Lenguaje SQL

1 INTRODUCCIÓN

1.1 ¿QUÉ ES?

El lenguaje de consulta estructurado o SQL (por sus siglas en inglés structured query language) es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales.

1.2 ¿POR QUÉ SURGE?

Debido a la diversidad de lenguajes y de bases de datos existentes, la manera de comunicarse entre unos y otros sería realmente complicada de gestionar de no ser por la existencia de estándares que nos permitan realizar las operaciones básicas de una forma universal.

1.3 UN POCO DE HISTORIA

- ✓ Al principio de los años setenta, los laboratorios de investigación Santa Teresa de IBM empezaron a trabajar en el proyecto System R. El objetivo de este proyecto era implementar un prototipo de SGBD relacional. A mediados de los años setenta, el proyecto de IBM dio como resultado un primer lenguaje denominado SEQUEL (Structured English Query Language), que por razones legales se denominó más adelante SQL (Structured Query Language).

  Al final de la década de los setenta y al principio de la de los ochenta, una vez finalizado el proyecto System R, IBM y otras empresas empezaron a utilizar el SQL en sus SGBD relacionales, con lo que este lenguaje adquirió una gran popularidad.

- ✓ SQL se convirtió oficialmente en el lenguaje estándar de ANSI (American National Standards Institute) en el año 1986, y de ISO (International Standards Organization) en 1987. Esta versión se denomino SQL/86

- ✓ En 1989, ANSI definió el SQL89 que introduce entre otras mejoras la definición de claves primarias, integridad de los datos, etc.

Una de las características más importantes fue la posibilidad de utilizarse a través de dos interfaces: interactivamente o dentro de programas de aplicación, como por ejemplo, Cobol, Fortran, Pascal, etc.

- ✓ En el año 1992 el estándar volvió a ser revisado y ampliado considerablemente para cubrir carencias de la versión anterior. Esta nueva versión del SQL, que se conoce con el nombre de SQL2 o SQL92, es la que nosotros presentaremos en este curso.

- ✓ En la actualidad el SQL es el estándar de facto de la inmensa mayoría de los SGBD comerciales. Y, aunque la diversidad de añadidos particulares que incluyen las distintas implementaciones comerciales del lenguaje es amplia, el soporte al estándar SQL-92 es general y muy amplio. Los sistemas de gestión de base de datos con soporte SQL más utilizados son, por orden alfabético:
    - o DB2
    - o Firebird
    - o Informix
    - o Interbase
    - o MySQL
    - o Oracle
    - o PostgreSQL
    - o Pervasive
    - o SQLite
    - o SQL Server
    - o Sybase ASE

1.4 CARACTERÍSTICAS

Es un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje de programación (ASP o PHP, etc.) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL...).

Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier

acción sobre la base de datos. El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo.

Además, el SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables que contrasta, por otra, con su accesibilidad de aprendizaje.

Por último comentar que el hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. En efecto, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

2 COMPONENTES DEL SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

2.1 COMANDOS

Existen dos tipos de comandos SQL:

- ✓ DDL (Data Definition Language) – Lenguaje de definición de datos

Permiten crear y definir nuevas bases de datos, campos en índices.

Comando	Descripción
CREATE	Este comando crea un objeto dentro de la base de datos
ALTER	Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.
DROP	Este comando elimina un objeto de la base de datos. Se puede combinar con la sentencia ALTER
TRUNCATE	Este comando borra todo el contenido de una tabla

Un objeto de la base de datos puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte

- ✓ DML (Data Manipulation Language) – Lenguaje de manipulación de datos

Permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Comando	Descripción
INSERT	Este comando agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional
UPDATE	Este permite modificar los valores de un conjunto de registros existentes en una tabla
DELETE	Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado

2.2 CLÁUSULAS

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desee seleccionar o manipular.

Cláusula	Descripción
FROM	Utilizada para especificar la tabla de la cual se van a seleccionar los registros
WHERE	Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar
GROUP BY	Utilizada para separar los registros seleccionados en grupos específicos
HAVING	Utilizada para expresar la condición que debe satisfacer cada grupo
ORDER BY	Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico. Por defecto, es decir si no se indica nada se ordenan ascendente (ASC). Se pueden ordenar descendente utilizando (DESC)

2.3 OPERADORES LÓGICOS

Los operadores lógicos se utilizan en las condiciones de las consultas para poder filtrar los datos.

