(parámetro1, [parámetro2, ...])]
```

Puedes anidar funciones dentro de funciones.

Existe una gran variedad para cada tipo de datos: numéricas, de cadenas de caracteres, de manejo de fechas, de conversión, etc.

Oracle proporciona una tabla con la que podemos hacer pruebas, esta tabla se llama Dual y contiene un único campo llamado **DUMMY** y una sola fila.

MySQL proporciona para pruebas la base de datos de nombre Test. Esta base de datos no contiene ninguna tabla.

En Oracle, podremos utilizar la tabla Dual en algunos de los ejemplos que vamos a ver en los siguientes apartados.


5.1. FUNCIONES NUMÉRICAS

¿Cómo obtenemos el cuadrado de un número o su valor absoluto? Nos referimos a valores numéricos y por tanto necesitaremos utilizar funciones numéricas.

Para trabajar con campos de tipo número tenemos las siguientes funciones:

- **ABS(n)**: Calcula el valor absoluto de un número n.

 **Oracle:** `SELECT ABS(-17) FROM DUAL; -- Resultado: 17`

 **MySQL:** `SELECT ABS(-17); -- Resultado: 17`

Observa que con MySQL no es necesario utilizar ninguna tabla. Basta con hacer la SELECT a la función. El resto de ejemplos expuestos funcionan igual. En MySQL los puedes probar sin poner la tabla DUAL.

- **EXP(n)**: Calcula en, es decir, el exponente en base e del número n.

`SELECT EXP(2) FROM DUAL; -- Resultado: 7,38`

- **CEIL(n)**: Calcula el valor entero inmediatamente superior o igual al argumento n.

`SELECT CEIL(17.4) FROM DUAL; -- Resultado: 18`

- **FLOOR(n)**: Calcula el valor entero inmediatamente inferior o igual al parámetro n.

`SELECT FLOOR(17.4) FROM DUAL; -- Resultado: 17`

- **MOD(m,n)**: Calcula el resto resultante de dividir m entre n.

`SELECT MOD(15, 2) FROM DUAL; --Resultado: 1`

- **POWER(valor, exponente)**: Eleva el valor al exponente indicado.

`SELECT POWER(4, 5) FROM DUAL; -- Resultado: 1024`

- **ROUND(n, decimales)**: Redondea el número n al siguiente número con el número de decimales que se indican.

`SELECT ROUND(12.5874, 2) FROM DUAL; -- Resultado: 12.59`

- **SQRT(n)**: Calcula la raíz cuadrada de n.

`SELECT SQRT(25) FROM DUAL; --Resultado: 5`

- **TRUNC(m,n)**: Trunca un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si "n" es negativo, el número es truncado desde la parte entera.

`SELECT TRUNC(127.4567, 2) FROM DUAL; -- Resultado: 127.45`

`SELECT TRUNC(4572.5678, -2) FROM DUAL; -- Resultado: 4500`

`SELECT TRUNC(4572.5678, -1) FROM DUAL; -- Resultado: 4570`

`SELECT TRUNC(4572.5678) FROM DUAL; -- Resultado: 4572`

- **SIGN(n)**: Si el argumento "n" es un valor positivo, retorna 1, si es negativo, devuelve -1 y 0 si es 0.

`SELECT SIGN(-23) FROM DUAL; -- Resultado: -1`

5.2. FUNCIONES DE CADENA DE CARACTERES

Ya verás como es muy común manipular campos de tipo carácter o cadena de caracteres. Como resultado podremos obtener caracteres o números. Estas son las funciones más habituales:

Oracle

- **CHR(n)**: Devuelve el carácter cuyo valor codificado es n.

```
SELECT CHR(81) FROM DUAL; --Resultado: Q
```

- **ASCII(n)**: Devuelve el valor ASCII de n.

```
SELECT ASCII('O') FROM DUAL; --Resultado: 79
```

- **CONCAT(cad1, cad2)**: Devuelve las dos cadenas unidas. Es equivalente al operador ||

```
SELECT CONCAT('Hola', 'Mundo') FROM DUAL; --Resultado: HolaMundo
```

- **LOWER(cad)**: Devuelve la cadena cad con todos sus caracteres en minúsculas.

```
SELECT LOWER('En MINÚSCULAS') FROM DUAL; --Resultado: en minúsculas
```

- **UPPER(cad)**: Devuelve la cadena cad con todos sus caracteres en mayúsculas.

```
SELECT UPPER('En MAYÚSCULAS') FROM DUAL; --Resultado: EN MAYÚSCULAS
```

- **INITCAP(cad)**: Devuelve la cadena cad con su primer carácter en mayúscula.

```
SELECT INITCAP('hola') FROM DUAL; --Resultado: Hola
```

- **LPAD(cad1, n, cad2)**: Devuelve cad1 con longitud n, ajustada a la derecha, rellenando por la izquierda con cad2.

```
SELECT LPAD('M', 5, '*') FROM DUAL; --Resultado: ****M
```

- **RPAD(cad1, n, cad2)**: Devuelve cad1 con longitud n, ajustada a la izquierda, rellenando por la derecha con cad2.

```
SELECT RPAD('M', 5, '*') FROM DUAL; --Resultado: M****
```

- **REPLACE(cad, ant, nue)**: Devuelve cad en la que cada ocurrencia de la cadena ant ha sido sustituida por la cadena nue.

```
SELECT REPLACE('correo@gmail.es', 'es', 'com') FROM DUAL; --Resultado: correo@gmail.com
```

- **SUBSTR(cad, m, n)**: Devuelve la cadena cad compuesta por n caracteres a partir de la posición m.

```
SELECT SUBSTR('1234567', 3, 2) FROM DUAL; --Resultado: 34
```

- **LENGTH(cad)**: Devuelve la longitud de cad.

```
SELECT LENGTH('hola') FROM DUAL; --Resultado: 4
```

- **TRIM(cad)**: Elimina los espacios en blanco a la izquierda y la derecha de cad y los espacios dobles del interior.

```
SELECT TRIM(' Hola de nuevo ') FROM DUAL; --Resultado: Hola de nuevo
```

TEMA 4: REALIZACIÓN DE CONSULTAS

- **LTRIM(cad):** Elimina los espacios a la izquierda que posea cad.

```
SELECT LTRIM(' Hola') FROM DUAL; --Resultado: Hola
```

- **RTRIM(cad):** Elimina los espacios a la derecha que posea cad.

```
SELECT RTRIM('Hola ') FROM DUAL; --Resultado: Hola
```

- **INSTR(cad, cadBuscada [, posInicial [, nAparición]]):** Obtiene la posición en la que se encuentra la cadena buscada en la cadena inicial cad. Se puede comenzar a buscar desde una posición inicial concreta e incluso indicar el número de aparición de la cadena buscada. Si no encuentra nada devuelve cero.

```
SELECT INSTR('usuarios', 'u') FROM DUAL; --Resultado: 1
SELECT INSTR('usuarios', 'u', 2) FROM DUAL; --Resultado: 3
SELECT INSTR('usuarios', 'u', 2, 2) FROM DUAL; --Resultado: 0
```

MySQL

- **ASCII(n):** Devuelve el valor ASCII de n.

```
SELECT ASCII('0'); --Resultado: 79
```

- **CONCAT(cad1, cad2):** Devuelve las dos cadenas unidas.

```
SELECT CONCAT('Hola', 'Mundo'); --Resultado: HolaMundo
```

- **LOWER(cad):** Devuelve la cadena cad con todos sus caracteres en minúsculas.

```
SELECT LOWER('En MINÚSCULAS'); --Resultado: en minúsculas
```

- **UPPER(cad):** Devuelve la cadena cad con todos sus caracteres en mayúsculas.

```
SELECT UPPER('En MAYÚSCULAS'); --Resultado: EN MAYÚSCULAS
```

- **INSTR(cadena, subcadena):** Devuelve la posición de la primera ocurrencia de la subcadena dentro de la cadena.