Operador	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	Negación lógica. Devuelve el valor contrario de la expresión.

2.4 OPERADORES DE COMPARACIÓN

Los operadores de comparación, al igual que los lógicos, se utilizan en las condiciones de las consultas para poder filtrar los datos.

Operador	Descripción
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor ó Igual que
>=	Mayor ó Igual que
=	Igual que
BETWEEN…AND	Utilizado para especificar un intervalo de valores
LIKE	Selecciona los registros cuyo valor de campo se asemeje, no teniendo en cuenta mayúsculas y minúsculas
IN y NOT IN	Da un conjunto de valores para un campo para los cuales la condición de selección es (o no) valida
IS Null y IS NOT Null	Selecciona aquellos registros donde el campo especificado esta (o no) vacío.

2.5 FUNCIONES DE AGREGADO

Las funciones de agregado permiten calcular ciertos estadísticos comunes en base a los valores de los campos de las tablas.

Función	Descripción
COUNT	Devuelve el conteo de los registros de la selección
SUM	Calcula la suma de los registros del campo especificado
AVG	Calcula el promedio de los valores de un campo determinado
MAX	Nos indica cual es el valor máximo del campo
MIN	Nos indica cual es el valor mínimo del campo

2.6 COMODINES

Los comodines son caracteres que tienen una funcionalidad particular y que son muy útiles a la hora de generar consultas.

Comodín	Descripción
*	Sustituye a todos los campos
% o *	Sustituye a cualquier cosa o nada dentro de una cadena
_ o ?	Sustituye un solo carácter dentro de una cadena

3 CONSULTAS DE SELECCIÓN

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros que se pueden almacenar en un objeto recordset. Este conjunto de registros es modificable.

SQL tiene una sentencia básica para recuperar información de una base de datos: la sentencia SELECT.

Hay muchas opciones y matices para la sentencia SELECT de SQL, por lo que presentaremos sus características gradualmente.

3.1 CONSULTAS BÁSICAS

La forma básica de la sentencia SELECT consta de cláusulas SELECT y FROM y tiene la siguiente forma:

```
SELECT    <campos>
FROM      <tablas>
```

Donde:

<campos> es una lista de atributos de las tablas contenidas en <lista de tablas> cuyos valores van a ser recuperados por la consulta.

EJEMPLO

Recuperar los nombres y apellidos de los empleados.

```
SELECT nombre, apellidos FROM empleados;
```

A la hora de definir consultas hay que tener en cuenta:

- ✓ Los diferentes campos y tablas de la consulta SQL van separados entre sí por comas.
- ✓ SQL NO distingue entre mayúsculas y minúsculas.
- ✓ Todas las sentencias SQL tienen que terminar con punto y coma (;)

Se pueden utilizar alias para renombrar los campos que se devuelven de la consulta. Para ello SQL dispone de la palabra reservada AS. La sintaxis es:

<campo a renombrar> AS <nombre a mostrar>

Por ejemplo

```
SELECT nif as dni FROM Empleados;
```

Ejercicios

1.1 Recuperar el nif, nombre, apellidos y fecha de nacimiento de los empleados
1.2 Recuperar todos los campos de la tabla Empleados
1.3 Recuperar los departamentos de la empresa y los NIFs de sus jefes
1.4 Recuperar las localizaciones de la empresa
1.5 Recuperar los proyectos y sus identificadores de la empresa
1.6 Recuperar el nif del empleado y los nombres de los familiares y su parentesco

Nomenclatura

Para que la revisión y el mantenimiento de las consultas sea una labor lo más sencilla posible es altamente recomendable seguir una nomenclatura que permita identificar los fallos y/o modificar consultas de una forma rápida y efectiva. La nomenclatura que vamos a seguir a lo largo del curso es:

- ✓ Los comandos, cláusulas, y operadores (lógicos y de comparación) de sentencias siempre con Mayúsculas.
- ✓ Los nombres de las tablas, campos y consultas en minúsculas

Aunque se pueden utilizar acentos y espacios en blanco para nombrar los campos, las tablas y las consultas es recomendable evitarlo porque cuando se exportan tablas a otros sistemas los acentos y los espacios en blanco pueden producir errores.

Si se utilizan espacios en blanco para llamar tablas o consultas se deben incluir sus nombres entre corchetes. Por ejemplo:

```
SELECT [Nombre Producto] FROM [Detalles del Pedido];
```

3.2 CONSULTAS CON CONDICIONES

Las condiciones dentro de la consulta SQL se especifican mediante la cláusula WHERE que se pospone a la forma básica quedando por tanto la estructura

```
SELECT    <campos>
FROM      <tablas>
WHERE     <condiciones>
```

Con este tipo de consultas se filtran los registros con el fin de recuperar solamente aquellos que cumplan unas condiciones preestablecidas.

A la hora de definir las consultas hay que tener en cuenta:

- ✓ Cuando se desee establecer una condición referida a un campo de texto la condición de búsqueda debe ir encerrada entre comillas simples;
- ✓ Las fechas se deben escribir siempre entre almohadillas (#) y con el formato mm-dd-aaaa en donde mm representa el mes, dd el día y aaaa el año.

  El separador de las fechas debe ser un guión (-) aunque muchas bases de datos permiten también la barra (/).

  Por ejemplo si deseamos referirnos al día 6 de Marzo de 1992 deberemos hacerlo de la siguiente forma; #03-06-1992# ó #3-6-1992#.
- ✓ Es usual agrupar las condiciones entre paréntesis

EJEMPLO

Recuperar el nombre y apellidos de los empleados cuyo salario es superior a 22000

```
SELECT Nombre, Apellidos FROM Empleados WHERE salario > 22000;
```

Filtros utilizando los operadores lógicos

Los operadores lógicos, en la consulta se utilizan con la siguiente sintaxis

<expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar. El resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

<expresión1>	Operador	<expresión1>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso

Si a cualquiera de las anteriores condiciones le anteponemos el operador NOT el resultado de la operación será el contrario al devuelto sin el operador NOT.

Filtros utilizando los operadores de comparación

Los operadores de comparación se utilizan en las consultas con la siguiente sintaxis:

<expresión1> operador <expresión2>

En donde expresión1 y expresión2 son las condiciones a evaluar.

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador BETWEEN…AND cuya sintaxis es:

<campo> [Not] BETWEEN valor1 AND valor2 (la condición Not es opcional)

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición NOT devolverá aquellos valores no incluidos en el intervalo.

El operador LIKE se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

<campo> LIKE <modelo>

<campo> es el campo cuyo valor se compara con la expresión indicada en <modelo>. Por modelo puede especificar un valor completo (Ana María), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (LIKE An*).

En la tabla siguiente se muestra cómo utilizar el operador LIKE para comprobar expresiones con diferentes modelos.

Tipo	Modelo	Coincide	No coincide
Varios caracteres	’a*a’	’aa’, ’aBa’, ’aBBBa’	’aBC’
Carácter especial	’a[*]a’	’a*a’	’aaa’
Varios caracteres	’ab*’	’abcdefg’, ’abc’	’cab’, ’aab’
Un solo carácter	’a?a’	’aaa’, ’a3a’, ’aBa’	’aBBBa’
Un solo dígito	’a#a’	’a0a’, ’a1a’, ’a2a’	’aaa’, ’a10a’
Rango de caracteres	’[a-z]’	’f’, ’p’, ’j’	’2’, ’&’
Fuera de un rango	’[!a-z]’	’9’, ’&’, ’%’	’b’, ’a’
Distinto de un dígito	’[!0-9]’	’A’, ’a’, ’&’, ’~’	’0’, ’1’, ’9’
Combinada	’a[!b-m]#’	’An9’, ’az0’, ’a99’	’abc’, ’aj0’

Como se puede observar el operador LIKE distingue entre mayúsculas y minúsculas.

El ejemplo siguiente devuelve los datos que comienzan con la letra P seguido de cualquier letra entre A y F y de tres dígitos:

```
LIKE ’P[A-F]###’
```

Este ejemplo devuelve los campos cuyo contenido empiece con una letra de la A a la D seguidas de cualquier cadena.

```
LIKE ’[A-D]*’
```

El operador IN devuelve aquellos registros cuyo campo indicado coincide con alguno de los en una lista. Su sintaxis es:

```
expresión [Not] IN (valor1, valor2, . . .)
```

El operador IS se emplea para comparar dos variables de tipo objeto. Este operador devuelve verdad si los dos objetos son iguales. La sintaxis es la siguiente:

```
<campo> IS <Objeto2>
```

Generalmente es común utilizarlo con el término NULL para comprobar si un campo es vacío.