```
SELECT INSTR('Hola', 'ol') -- Devuelve 2
```

- **RPAD(cad1, n, cad2):** Devuelve cad1 con longitud n, ajustada a la izquierda, rellenando por la derecha con cad2.

```
SELECT RPAD('M', 5, '*'); --Resultado: M****
```

- **REPLACE(cad, ant, nue):** Devuelve cad en la que cada ocurrencia de la cadena ant ha sido sustituida por la cadena nue.

```
SELECT REPLACE('correo@gmail.es', 'es', 'com'); --Resultado: correo@gmail.com
```

- **SUBSTR(cad, m, n):** Devuelve la cadena cad compuesta por n caracteres a partir de la posición m.

```
SELECT SUBSTR('1234567', 3, 2); --Resultado: 34
```

- **LENGTH(cad):** Devuelve la longitud de cad.

```
SELECT LENGTH('hola'); --Resultado: 4
```

TEMA 4: REALIZACIÓN DE CONSULTAS

- **TRIM(cad):** Elimina los espacios en blanco a la izquierda y la derecha de cad y los espacios dobles del interior.

```
SELECT TRIM(' Hola de nuevo '); --Resultado: Hola de nuevo
```

- **LTRIM(cad):** Elimina los espacios a la izquierda que posea cad.

```
SELECT LTRIM(' Hola'); --Resultado: Hola
```

- **RTRIM(cad):** Elimina los espacios a la derecha que posea cad.

```
SELECT RTRIM('Hola '); --Resultado: Hola
```

5.3. FUNCIONES DE MANEJO DE FECHAS

La fecha de emisión de una factura, de llegada de un avión, de ingreso en una web, podríamos seguir poniendo infinidad de ejemplos, lo que significa que es una información que se requiere en muchas situaciones y es importante guardar.

Oracle

En los SGBD se utilizan mucho las fechas. Oracle tiene dos tipos de datos para manejar fechas, son **DATE** y **TIMESTAMP**.

- **DATE** almacena fechas concretas **incluyendo a veces la hora**.
- **TIMESTAMP** almacena **un instante de tiempo más concreto** que puede incluir hasta fracciones de segundo.

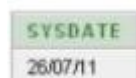
Podemos realizar operaciones numéricas con las fechas:

- Le podemos **sumar números** y esto se entiende como sumarles días, si ese número tiene decimales se suman días, horas, minutos y segundos.
- La **diferencia** entre dos fechas también nos dará un número de días.

En Oracle tenemos las siguientes funciones más comunes:

- **SYSDATE** Devuelve la fecha y hora actuales. Ejemplo: **SELECT SYSDATE FROM DUAL;** -

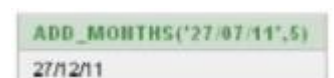
-Resultado: 26/07/11



- **SYSTIMESTAMP** Devuelve la fecha y hora actuales en formato **TIMESTAMP**. Ejemplo: **SELECT SYSTIMESTAMP FROM DUAL;** -
-Resultado: 26-JUL-11 08.32.59,609000 PM +02:00



- **ADD_MONTHS(fecha, n)** Añade a la fecha el número de meses indicado con n. Ejemplo: **SELECT ADD_MONTHS('27/07/11', 5) FROM DUAL;** --
Resultado: 27/12/11

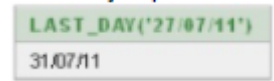


- **MONTHS_BETWEEN(fecha1, fecha2)** Devuelve el número de meses que hay entre fecha1 y fecha2. Ejemplo: **SELECT MONTHS_BETWEEN('12/07/11','12/03/11') FROM DUAL;** --Resultado: 4



TEMA 4: REALIZACIÓN DE CONSULTAS

- **LAST_DAY(fecha)** Devuelve el último día del mes al que pertenece la fecha. El valor devuelto es tipo **DATE**. Ejemplo: **SELECT LAST_DAY('27/07/11') FROM DUAL;** --Resultado: 31/07/11



- **NEXT_DAY(fecha, d)** Indica el día que corresponde si añadimos a la fecha el día d. El día devuelto puede ser texto ('Lunes', 'Martes', ..) o el número del día de la semana (1=lunes, 2=martes, ..) dependiendo de la configuración. Ejemplo: **SELECT NEXT_DAY('31/12/11','LUNES') FROM DUAL;** --Resultado: 02/01/12
- **EXTRACT(valor FROM fecha)** Extrae un valor de una fecha concreta. El valor puede ser day, month, year, hours, etc. Ejemplo: **SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;** --Resultado: 7

En Oracle: Los operadores aritméticos "+" (más) y "-" (menos) pueden emplearse para las fechas. Por ejemplo:

```
SELECT SYSDATE - 5;
```

Devolvería la fecha correspondiente a 5 días antes de la fecha actual.

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo fecha.

MySQL

En los SGBD se utilizan mucho las fechas. MySQL tiene dos tipos de datos para manejar fechas, son **DATE** y **TIMESTAMP**.

- **DATE** almacena **fechas concretas incluyendo a veces la hora**. En formato yyyy-mm-dd.
- **TIMESTAMP** almacena **un instante de tiempo más concreto** que incluye la hora, minutos y segundos, en formato yyyy-mm-dd hh:mm:ss. Utiliza el mismo formato que el tipo **DATETIME**.

Podemos realizar operaciones numéricas con las fechas:

- Le podemos **sumar números** y esto se entiende como sumarles días.
- La **diferencia** entre dos fechas también nos dará un número de días.

En MySQL tenemos las siguientes funciones más comunes:

- **CURDATE() O NOW()** Devuelve la fecha actual, y la fecha y hora actual, respectivamente.

Ejemplo: **SELECT CURDATE();** --Resultado: 2014-12-26

Ejemplo: **SELECT NOW();** -- Resultado: 2014-12-26 10:24:32

- **CURTIME()** Devuelve la hora actual.

Ejemplo: **SELECT CURTIME();** --Resultado: 10:24:32

- **CURRENT_TIMESTAMP()** Devuelve la fecha y hora actuales en formato **DATETIME**.

Ejemplo: **SELECT CURRENT_TIMESTAMP();** --Resultado: 2014-12-26 10:24:32

- **ADDDATE(fecha, n)** Añade a la fecha el número de días indicado con n.

Ejemplo: **SELECT ADDDATE('2014-11-01', 5);** --Resultado: 2014-11-06

TEMA 4: REALIZACIÓN DE CONSULTAS

- **DATEDIFF(fecha1,fecha2)** Devuelve el número de días que hay entre fecha1 y fecha2.
Ejemplo `SELECT DATEDIFF('2014-04-19','2014-04-2');` -- Devuelve 17.
- **DATE_FORMAT(fecha,formato)** Devuelve la fecha formateada según el formato especificado.

Ejemplo: `SELECT DATE_FORMAT('2014-04-19','%d/%m/%Y');` -- Devuelve 19/04/2014

Ejemplo: `SELECT DATE_FORMAT('2014-04-19','%d/%m/%y');` -- Devuelve 19/04/14

(Consultar la documentación de MySQL para ver los formatos posibles).

- **DAY(fecha)** Devuelve el día del mes de una fecha.
Ejemplo: `SELECT DAY('2014-12-19');` --Resultado: 19
- **MONTH(fecha)** Devuelve el mes de una fecha.
Ejemplo: `SELECT MONTH('2014-07-20');` --Resultado: 7
- **YEAR(fecha)** Devuelve el año de una fecha.
Ejemplo: `SELECT YEAR('2014-07-20');` --Resultado: 2014
- **DAYNAME(fecha)** Devuelve el nombre del día de una fecha.
Ejemplo: `SELECT DAYNAME('2014-12-19');` --Resultado: Friday
- **MONTHNAME(fecha)** Devuelve el nombre del mes de una fecha.
Ejemplo: `SELECT MONTHNAME('2014-12-19');` --Resultado: December

Se pueden emplear estas funciones enviando como argumento el nombre de un campo o columna de tipo fecha

Recomendación

En el siguiente enlace tienes más ejemplos de manejo de fechas con MYSQL: [Trabajar con fechas en MySQL](#)

Para saber más

En la documentación oficial de MySQL puedes consultar todas las funciones existentes para trabajar con fechas. [Funciones de fecha y hora en MySQL](#)

5.4. FUNCIONES DE CONVERSIÓN

Los SGBD tienen funciones que pueden pasar de un tipo de dato a otro.

Tanto Oracle como MySQL convierten automáticamente datos de manera que el resultado de una expresión tenga sentido. Por tanto, de manera automática se pasa de texto a número y al revés. Ocurre lo mismo para pasar de tipo texto a fecha y viceversa. Pero existen ocasiones en que queramos realizar esas conversiones de modo explícito, para ello contamos con funciones de conversión.

Oracle

- **TO_NUMBER(cad, formato)** Convierte textos en números. Se suele utilizar para dar un formato concreto a los números. Los formatos que podemos utilizar son los siguientes:

Símbolo	Significado
9	Posiciones numéricas. Si el número que se quiere visualizar contiene menos dígitos de los que se especifican en el formato, se rellena con blancos.
0	Visualiza ceros por la izquierda hasta completar la longitud del formato especificado.
\$	Antepone el signo de dólar al número.
L	Coloca en la posición donde se incluya, el símbolo de la moneda local (se puede configurar en la base de datos mediante el parámetro <code>NSL_CURRENCY</code>)
S	Aparecerá el símbolo del signo.
D	Posición del símbolo decimal, que en español es la coma.
G	Posición del separador de grupo, que en español es el punto.

- **TO_CHAR(d, formato)** Convierte un número o fecha `d` a cadena de caracteres, se utiliza normalmente para fechas ya que de número a texto se hace de forma implícita como hemos visto antes.
- **TO_DATE(cad, formato)** Convierte textos a fechas. Podemos indicar el formato con el que queremos que aparezca.

Para las funciones **TO_CHAR** y **TO_DATE**, en el caso de fechas, indicamos el formato incluyendo los siguientes símbolos:

Formatos para fechas y su significado.

Símbolo	Significado
YY	Año en formato de dos cifras
YYYY	Año en formato de cuatro cifras
MM	Mes en formato de dos cifras
MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes
DY	Día de la semana en tres letras
DAY	Día completo de la semana
DD	Día en formato de dos cifras
D	Día de la semana del 1 al 7
Q	Semestre
WW	Semana del año
AM PM	Indicador a.m. Indicador p.m.
HH12 HH24	Hora de 1 a 12 Hora de 0 a 23
MI	Minutos de 0 a 59
SS SSSS	Segundos dentro del minuto Segundos dentro desde las 0 horas

MySQL

Solo vamos a destacar las siguientes funciones:

- **CONVERT(expresion,tipo_dato)** Devuelve la expresión convertida al tipo de dato suministrado.

Ejemplo: **SELECT CONVERT(21,DECIMAL(6,2));** -- Devuelve 21.00.

- **CAST(expresion AS tipo_dato)** Análoga a la anterior.

Ejemplo: **SELECT CAST(21 AS DECIMAL(6,2));** -- Devuelve 21.00.

Recomendación

En el siguiente enlace puedes ver más ejemplos de funciones de conversión en MySQL. [Ejemplos de funciones de conversión](#)

5.5. OTRAS FUNCIONES: NVL Y DECODE – IFNULL

¿Recuerdas que era el valor **NULL**? Cualquier columna de una tabla podía contener un valor nulo independientemente al tipo de datos que tuviera definido. Eso sí, esto no era así en los casos en que definíamos esa columna como no nula (**NOT NULL**), o que fuera clave primaria (**PRIMARY KEY**).

Cualquier operación que se haga con un valor **NULL** devuelve un **NULL**. Por ejemplo, si se intenta dividir por **NULL**, no nos aparecerá ningún error sino que como resultado obtendremos un **NULL** (no se producirá ningún error tal y como puede suceder si intentáramos dividir por cero).

También es posible que el resultado de una función nos de un valor nulo.

Oracle

Por tanto, es habitual encontrarnos con estos valores y es entonces cuando aparece la necesidad de poder hacer algo con ellos. Las **funciones con nulos** nos permitirán hacer algo en caso de que aparezca un valor nulo.

- **NVL(valor, expr1)**

Si valor es **NULL**, entonces devuelve **expr1**. Ten en cuenta que **expr1** debe ser del mismo tipo que valor.

¿Y no habrá alguna función que nos permita evaluar expresiones? La respuesta es afirmativa y esa función se llama **DECODE**.

- **DECODE(expr1, cond1, valor1 [, cond2, valor2, ...], default)**

Esta función evalúa una expresión **expr1**, si se cumple la primera condición (**cond1**) devuelve el **valor1**, en caso contrario evalúa la siguiente condición y así hasta que una de las condiciones se cumpla. Si no se cumple ninguna condición se devuelve el valor por defecto que hemos llamado default.

Si en la tabla **PONENTE** queremos un listado de las especialidades, podemos pedir que cuando una especialidad no exista, aparezca el texto No tiene especialidad conocida, para ello podemos utilizar la siguiente consulta:

- **SELECT NVL(especialidad, 'No tiene especialidad conocida') FROM PONENTE;**

Obtendremos:

**Resultado de la
sentencia SELECT**

ESPECIALIDAD
No tiene dirección conocida
'Bases de Datos'

MySQL

Por tanto, es habitual encontrarnos con estos valores y es entonces cuando aparece la necesidad de poder hacer algo con ellos. Las **funciones con nulos y de control de flujo** nos permitirán hacer algo en caso de que aparezca un valor nulo.

- **IFNULL(expresión1,expresión2)**

Devuelve <expresión1> si <expresión1> es distinto de NULL, en otro caso devuelve <expresión2>.

Ejemplo: **SELECT IFNULL(NULL,'No hay dato');** --- Devuelve 'No hay dato'

Si en la tabla EMPLEADO queremos un listado de sus departamentos, podemos pedir que cuando un departamento no exista, aparezca el texto No tiene departamento, para ello podemos utilizar la siguiente consulta:

- **SELECT cdemp, IFNULL(cddep, 'No tiene departamento asignado') FROM empleado;**

Otra función interesante es la de control de flujo:

- **IF(expresión1,expresión2,expresión3)** Devuelve <expresión2> o <expresión3> en función de si <expresión1> es VERDAD o FALSO respectivamente.

Ejemplo: **SELECT IF(3>5, 'verdad', 'falso');** -- Devuelve 'falso'.

6. CONSULTAS DE RESUMEN

Seguro que alguna vez has necesitado realizar cálculos sobre un campo para obtener algún resultado global, por ejemplo, si tenemos una columna donde estamos guardando las notas que obtienen unos alumnos o alumnas en Matemáticas, podríamos estar interesados en saber cual es la nota máxima que han obtenido o la nota media.

La sentencia **SELECT** nos va a permitir **obtener resúmenes de los datos de modo vertical**. Para ello consta de una serie de cláusulas específicas (**GROUP BY, HAVING**) y tenemos también unas **funciones** llamadas **de agrupamiento o de agregado** que son las que nos dirán qué cálculos queremos realizar sobre los datos (sobre la columna).

Hasta ahora las consultas que hemos visto daban como resultado un subconjunto de filas de la tabla de la que extraíamos la información. Sin embargo, este tipo de consultas que vamos a ver no corresponde con ningún valor de la tabla sino un **total calculado** sobre los datos de la tabla. Esto hará que las consultas de resumen tengan limitaciones que iremos viendo.

Las funciones que podemos utilizar se llaman de agrupamiento (de agregado). Éstas **toman un grupo de datos** (una columna) y **producen un único dato que resume el grupo**. Por ejemplo, la función **SUM()** acepta una columna de datos numéricos y devuelve la suma de estos.

El simple hecho de utilizar una función de agregado en una consulta la convierte en consulta de resumen.

Todas las funciones de agregado tienen una estructura muy parecida: **FUNCIÓN ([ALL | DISTINCT] Expresión)** y debemos tener en cuenta que:

- La palabra **ALL** indica que se tienen que tomar todos los valores de la columna. Es el valor por defecto.
- La palabra **DISTINCT** indica que se considerarán todas las repeticiones del mismo valor como uno solo (considera valores distintos).
- El grupo de valores sobre el que actúa la función lo determina el resultado de la expresión que será el nombre de una columna o una expresión basada en una o varias columnas. Por tanto, en la expresión nunca puede aparecer ni una función de agregado ni una subconsulta.
- Todas las funciones se aplican a las filas del origen de datos una vez ejecutada la cláusula **WHERE** (si la tuviéramos).
- Todas las funciones (excepto **COUNT**) ignoran los valores **NULL**.
- Podemos encontrar una función de agrupamiento dentro de una lista de selección en cualquier sitio donde pudiera aparecer el nombre de una columna. Es por eso que puede formar parte de una expresión pero no se pueden anidar funciones de este tipo.
- No se pueden mezclar funciones de columna con nombres de columna ordinarios, aunque hay excepciones que veremos más adelante.

Para saber más

Puedes acceder a este enlace si quieres conocer más sobre este tipo de consultas. [Consultas de resumen.](#)

6.1. FUNCIONES DE AGREGADO: SUM Y COUNT.

Sumar y contar filas o datos contenidos en los campos es algo bastante común. Imagina que para nuestra tabla Usuarios necesitamos sumar el número de créditos total que tienen nuestros jugadores. Con una función que sumara los valores de la columna crédito sería suficiente, siempre y cuando lo agrupáramos por cliente, ya que de lo contrario lo que obtendríamos sería el total de todos los clientes jugadores.

- La función **SUM**:
 - **SUM([ALL|DISTINCT] expresión)**
 - Devuelve la suma de los valores de la expresión.
 - Sólo puede utilizarse con columnas cuyo tipo de dato sea número. El resultado será del mismo tipo aunque puede tener una precisión mayor.
 - Por ejemplo,