Ejercicios

2.1 Recuperar el nombre y apellido del empleado con el nif 12345674D

2.2 Recuperar el nif y dirección de la empleada Ana Villaverde

2.3 Recuperar el número y nombre de los departamentos diferentes del departamento Direccion

2.4 Recuperar las localizaciones diferentes de Pamplona y Donosti

2.5 Recuperar el nombre, apellidos y fecha de nacimiento de los empleados con fecha de nacimiento posterior al 1 de Enero de 1970

2.6 Recuperar el nombre, apellidos y fecha de nacimiento de los empleados con fecha de nacimiento entre el 1 de Junio de 1965 y el 31 de Diciembre de 1980

2.7 Recuperar el nombre y apellidos de los empleados cuyo salario es igual o inferior a 22000

2.8 Recuperar el nombre y apellidos de los empleados cuyo salario está comprendido entre 17000 y 22000

2.9 Recuperar el departamento y el nif jefe asociado para los departamentos de Desarrollo e Investigacion

2.10 Recuperar nombre, apellidos, fecha de nacimiento y salario de los empleados cuya fecha de nacimiento está comprendida entre el 1 de Enero de 1979 y el 31 de Diciembre de 2009 o su sueldo es superior a 20000.

2.11 Recuperar nif, nombre y apellidos de los empleados cuyo apellidos comienzan por ‘Go’

2.12 Recuperar todos los campos de los empleados que no están asignados a ningún departamento

2.13 Recuperar los nombres de los proyectos y las localizaciones que contengan ‘01’

2.14 Recuperar todos los campos de los proyectos que cuyo nombre empiece por ‘Des’ pero su número de departamento sea distinto de 3

2.15 Recuperar todos los campos de Empleados cuyo campo sexo sea distinto de H o M

2.16 Recuperar nif, nombre y apellidos de los empleados cuyo nombre comienza por M

2.17 Recuperar nombre, apellidos y dirección de los empleados que tengan como numero de calle 47

2.18 Recuperar nif, nombre y apellidos de los empleados que termine con una ‘C’ en su nif

3.3 ORDENANDO LOS REGISTROS

En una consulta SQL se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY. La sintaxis de uso de la cláusula es:

```
SELECT      <campos>
FROM        <tablas>
[ WHERE     <condiciones> ] (la cláusula WHERE es opcional)
ORDER BY    <campos> [ASC / DESC] (las cláusulas ASC y DESC son
                                   opcionales)
```

Las cláusulas ASC y DESC especifican el orden de los registros: ASCENDENTE o DESCENDENTE. Si no se especifica el tipo de ordenación, por defecto, los registros se muestran ordenados ascendentes, es decir, como si se especificara la cláusula ASC.

EJEMPLO

Recuperar el nif del empleado y los nombres de sus familiares asociados ordenando ascendente por nif y descendente por nombre.

```
SELECT nif, nombre FROM familiares ORDER BY nif, nombre DESC;
```

3.4 CONSULTAS CON CONDICIONES ADICIONALES

El lenguaje SQL proporciona una serie de cláusulas adicionales que nos proporcionan mayor funcionalidad a la hora de realizar consultas.

Las cláusulas que vamos a presentar a continuación se caracterizan porque todas ellas definen condiciones de filtrado entre el comando SELECT y el primer nombre del campo a recuperar.

Cláusula	Descripción
ALL	Devuelve todos los campos de la tabla
TOP	Devuelve un determinado número de registros de la tabla
DISTINCT	Omite los registros cuyos campos seleccionados coincidan totalmente
DISTINCTROW	Omite los registros duplicados basándose en la totalidad del registro y no sólo en los campos seleccionados.

3.4.1 ALL

Con esta cláusula el motor de la base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL.

Es equivalente a utilizar '*' en la definición de los campos de la consulta.

EJEMPLO

Recuperar todos los campos de la tabla Empleados.

```
SELECT ALL FROM Empleados;
```

Nota: No es conveniente abusar de esta cláusula ya que obligamos al motor de la base de datos a analizar la estructura de la tabla para averiguar los campos que contiene. Es mucho más eficiente indicar el listado de campos deseados.

3.4.2 TOP

Devuelve un cierto número de registros. El número de registros que se quieren obtener se especifica a continuación de la cláusula. Normalmente se suele utilizar conjuntamente con la cláusula ORDER BY.

EJEMPLO

Recuperar los cinco primeros empleados ordenados por nombre y apellido.

```
SELECT TOP 5 nombre, apellidos FROM Empleados ORDER BY nombre,apellidos;
```

Como se puede observar, mediante el uso de esta cláusula se puede limitar el número de registros que nos devuelve la consulta.

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de registros.

Además hay que tener en cuenta que la cláusula TOP no distingue entre registros iguales de forma que si los últimos registros que se devuelven en la consulta son iguales, TOP los devolverá todos independientemente de que sobrepase el límite fijado en la consulta.

Conjuntamente con la cláusula TOP se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros del total de registros que devolvería la consulta sin la utilización de la cláusula.

EJEMPLO

Recuperar el veinticinco por ciento de los empleados ordenados por nombre y apellido.

```
SELECT TOP 25 PERCENT nombre, apellidos FROM Empleados ORDER BY nombre,apellidos;
```

3.4.3 DISTINCT

Con esta cláusula el motor de la base de datos omite los registros que contienen datos duplicados en los campos seleccionados, es decir, para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta, éstos deben ser únicos.

EJEMPLO

Recuperar sólo los nombres diferentes de los familiares de los empleados.

```
SELECT DISTINCT nombre FROM familiares;
```

3.4.4 DISTINCTROW

Con esta cláusula el motor de la base de datos devuelve los registros diferentes en una tabla, independientemente de los campos seleccionados en la consulta.

A diferencia de la cláusula anterior (DISTINCT) que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campos indicados en la cláusula SELECT.

EJEMPLO

Recuperar los nombres de los familiares de los empleados omitiendo aquellos registros duplicados que tienen todos los campos iguales

```
SELECT DISTINCTROW nombre FROM familiares;
```

Ejercicios

3.1 Recuperar el nif, nombre, apellidos y fecha de nacimiento de los empleados ordenados por fecha de nacimiento ascendente.

3.2 Recuperar los departamentos de la empresa ordenados descendente.

3.3 Recuperar las localizaciones de la empresa ordenadas ascendente;

3.4 Recuperar el nif del empleado, nombre del familiar y fecha de nacimiento del familiar, ordenados por nif descendente, nombre ascendente y fecha de nacimiento descendente.

3.5 Recuperar todos los campos de la tabla Familiares (sin definir todos los campos en la consulta y sin utilizar el comodín '*')

3.6 Recuperar solo una localización diferente de Pamplona y Donosti

3.7 Recuperar el cincuenta por ciento de proyectos cuyo nombre empieza por 'Dir'

3.8 Recuperar los nifs de los empleados que tienen familiares

3.9 Recuperar los diferentes números de departamentos de la tabla Empleados

3.10 Recuperar el nombre, apellidos y fecha de nacimiento de los empleados con fecha de nacimiento entre el 1 de Junio de 1965 y el 31 de Diciembre de 1980 ordenados descendente por fecha de nacimiento y ascendente por nombre y apellidos.

3.11 Recuperar dos registros con el nombre, apellidos y fecha de nacimiento de los empleados con fecha de nacimiento posterior al 1 de Enero de 1970 ordenados descendente por la fecha de nacimiento.

3.12 Recuperar nombre, apellidos, fecha de nacimiento y salario de los empleados cuya fecha de nacimiento esta comprendida entre el 1 de Enero de 1979 y el 31 de Diciembre de 2009 o su sueldo es superior a 20000 ordenados descendente por el salario.

3.13 Recuperar el cincuenta por ciento de los registros formados por nif, nombre y apellidos de los empleados cuyos apellidos comienzan por 'Go' ordenados por nombre ascendente.

3.14 Recuperar el cincuenta por ciento de los registros formados por nif, nombre y apellidos de los empleados cuyos apellidos comienzan por 'Go' ordenados por nombre descendente.