```
SELECT SUM(precio) FROM CONFERENCIA; en Oracle
SELECT SUM( salario) FROM empleado; en MySQL
```

- La función **COUNT**:
 - **COUNT([ALL|DISTINCT] expresión)**
 - Cuenta los elementos de un campo. **Expresión** contiene el nombre del campo que deseamos contar. Los operandos de expresión pueden incluir el nombre del campo, una constante o una función.
 - Puede contar cualquier tipo de datos incluido texto.
 - **COUNT** simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan.
 - La función **COUNT** no cuenta los registros que tienen campos **NULL** a menos que **expresión** sea el carácter comodín **asterisco (*)**.
 - Si utilizamos **COUNT(*)**, calcularemos el total de filas, incluyendo aquellas que contienen valores **NULL**.
 - Por ejemplo,

```
SELECT COUNT(nombre) FROM PONENTE; en Oracle
SELECT COUNT(*) FROM PONENTE; en Oracle
SELECT COUNT(cddep) FROM empleado; en MySQL //Nos permite saber
cuantos empleados tienen departamento asignado.
SELECT COUNT(*) FROM empleado; en MySQL //Nos permite saber
cuantos empleados hay en total
```



Ejercicio resuelto

Oracle

Utilizando las tablas y datos descargados anteriormente, vamos a realizar una consulta donde obtengamos el número total de salas cuya capacidad sea igual o superior a 200.

Mostrar retroalimentación

```
SELECT COUNT(*) AS NUM_SALAS FROM SALA WHERE capacidad >= 200;
```

MySQL

Utilizando las tablas y datos de la empresa **Proyectosx** descargados anteriormente, vamos a realizar una consulta donde contemos el número de empleados con departamento asignado. Pero esta vez lo realizamos de otra forma a la que se ha visto anteriormente.

Mostrar retroalimentación

```
SELECT COUNT(cdept) FROM empleado WHERE cdept IS NOT NULL;
```

6.2. FUNCIONES DE AGREGADO: MIN Y MAX.

¿Y si pudiéramos encontrar el valor máximo y mínimo de una lista enormemente grande? Esto es lo que nos permiten hacer las siguientes funciones.

- **Función MIN:**

- **MIN ([ALL| DISTINCT] expresión)**
- Devuelve el valor mínimo de la expresión sin considerar los nulos (**NULL**).
- En expresión podemos incluir el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).
- Un ejemplo sería:

```
SELECT MIN(precio) FROM CONFERENCIA; // en Oracle
SELECT MIN(salario) FROM empleado; //en MySQL: -- Obtiene el
menor salario
SELECT MIN(fecha_ingreso) FROM empleado; // MySQL: -- Obtiene la
fecha más antigua de ingreso
```

- **Función MAX:**

- **MAX ([ALL| DISTINCT] expresión)**
- Devuelve el valor máximo de la expresión sin considerar los nulos (**NULL**).
- En expresión podemos incluir el nombre de un campo de una tabla, una constante o una función (pero no otras funciones agregadas de SQL).

```
SELECT MAX(precio) FROM CONFERENCIA; //Oracle
SELECT MAX (salario) FROM empleado; // MySQL -- obtiene el mayor
salario
SELECT MAX(fecha_ingreso) FROM empleado; // MySQL -- Obtiene la
fecha más actual o reciente de ingreso
```

6.3. FUNCIONES DE AGREGADO: AVG, VAR, STDEV Y STDEVP.

Quizás queramos obtener datos estadísticos de los datos guardados en nuestra base de datos. Para ello podemos hacer uso de las funciones que calculan el promedio, la varianza y la desviación típica.

- **Función AVG ([ALL| DISTINCT] expresión)**

- Devuelve el promedio de los valores de un grupo, para ello se omiten los valores nulos (**NULL**).
- El grupo de valores será el que se obtenga como resultado de la expresión y ésta puede ser un nombre de columna o una expresión basada en una columna o varias de la tabla.
- Se aplica a campos tipo número y el tipo de dato del resultado puede variar según las necesidades del sistema para representar el valor.

Ejemplos en MySQL:

SELECT AVG(salario) as 'Salario medio' FROM empleado;

SELECT round(AVG(salario),2) as 'Salario medio' FROM empleado; -- para sacar sólo dos decimales

Oracle

- **Función VAR ([ALL| DISTINCT] expresión)**

- Devuelve la varianza estadística de todos los valores de la expresión.
- Como tipo de dato admite únicamente columnas numéricas. Los valores nulos (**NULL**) se omiten.

- **Función STDEV ([ALL| DISTINCT] expresión)**

- Devuelve la desviación típica estadística de todos los valores de la expresión.
- Como tipo de dato admite únicamente columnas numéricas. Los valores nulos (**NULL**) se omiten.



Ejercicio resuelto

Oracle

Utilizando las tablas y datos descargados anteriormente, vamos a realizar una consulta donde obtengamos la media de las gratificaciones de los ponentes

Mostrar retroalimentación

```
SELECT AVG(gratificacion) AS MEDIA_GRATIFICACIONES FROM PARTICIPAR;
```

MySQL

Utilizando las tablas y datos de la empresa **Proyectosx** descargados anteriormente, vamos a realizar una consulta donde obtengamos la media de horas trabajadas en proyectos de la tabla trabaja.

Mostrar retroalimentación

```
SELECT AVG(nhoras) FROM trabaja;
```


7. AGRUPAMIENTO DE REGISTROS

Hasta aquí las consultas de resumen que hemos visto obtienen totales de todas las filas de un campo o una expresión calculada sobre uno o varios campos. Lo que hemos obtenido ha sido una única fila con un único dato.

Ya verás como en muchas ocasiones en las que utilizamos consultas de resumen nos va a interesar calcular **totales parciales**, es decir, **agrupados según un determinado campo**.

De este modo podríamos obtener de una tabla EMPLEADOS, en la que se guarda su sueldo y su actividad dentro de la empresa, el valor medio del sueldo en función de la actividad realizada en la empresa. También podríamos tener una tabla clientes y obtener el número de veces que ha realizado un pedido, etc.

En todos estos casos en lugar de una única fila de resultados necesitaremos una fila por cada actividad, cada cliente, etc.

Podemos obtener estos **subtotales** utilizando la cláusula **GROUP BY**. La sintaxis es la siguiente:

```
SELECT columna1, columna2, ...  
FROM tabla1, tabla2, ...  
WHERE condición1, condición2, ...  
GROUP BY columna1, columna2, ...  
HAVING condición  
ORDER BY ordenación;
```

En la cláusula **GROUP BY** se colocan las columnas por las que vamos a agrupar. En la cláusula **HAVING** se especifica la condición que han de cumplir los grupos para que se realice la consulta.

Es muy importante que te fijas bien en el orden en el que se ejecutan las cláusulas:

1. **WHERE** que filtra las filas según las condiciones que pongamos.
2. **GROUP BY** que crea una tabla de grupos nueva.
3. **HAVING** filtra los grupos.
4. **ORDER BY** que ordena o clasifica la salida.

Las columnas que aparecen en el **SELECT** y que no aparezcan en la cláusula **GROUP BY** deben tener una función de agrupamiento. Si esto no se hace así producirá un error. Otra opción es poner en la cláusula **GROUP BY** las mismas columnas que aparecen en **SELECT**.

MySQL

Obtenemos el total de empleados por departamento. Incluso aparecerá el total de empleados que no están asignados a un departamento, esto es, su departamento es NULL.

```
SELECT cddep, count(cddep) FROM empleado GROUP BY cddep;
```

Si estuviéramos interesados en el total de empleados de cada departamento que ganan un salario superior a 1500€, pero que sólo se mostraran los departamentos con más de 2 empleados con salario superior a 1500€:

```
SELECT cddep, COUNT(cddep) FROM empleado WHERE salario > 1500 GROUP BY cddep  
HAVING COUNT(cddep) >2;
```

Oracle

Primero vamos a obtener el número total de asistentes por empresa (sin contar con aquellos de los que no se conoce su empresa)

```
SELECT empresa, COUNT(codigo) AS NUM_ASISTENTES FROM ASISTENTE WHERE empresa  
IS NOT NULL GROUP BY empresa;
```

Y si estuviéramos interesados en el número total de asistentes pero sólo de las empresas BigSoft y ProgConsulting sería:

```
SELECT empresa, COUNT(codigo) AS NUM_ASISTENTES FROM ASISTENTE WHERE empresa  
IS NOT NULL GROUP BY empresa HAVING empresa = 'BigSoft' OR empresa=  
'ProgConsulting';
```

8. CONSULTAS MULTITABLAS

Recuerda que una de las propiedades de las bases de datos relacionales era que distribuíamos la información en varias tablas que a su vez estaban relacionadas por algún campo común. Así evitábamos repetir datos. Por tanto, también será frecuente que tengamos que consultar datos que se encuentren distribuidos por distintas tablas.

MySQL

Si disponemos de una tabla EMPLEADO cuya clave principal es cdemp y esta tabla a su vez está relacionada con la tabla DEPARTAMENTO a través del campo **cddep**. Si quisiéramos obtener el nombre de los empleados y el nombre y ciudad de sus departamentos, necesitaríamos coger datos de ambas tablas pues el nombre y ciudad de cada departamento se guarda en la tabla DEPARTAMENTO. Esto significa que cogeremos filas de una y de otra.

Imagina también que en lugar de tener una tabla EMPLEDO, dispusiéramos de dos por tenerlas en servidores distintos. Lo lógico es que en algún momento tendríamos que unir las.

TEMA 4: REALIZACIÓN DE CONSULTAS

Hasta ahora las consultas que hemos usado se referían a una sola tabla, pero también es posible hacer consultas usando varias tablas en la misma sentencia **SELECT**. Esto permitirá realizar distintas operaciones como son:

- La composición interna.
- La composición externa.

En la versión SQL de 1999 se especifica una nueva sintaxis para consultar varias tablas que MySQL incorpora, así que también la veremos. La razón de esta nueva sintaxis era separar las condiciones de asociación respecto a las condiciones de selección de registros.

La sintaxis es la siguiente:

```
SELECT tabla1.columna1, tabla1.columna2, ..., tabla2.columna1, tabla2.columna2,
...
FROM tabla1
[JOIN tabla2 USING (columna) |
[JOIN tabla2 ON (tabla1.columna=tabla2.columna)] |
[LEFT | RIGHT | FULL OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)]
```

Oracle

Si disponemos de la tabla CONFERENCIA cuya clave principal es referencia y esta tabla a su vez está relacionada con la tabla SALA a través del campo **sala**. Si quisiéramos obtener la capacidad de las salas en la que se realiza cada conferencia necesitaríamos coger datos de ambas tablas pues la capacidad de las salas se guarda en la tabla SALA. Esto significa que cogeremos filas de una y de otra.

Imagina también que en lugar de tener una tabla PONENTE, dispusiéramos de dos por tenerlas en servidores distintos. Lo lógico es que en algún momento tendríamos que unirlos.

Hasta ahora las consultas que hemos usado se referían a una sola tabla, pero también es posible hacer consultas usando varias tablas en la misma sentencia **SELECT**. Esto permitirá realizar distintas operaciones como son:

- La composición interna.
- La composición externa.

En la versión SQL de 1999 se especifica una nueva sintaxis para consultar varias tablas que Oracle incorpora, así que también la veremos. La razón de esta nueva sintaxis era separar las condiciones de asociación respecto a las condiciones de selección de registros.

La sintaxis es la siguiente:

```
SELECT tabla1.columna1, tabla1.columna2, ..., tabla2.columna1, tabla2.columna2,
...
FROM tabla1
[CROSS JOIN tabla2] |
[NATURAL JOIN tabla2] |
[JOIN tabla2 USING (columna) |
[JOIN tabla2 ON (tabla1.columna=tabla2.columna)] |
[LEFT | RIGHT | FULL OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)]
```

Recomendación

En el siguiente enlace puedes ver ejemplos de los diferentes tipos de JOIN. [Tipos de JOIN](#)

8.1. COMPOSICIONES INTERNAS

¿Qué ocurre si combinamos dos o más tablas sin ninguna restricción? El resultado será un producto cartesiano.

El producto cartesiano entre dos tablas da como resultado todas las combinaciones de todas las filas de esas dos tablas.

Se indica poniendo en la cláusula **FROM** las tablas que queremos componer separadas por comas. Y puedes obtener el producto cartesiano de las tablas que quieras.

Como lo que se obtiene son todas las posibles combinaciones de filas, debes tener especial cuidado con las tablas que combinas. Si tienes dos tablas de 10 filas cada una, el resultado tendrá 10x10 filas, a medida que aumentemos el número de filas que contienen las tablas, mayor será el resultado final, con lo cual se puede considerar que nos encontraremos con una operación costosa.

Esta operación no es de las más utilizadas ya que coge una fila de una tabla y la asocia con todos y cada uno de las filas de la otra tabla, independientemente de que tengan relación o no. Lo más normal es que queramos seleccionar los registros según algún criterio.

Necesitaremos **discriminar** de alguna forma para que únicamente aparezcan filas de una tabla que estén relacionadas con la otra tabla. A esto se le llama **asociar tablas (JOIN)**.

Para hacer una composición interna se parte de un producto cartesiano y se eliminan aquellas filas que no cumplen la condición de composición.

Lo importante en las composiciones internas es emparejar los campos que han de tener valores iguales.

Las reglas para las composiciones son:

- Pueden combinarse tantas tablas como se desee.
- El criterio de combinación puede estar formado por más de una pareja de columnas.
- En la cláusula **SELECT** pueden citarse columnas de ambas tablas, condicionen o no, la combinación.
- Si hay columnas con el mismo nombre en las distintas tablas, deben identificarse especificando la tabla de procedencia o utilizando un alias de tabla.
- Las columnas que aparecen en la cláusula **WHERE** se denominan **columnas de emparejamiento** ya que son las que permiten emparejar las filas de las dos tablas. Éstas no tienen por qué estar incluidas en la lista de selección. Emparejaremos tablas que estén relacionadas entre sí y además, una de las columnas de emparejamiento será clave principal en su tabla. Cuando emparejamos campos debemos especificar de la siguiente forma: **NombreTabla1. Camporelacionado1 = NombreTabla2.Camporelacionado2.**

Puedes combinar una tabla consigo misma pero debes poner de manera obligatoria un alias a uno de los nombres de la tabla que vas a repetir.

TEMA 4: REALIZACIÓN DE CONSULTAS

MySQL

Veamos un ejemplo, si queremos obtener el nombre y ciudad de los departamentos de los empleados, así como el código, nombre y fecha de ingreso, tendremos que utilizar alias o anteponer el nombre de la tabla a las columnas que se denominen igual en ambas tablas que se cambian:

```
SELECT e.cdemp, e.nombre, e.fecha_ingreso, d.nombre, d.ciudad
FROM empleado e, departamento d
WHERE e.cddep= d.cddep;
```

Observa que ambas tablas, empleado y departamento están relacionadas por la columna cddep (código de departamento) y que en ambas tablas la columna se denomina igual, y lo mismo ocurre con el nombre del empleado y departamento, en ambas tablas se guarda en una columna denominada nombre. Por ello, es obligatorio anteponer al nombre de cada columna que se denomine igual, el nombre de la tabla de procedencia (**tabla.columna**) o bien, poner un alias a la tabla y utilizar el alias (**alias.columna**).

En el ejemplo, se ha optado por poner a empleado el alias **e**, y a departamento el alias **d**, de manera que todas las columnas quedan totalmente claras si le anteponemos ese alias.

Puedes **combinar una tabla consigo misma** pero debes poner de manera obligatoria un alias a uno de los nombres de la tabla que vas a repetir.

Por ejemplo, la tabla empleado tiene una (inter)relación reflexiva, pues un empleado tiene como jefe a otro empleado. De hecho, ésto hace que aparezca dentro de la misma tabla la clave foránea **cdjefe** que hace referencia a la clave primaria **cdemp**.

Supongamos que nuestra consulta es para obtener el código y nombre de cada empleado con el nombre de su jefe ¿cómo la redactaríamos?. Observa que habrá que componer o combinar la tabla empleado con ella misma, desde el punto de vista de los empleados y el de los jefes. La consulta quedaría así:

```
SELECT e.cdemp, e.nombre, je.nombre
FROM empleado e, empleado je
WHERE e.cdjefe = je.cdemp;
```

Oracle

Veamos un ejemplo, si queremos obtener la referencia, el tema, el nombre de la sala en la que se celebra y la capacidad tendremos:

```
SELECT referencia, tema, nombre, capacidad FROM CONFERENCIA, SALA
WHERE CONFERENCIA.sala= SALA.Nombre;
```

Ahora vamos a obtener el código y el nombre de los asistentes (sin repetirse) que asisten a alguna conferencia que tenga la palabra "datos" en su tema.

```
SELECT DISTINCT A.codigo, A.nombre FROM ASISTENTE A, CONFERENCIA C, ASISTIR
AT, WHERE A.codigo= AT.codasistente AND C.referencia=AT.refconferencia AND
C.Tema LIKE '%DATOS%';
```

TEMA 4: REALIZACIÓN DE CONSULTAS



Ejercicio resuelto

Oracle

Utilizando las tablas y datos descargados anteriormente, vamos a realizar una consulta donde obtengamos el nombre de los ponentes junto a la referencia y el tema de las conferencias en las que han participado

Mostrar retroalimentación

```
SELECT P.nombre, P.apellido1, P.apellido2, C.referencia, C.tema
FROM PONENTE P, PARTICIPAR PR, CONFERENCIA C
WHERE P.codigo=PR.codponente AND PR.refconferencia=C.referencia;
```

Obtener un listado con el nombre y apellidos de los asistentes que hayan asistido a la conferencia con referencia 'PWB1314'

Mostrar retroalimentación

```
SELECT A.nombre, A.apellido1, A.apellido2 FROM ASISTENTE A, ASISTIR AT, CONFERENCIA C WHERE C.referencia=AT.refconferencia AND A.codigo=AT.codasistente AND C.referencia='PWB1314';
```

MySQL

Utilizando las tablas y datos de la empresa Proyectosx descargados anteriormente, vamos a realizar una consulta donde obtengamos el código y nombre de los empleados junto a su salario y código de los proyectos en los que trabajan.

Mostrar retroalimentación

```
SELECT empleado.cdemp, nombre, salario, cdpro
FROM empleado, trabaja
WHERE empleado.cdemp=trabaja.cdemp;(o poniendo alias)
SELECT e.cdemp, e.nombre, e.salario, t.cdpro
FROM empleado e, trabaja t
WHERE e.cdemp=t.cdemp;
```

Obtener un listado como el anterior, pero sólo para los empleados del departamento de código '03'.

Mostrar retroalimentación

```
SELECT e.cdemp, e.nombre, e.salario, t.cdpro
FROM empleado e, trabaja t
WHERE e.cdemp=t.cdemp AND
e.cddep='03';
```

8.2. COMPOSICIONES EXTERNAS

¿Has pensado que puede que te interese seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla? Esto puede ser necesario.

MySQL

Imagina que tenemos, tal como ocurre en nuestra base de datos **proyectosx** guardadas en dos tablas la información de los empleados de la empresa (**cdemp**, **nombre**, **salario** y **cddep**) por otro lado los departamentos (**cddep**, **nombre**) de esa empresa. Recientemente se ha remodelado la empresa y se han creado un par de departamentos más pero no se les ha asignado los empleados. Si tuviéramos que obtener un

TEMA 4: REALIZACIÓN DE CONSULTAS

informe con los datos de los empleados por departamento, seguro que deben aparecer esos departamentos aunque no tengan empleados. Para poder hacer esta combinación usaremos las composiciones externas.

¿Cómo es el formato? En este caso lo mejor es utilizar directamente la sintaxis derivada del estándar SQL 99, mediante una composición externa **LEFT OUTER JOIN**.

```
SELECT    tabla1.columna1,    tabla1.columna2,    ...,    tabla2.columna1,
          tabla2.columna2, ...
FROM tabla1
[LEFT | RIGTH  OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)]
```

En nuestro ejemplo, como queremos que aparezcan todos los departamentos, estén o no relacionados con empleados, pondremos esa tabla, la tabla DEPARTAMENTO, a la izquierda de **LEFT OUTER JOIN**.

Obtendríamos el mismo resultado, si pusiéramos la tabla DEPARTAMENTO a la derecha de **RIGHT OUTER JOIN**.

Oracle

Imagina que tenemos en una base de datos guardadas en dos tablas la información de los empleados de la empresa (**Cod_empleado, Nombre, Apellidos, salario y Cod_dpto**) por otro lado los departamentos (**Codigo_dep, Nombre**) de esa empresa. Recientemente se ha remodelado la empresa y se han creado un par de departamentos más pero no se les ha asignado los empleados. Si tuviéramos que obtener un informe con los datos de los empleados por departamento, seguro que deben aparecer esos departamentos aunque no tengan empleados. Para poder hacer esta combinación usaremos las composiciones externas.

¿Cómo es el formato? Muy sencillo, añadiremos un signo más entre paréntesis (+) en la igualdad entre campos que ponemos en la cláusula **WHERE**. El carácter (+) irá detrás del nombre de la tabla en la que deseamos aceptar valores nulos.

En nuestro ejemplo, la igualdad que tenemos en la cláusula **WHERE** es **Cod_dpto (+)= Codigo_dep** ya que es en la tabla empleados donde aparecerán valores nulos.

8.3. COMPOSICIONES DE LA VERSION SQL99

Como has visto, SQL incluye en esta versión mejoras de la sintaxis a la hora de crear composiciones en consultas. Recuerda que la sintaxis es la siguiente:

```
SELECT tabla1.columna1, tabla1.columna2, ..., tabla2.columna1, tabla2.columna2, ...
FROM tabla1
    [CROSS JOIN tabla2] |
    [NATURAL JOIN tabla2] |
    [JOIN tabla2 USING (columna)] |
    [JOIN tabla2 ON (tabla1.columna=tabla2.columna)] |
    [LEFT | RIGTH | FULL OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)];
```


TEMA 4: REALIZACIÓN DE CONSULTAS

CROSS JOIN: creará un producto cartesiano de las filas de ambas tablas por lo que podemos olvidarnos de la cláusula **WHERE**.

NATURAL JOIN: detecta automáticamente las claves de unión, basándose en el nombre de la columna que coincide en ambas tablas. Por supuesto, se requerirá que las columnas de unión tengan el mismo nombre en cada tabla. Además, esta característica funcionará incluso si no están definidas las claves primarias o ajenas.

JOIN USING: las tablas pueden tener más de un campo para relacionar y no siempre queremos que se relacionen por todos los campos. Esta cláusula permite establecer relaciones indicando qué campo o campos comunes se quieren utilizar para ello.

JOIN ON: se utiliza para unir tablas en la que los nombres de columna no coinciden en ambas tablas o se necesita establecer asociaciones más complicadas.

OUTER JOIN: se puede eliminar el uso del signo (+) para composiciones externas utilizando un **OUTER JOIN**, de este modo resultará más fácil de entender. (**Oracle**)

LEFT OUTER JOIN: es una composición externa izquierda, todas las filas de la tabla de la izquierda se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

RIGTH OUTER JOIN: es una composición externa derecha, todas las filas de la tabla de la derecha se devuelven, aunque no haya ninguna columna correspondiente en las tablas combinadas.

FULL OUTER JOIN: es una composición externa en la que se devolverán todas las filas de los campos no relacionados de ambas tablas. (**Oracle**)

MySQL

Podríamos transformar algunas de las consultas con las que hemos estado trabajando:

Veamos un ejemplo, si queremos obtener el nombre y ciudad de los departamentos de los empleados, así como el código, nombre y fecha de ingreso, tendremos que utilizar alias o anteponer el nombre de la tabla a las columnas que se denominen igual en ambas tablas que se cambian:

```
SELECT e.cdemp, e.nombre, e.fecha_ingreso, d.nombre, d.ciudad
FROM empleado e INNER JOIN departamento d ON e.cddep= d.cddep;
```

Queríamos también, una consulta para obtener el código y nombre de cada empleado con el nombre de su jefe ¿cómo la redactaríamos ahora?. Observa que habrá que componer o combinar la tabla empleado con ella misma, desde el punto de vista de los empleados y el de los jefes. La consulta quedaría así:

```
SELECT e.cdemp, e.nombre, je.nombre
FROM empleado e INNER JOIN empleado je ON e.cdjefe = je.cdemp;
```

Oracle

Podríamos transformar algunas de las consultas con las que hemos estado trabajando:

Queríamos obtener de cada conferencia la referencia, el tema, el nombre de la sala en la que se celebra y la capacidad. Es una consulta de composición interna, luego utilizaremos **JOIN ON**:

```
SELECT referencia, tema, nombre, capacidad FROM CONFERENCIA JOIN SALA ON  
CONFERENCIA.sala= SALA.nombre;
```

Queríamos también, obtener un listado con los nombres de los distintos departamentos y sus jefes con sus datos personales. Ten en cuenta que deben aparecer todos los departamentos, aunque no tengan asignado ningún jefe. Aquí estamos ante una composición externa, luego podemos utilizar **OUTER JOIN**:

```
SELECT D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1, E.APELLIDO2  
FROM DEPARTAMENTOS D LEFT OUTER JOIN EMPLEADOS E ON ( D.JEFE = E.DNI);
```

Como recomendación, utilizaremos siempre la sintaxis de SQL99, **utilizando JOIN tanto para composiciones internas como para composiciones externas.**

Para saber más

En MySQL también se utilizan las composiciones, aquí puedes verlo: [Composiciones.](#)

9. OTRAS CONSULTAS MULTITABLAS: UNIÓN, INTERSECCIÓN Y DIFERENCIA DE CONSULTAS.

Seguro que cuando empieces a trabajar con bases de datos llegará un momento en que dispongas de varias tablas con los mismos datos guardados para distintos registros y quieras unirlos en una única tabla. ¿Esto se puede hacer? Es una operación muy común junto a otras. Al fin y al cabo, una consulta da como resultado un conjunto de filas y con conjuntos podemos hacer entre otras, tres tipos de operaciones comunes como son: unión, intersección y diferencia.

UNION: combina las filas de un primer **SELECT** con las filas de otro **SELECT**, desapareciendo las filas duplicadas.

INTERSECT: examina las filas de dos **SELECT** y devolverá aquellas que aparezcan en ambos conjuntos. Las filas duplicadas se eliminarán.

MINUS: devuelve aquellas filas que están en el primer **SELECT** pero no en el segundo. Las filas duplicadas del primer **SELECT** se reducirán a una antes de comenzar la comparación.

Para estas tres operaciones es muy **importante** que utilices en los dos **SELECT** el **mismo número y tipo de columnas** y en el **mismo orden**.

Estas operaciones se pueden combinar anidadas, pero es conveniente utilizar paréntesis para indicar que operación quieres que se haga primero.

Veamos un ejemplo de cada una de ellas.

TEMA 4: REALIZACIÓN DE CONSULTAS

UNIÓN: Obtener los nombres y ciudades de todos los proveedores y clientes de Alemania.

```
SELECT NombreCia, Ciudad FROM PROVEEDORES WHERE Pais = 'Alemania'
UNION
SELECT NombreCia, Ciudad FROM CLIENTES WHERE Pais = 'Alemania';
```

INTERSECCIÓN: Una academia de idiomas da clases de inglés, frances y portugueses; almacena los datos de los alumnos en tres tablas distintas unas llamadas "ingles", en una tabla denominada "frances" y los que aprenden portugueses en la tabla "portugues". La academia necesita el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles información sobre los exámenes.

```
SELECT nombre, domicilio FROM ingles INTERSECT
SELECT nombre, domicilio FROM frances INTERSECT
SELECT nombre, domicilio FROM portugues;
```

DIFERENCIA: Ahora la academia necesita el nombre y domicilio solo de todos los alumnos que cursan inglés (no quiere a los que ya cursan portugués pues va a enviar publicidad referente al curso de portugués).

```
SELECT nombre, domicilio FROM INGLES
MINUS
SELECT nombre,domicilio FROM PORTUGUES;
```

10. SUBCONSULTAS

A veces tendrás que utilizar en una consulta los resultados de otra que llamaremos **subconsulta**. La sintaxis es:

```
SELECT listaExpr
FROM tabla
WHERE expresión OPERADOR
      ( SELECT listaExpr
        FROM tabla);
```

La subconsulta puede ir dentro de las cláusulas **WHERE**, **HAVING** o **FROM**.

El **OPERADOR** puede ser **>**, **<**, **>=**, **<=**, **!=**, **=** o **IN**. Las subconsultas que se utilizan con estos operadores **devuelven un único valor**, si la subconsulta devolviera más de un valor devolvería un error.

Como puedes ver en la sintaxis, las subconsultas **deben ir entre paréntesis y a la derecha** del operador.

MySQL

```
SELECT nombre, salario
FROM empleado
WHERE salario <
      (SELECT salario FROM empleado
       WHERE nombre = 'Esperanza Amarillo');
```

Obtendríamos el nombre de los empleados y el sueldo de aquellos que cobran menos que Esperanza Amarillo.

TEMA 4: REALIZACIÓN DE CONSULTAS

Oracle