3.15 Recuperar los nifs diferentes de los empleados que han trabajado la última semana.

3.16 Recuperar los nifs y numeros de proyecto diferentes de los empleados que han trabajado la última semana.

3.5 CONSULTAS CON OPERACIONES ARITMÉTICAS

En una consulta SQL, además de recuperar campos de tablas, se pueden obtener resultados en base a operaciones que se realicen sobre los campos de las tablas. Las operaciones aritméticas típicas son la suma, la resta, la multiplicación y la división. Los operadores que se utilizan son habituales:

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División

Los operadores aritméticos se utilizan en las consultas con la siguiente sintaxis:

<campo1> operador <campo2, valor>

EJEMPLO

Recuperar el salario de los empleados en pesetas

```
SELECT salario * 166.386 FROM empleados;
```

3.6 CONSULTAS AGRUPANDO REGISTROS

El lenguaje SQL permite la agrupación de registros mediante la cláusula GROUP BY. La cláusula GROUP BY agrupa, como su propio nombre indica, filas que tienen el mismo valor para un atributo, en grupos distintos.

La sintaxis de uso de la cláusula es:

```
SELECT      <campos>
FROM        <tablas>
[ WHERE     <condiciones> ] ( la cláusula WHERE es opcional)
GROUP BY    <campos>
[ ORDER BY  <campos> ] ( la cláusula ORDER BY es opcional)
```

EJEMPLO

Recuperar los nombres de los familiares de los empleados agrupando por el nombre.

```
SELECT  nombre FROM familiares GROUP BY nombre;
```

El lenguaje SQL permite además filtrar los resultados en función de las agrupaciones realizadas por la cláusula GROUP BY. Para ello se utiliza la cláusula HAVING.

La cláusula HAVING es similar a la cláusula WHERE, salvo que aquella se usa como condición de búsqueda cuando se especifica la cláusula GROUP BY. Por lo tanto el funcionamiento es similar al ya visto para WHERE. La única diferencia es que HAVING se aplica a condiciones de grupo.

La sintaxis de uso de la cláusula es:

```
SELECT      <campos>
FROM        <tablas>
[ WHERE     <condiciones para campos> ] ( la cláusula WHERE es opcional)
GROUP BY    <campos>
HAVING      <condiciones para grupos>
[ ORDER BY  <campos> ] ( la cláusula ORDER BY es opcional)
```

EJEMPLO

Recuperar los nombres de los familiares de los empleados agrupando por el nombre y mostrando los registros que empiecen por ‘P’

```
SELECT nombre FROM familiares GROUP BY nombre HAVING nombre LIKE ‘P*’;
```

3.7 CONSULTAS CON FUNCIONES DE AGREGADO

Entendemos por funciones de agregado, todas aquellas que permiten realizar operaciones de conteo de filas, suma de atributos, obtención de medias, etc. Dichas funciones se especifican a continuación de la palabra reservada SELECT. Las funciones soportadas por SQL son:

Función	Descripción
COUNT	Devuelve el conteo de los registros de la selección
SUM	Calcula la suma de los registros del campo especificado
AVG	Calcula el promedio de los valores de un campo determinado
MAX	Nos indica cual es el valor máximo del campo
MIN	Nos indica cual es el valor mínimo del campo

3.7.1 COUNT

Cuenta todas las filas de las tablas accedidas mediante una consulta SQL.

Su sintaxis es la siguiente:

COUNT(expr)

Donde expr contiene el nombre del campo que se desea contar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL). Puede contar cualquier tipo de datos incluso texto.

Aunque expr puede realizar un cálculo sobre un campo, COUNT simplemente cuenta el número de registros sin tener en cuenta qué valores se almacenan en los registros. La función COUNT no cuenta los registros que tienen campos NULL a menos que expr sea el carácter comodín asterisco (*). Si utiliza un asterisco, COUNT calcula el número total de registros, incluyendo aquellos que contienen campos NULL.

COUNT(*) es considerablemente más rápida que COUNT(Campo).

Nota: No se debe poner el asterisco entre comillas ('*')

Si expr identifica a múltiples campos, la función COUNT cuenta un registro sólo si al menos uno de los campos no es NULL. Si todos los campos especificados son NULL, no se cuenta el registro. Hay que separar los nombres de los campos con ampersand (&).

EJEMPLO

Recuperar el total de empleados de la empresa

```
SELECT COUNT(*) FROM empleados;
```

3.7.2 SUM

Realiza una suma acumulativa de un atributo para todas las filas accedidas mediante una consulta SQL.

Su sintaxis es la siguiente:

```
SUM(expr)
```

Donde expr contiene el nombre del campo que se desea sumar. Los operandos de expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL.

EJEMPLO

Recuperar la suma de las horas de todos los empleados de la empresa

```
SELECT SUM(horas) FROM trabaja_en;
```

3.7.3 AVG

Realiza una media aritmética de los atributos para todas las filas accedidas mediante la consulta SQL.

Su sintaxis es la siguiente:

```
AVG(expr)
```

En donde expr representa el campo que contiene los datos numéricos para los que se desea calcular la media o una expresión que realiza un cálculo utilizando los datos de dicho campo. La media calculada por AVG es la media aritmética (la suma de los valores dividido por el número de valores). La función AVG no incluye ningún campo NULL en el cálculo.

EJEMPLO

Recuperar la media de los salarios de todos los empleados de la empresa

```
SELECT AVG(salario) FROM empleados;
```

3.7.4 MAX

Obtiene el máximo valor del atributo especificado, de entre todas las filas seleccionadas mediante la sentencia SQL.

Su sintaxis es la siguiente:

```
MAX(expr)
```

Donde expr contiene el nombre del campo que se desea realizar el cálculo. Expr puede incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL.

EJEMPLO

Recuperar el máximo salario de todos los empleados de la empresa

```
SELECT MAX(salario) FROM empleados;
```

3.7.5 MIN

Obtiene el mínimo valor del atributo especificado, de entre todas las filas seleccionadas mediante la sentencia SQL.

Su sintaxis es la siguiente:

```
MIN(expr)
```

Donde expr contiene el nombre del campo que se desea realizar el cálculo. Expr puede incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL.

EJEMPLO

Recuperar el mínimo salario de todos los empleados de la empresa

```
SELECT MIN(salario) FROM empleados;
```

Ejercicios

4.1 Recuperar el número de empleados de la empresa

4.2 Recuperar el sueldo medio de los empleados de la empresa.

4.3 Recuperar el gasto total en pesetas realizado por la empresa por el pago de los sueldos de los empleados.

4.4 Recuperar los numeros de proyecto de la tabla Trabaja_En sin utilizar la cláusula DISTINCT.

4.5 Recuperar el número de familiares asociados a los empleados.

4.6 Recuperar todas las horas trabajadas por los empleados.

4.7 Recuperar el número de delegaciones de la empresa.

4.8 Recuperar los nifs de los empleados que tienen familiares sin utilizar la cláusula DISTINCT.

4.9 Recuperar el total de horas trabajadas para el número de proyecto 21.

4.10 Recuperar el nombre de los familiares y las veces que aparecen repetidos.

4.11 Recuperar el número de empleados que tiene hijos o hijas.

4.12 Recuperar el número de empleados cuyo sueldo es superior a 18000.

4.13 Recuperar el número de proyectos de la empresa.

4.14 Recuperar el número de empleadas de la empresa.

4.15 Recuperar el número de empleados que están casados.

4.16 Recuperar el numero de empleados cuyo nombre empieza por ‘J’ y su apellido contiene una ‘e’’.

4.17 Recuperar el número de empleados que trabajan en el número de proyecto 31.

4.18 Recuperar el número de proyectos que se realizan en cada delegación.

4.19 Recuperar el número de empleados nacidos entre el 25 de Febrero de 1972 y el 15 de Agosto de 1997

4.20 Recuperar el número de empleados cuyo apellidos comienzan por Go

4.21 Recuperar el sueldo medio de los empleados de la empresa sin utilizar la función AVG.

3.8 SUBCONSULTAS

Se utilizan para expresar ciertas condiciones cuando no hay más remedio que obtener el valor que buscamos como resultado de una consulta.

Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT...INTO, DELETE, o UPDATE o dentro de otra subconsulta.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT o en una cláusula WHERE o HAVING

Existen cuatro formas de sintaxis para crear una subconsulta:

1. SELECT <campos>
   FROM <tablas>
   WHERE <campo> [NOT] IN (<Instrucción SQL>)
   [GROUP BY / HAVING/ ORDER BY]

Donde <Instrucción SQL> es una instrucción SELECT, que sigue el mismo formato y reglas que cualquier otra instrucción SELECT.

Muy Importante: La instrucción SQL debe ir entre paréntesis.

El predicado IN se emplea para recuperar únicamente aquellos registros de la consulta principal para los que los registros de la subconsulta contienen un valor igual.

Inversamente se puede utilizar NOT IN para recuperar únicamente aquellos registros de la consulta principal para los que no hay ningún registro de la subconsulta que contenga un valor igual.