```
SELECT nombre, apellido1, apellido2 FROM
ASISTENTE WHERE fechanac >= (SELECT fechanac
FROM ASISTENTES WHERE nombre='Felipe' AND apellido1='Martín' AND
apellido2='Comillas');
```

Obtendríamos el nombre de los asistentes mayores que 'Felipe Martín Comillas'

Los tipos de datos que devuelve la subconsulta y la columna con la que se compara ha de ser el mismo.

¿Qué hacemos si queremos comparar un valor con varios, es decir, si queremos que la subconsulta devuelva más de un valor y comparar el campo que tenemos con dichos valores? Imagina que queremos ver si el sueldo de un empleado que es administrativo es mayor o igual que el sueldo medio de otros puestos en la empresa. Para saberlo deberíamos calcular el sueldo medio de las demás ocupaciones que tiene la empresa y éstos compararlos con la de nuestro empleado. Como ves, el resultado de la subconsulta es más de una fila. ¿Qué hacemos?

Cuando el resultado de la subconsulta es más de una fila, SQL utiliza **instrucciones especiales** entre el operador y la consulta. Estas instrucciones son:

- **ANY.** Compara con cualquier fila de la consulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta.
- **ALL.** Compara con todas las filas de la consulta. La instrucción resultará cierta si es cierta toda la comparación con los registros de la subconsulta.
- **IN.** No utiliza comparador, lo que hace es comprobar si el valor se encuentra en el resultado de la subconsulta.
- **NOT IN.** Comprueba si un valor no se encuentra en una subconsulta.

MySQL

En la siguiente consulta obtenemos el empleado que menos cobra:

```
SELECT nombre, salario
FROM empleado
WHERE salario <= ALL (SELECT salario FROM empleado);
```

Oracle

En la siguiente consulta obtenemos el nombre del asistente más joven:

```
SELECT nombre
FROM ASISTENTE
WHERE fechanac <= ALL (SELECT fechanac FROM ASISTENTE);
```

Para saber más

¿Quieres más ejemplos con los que practicar? [Ejercicios SQL.](#)

11. REALIZANDO UN REPASO MEDIANTE UN EJEMPLO PRÁCTICO



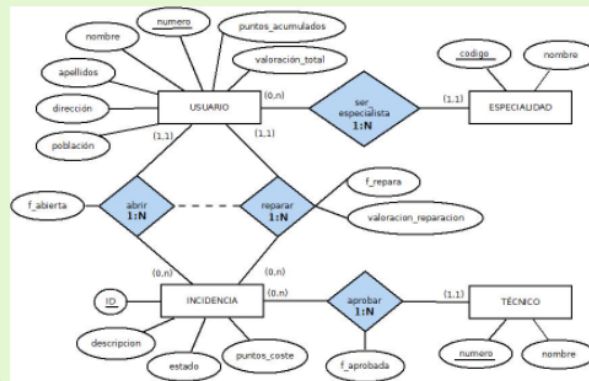
Caso práctico

Ana y Juan ya han visto todo lo que se puede hacer con el lenguaje SQL pero Ada les ha pasado un ejemplo más con otra Base de Datos en la que se les explica mediante algunos videotutoriales algunos conceptos que ya conocen y les sirve para repasar.

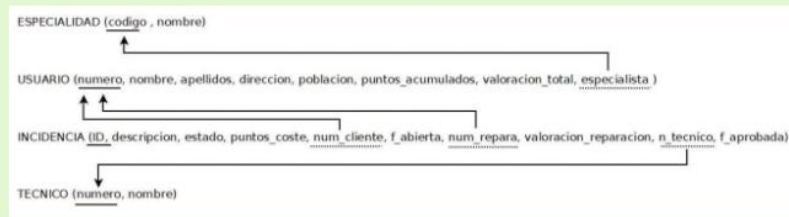
La BD TRUEKASA trata sobre intercambios o trueques que realizan los diferentes usuarios de la aplicación con respecto a incidencias que se producen en las viviendas (fontanería, albañilería, pintura, etc.). Dichas incidencias las abren y reparan los propios usuarios y hay unos técnicos que las aprueban. Se adjunta a continuación el script que puedes descargar para practicar:

[Script de la BD Truekasa](#) (zip - 1.84 KB).

El diagrama Entidad-Relación es el siguiente:



Y el modelo relacional resultante sería:



Para comenzar el repaso se puede visualizar en el siguiente videotutorial en qué consiste la BD TRUEKASA y repasar los conceptos básicos aprendidos en esta unidad utilizando SQL para realizar alguna consulta utilizando diferentes cláusulas. <https://www.youtube.com/watch?v=fS2cuJjXleQ>

A continuación, se puede ver cómo realizar varias consultas sencillas utilizando diferentes partes de la sentencia SELECT. <https://www.youtube.com/watch?v=JlCdCuKQG8M>

En el siguiente videotutorial se explica la diferencia de sintaxis entre el uso del WHERE y el JOIN para unir tablas y se realizan algunas consultas utilizando varias tablas. También se realiza algún ejemplo de consulta que utiliza subconsultas. Es muy necesario e imprescindible saber utilizar bien las consultas multitablas uniéndolas correctamente y siempre utilizando las tablas estrictamente necesarias así como también utilizar en alguna ocasión alguna subconsulta dentro de la consulta principal. <https://www.youtube.com/watch?v=JJkeETZkRRI>

Seguimos repasando viendo en el siguiente videotutorial una consulta más compleja utilizando las cláusulas de agrupamiento GROUP BY y la cláusula HAVING para filtrar o poner condiciones a esos grupos. https://www.youtube.com/watch?v=s_3VxgcDQpI

TEMA 4: REALIZACIÓN DE CONSULTAS

Para continuar veremos en este videotutorial cómo utilizar funciones en una sentencia de consultas. Éstas son muy útiles pues nos devuelven y facilitan muchas operaciones que se deben realizar con los datos de una BD. Dependiendo del SGBD utilizado siempre es aconsejable consultar las referencias oficiales para su correcta utilización. En este caso al estar utilizando MySQL se consultan la referencia apropiada para utilización de los distintos tipos de funciones (numéricas, fechas, etc.) <https://www.youtube.com/watch?v=0PPiCoeMZ5o>

Y por último en este videotutorial se ve un ejemplo claro de la utilización de las dos relaciones que hay entre las mismas dos tablas. Si nos fijamos, los usuarios pueden tener dos roles diferentes: como clientes cuando abren incidencias y como reparadores cuando arreglan una incidencia. Por tanto, para poder utilizar y realizar consultas multitablas enlazándolas por las claves adecuadas es importante conocer qué información debemos tratar en cada caso, tanto en la cláusula SELECT como en las posibles condiciones en el WHERE y por supuesto enlazar tantas veces como sean necesarias las tablas que utilizamos. <https://www.youtube.com/watch?v=SFVu-0AqpG8>

12. ENLACES DE REFUERZO Y AMPLIACIÓN

- Documentación oficial del SGBD MySQL de la sintaxis de la sentencia SELECT (En Inglés): <https://dev.mysql.com/doc/refman/5.7/en/select.html>
- Documentación oficial del SGBD ORACLE de la sintaxis de la sentencia SELECT con ejemplos (En Inglés): https://docs.oracle.com/cd/B12037_01/server.101/b10759/statements_10002.htm#i2065706
- Documentación oficial del SGBD POSTGRESQL de la sintaxis de la sentencia SELECT (En Inglés): <http://www.postgresql.org/docs/9.0/static/sql-select.html>
- Sintaxis de la sentencia SELECT junto con la clausula FROM y ejemplos On-Line http://www.w3schools.com/sql/sql_select.asp
- Uso de la clausula WHERE junto con Operadores Relacionales con la sentencia SELECT y ejemplos On-Line http://www.w3schools.com/sql/sql_where.asp
- Uso de operadores Lógicos AND y OR en la clausula WHERE y ejemplos On-Line http://www.w3schools.com/sql/sql_and_or.asp