EJEMPLO

Recuperar el nif y nombre de los empleados que trabajan en el departamento de Investigacion.

```
SELECT nif, nombre FROM empleados WHERE num_dpto IN
(SELECT num_dpto FROM departamentos WHERE nombre_dpto = 'Investigacion');
```

2. SELECT <campos>
   FROM <tablas>
   WHERE [NOT] EXISTS (<Instrucción SQL>)
   [GROUP BY / HAVING/ ORDER BY]

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro.

3. SELECT <campos>
   FROM <tablas>
   WHERE <comparación> [ANY | ALL | SOME] (<Instrucción SQL>)
   [GROUP BY / HAVING/ ORDER BY]

Donde <comparación> es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta.

Los predicados ANY o SOME se utilizan para recuperar aquellos registros de la consulta principal que satisfacen la comparación con cualquier otro registro recuperado en la subconsulta.

EJEMPLO

Recuperar nif, nombre, apellidos y sueldo de los empleados cuyos sueldos son superiores a 18000 y menores que 26000

```
SELECT nif, nombre, apellidos, salario FROM empleados WHERE salario > ANY (SELECT 18000+(num_dpto*2000) FROM departamentos);
```

Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos empleados que tienen un sueldo superior a 26000.

4. SELECT (<Instrucción SQL>) [<campos>]
   FROM <tablas>
   [WHERE / GROUP BY / HAVING/ ORDER BY]

En este caso se utiliza una subconsulta para obtener otros campos en la lista de campos de una instrucción SELECT.

EJEMPLO

Recuperar el nombre del departamento y la localización para el número de departamento 1

```
SELECT (SELECT nombre_dpto FROM departamentos WHERE num_dpto=1)
AS departamento, localizacion FROM localizaciones WHERE num_dpto=1
```

En este caso se utiliza una subconsulta en lugar de una expresión en la lista de campos de una instrucción SELECT.

3.9 UNION

Se utiliza la operación UNION para crear una consulta combinando los resultados de dos o más consultas independientes.

A este tipo de consultas se las denomina consultas de unión externas.

Su sintaxis es:

```
SELECT      (campos)
FROM        (tablas)
[ WHERE / GROUP BY / HAVING]
UNION       [ ALL ]
SELECT      (campos)
FROM        (tablas)
[ WHERE / GROUP BY / HAVING]
[ ORDER BY ]
```

- ✓ Todas las consultas en una operación UNION deben recuperar el mismo número de campos, no obstante los campos no tienen porqué tener el mismo tamaño o el mismo tipo de datos.
- ✓ Si no se indica lo contrario, no se devuelven registros duplicados cuando se utiliza la operación UNION.
- ✓ Se puede incluir el predicado ALL para asegurar que se devuelven todos los registros. Esto hace que la consulta se ejecute más rápidamente.
- ✓ Se puede utilizar una cláusula GROUP BY y/o HAVING en cada argumento consulta para agrupar los datos devueltos.
- ✓ Se puede utilizar una cláusula ORDER BY al final del último argumento consulta para visualizar los datos devueltos en un orden específico.

EJEMPLO

Recuperar todos los nombres registrados en la empresa

```
SELECT nombre FROM empleados UNION SELECT nombre FROM familiares
```

También se pueden crear uniones a partir de la unión de una tabla entera con una consulta o con otra tabla. La sintaxis en este caso es:

```
TABLE        (Nombre Tabla)
UNION        [ ALL ]
SELECT       (campos)
FROM         (tablas)
[ WHERE / GROUP BY / HAVING]
[ ORDER BY ]
```

EJEMPLO

Recuperar en una única tabla la información de la tabla Localizaciones y los campos nombre_dpto y num_dpto de la tabla Departamentos

```
TABLE localizaciones UNION SELECT nombre_dpto, num_dpto FROM departamentos
```

3.10 CONSULTAS CON PARÁMETROS

Las consultas con parámetros son aquellas cuyas condiciones de búsqueda se definen mediante parámetros.

Si se ejecutan directamente desde la base de datos donde han sido definidas aparecerá un mensaje solicitando el valor de cada uno de los parámetros.

Si deseamos ejecutarlas desde una aplicación hay que asignar primero el valor de los parámetros y después ejecutarlas. Su sintaxis es la siguiente:

PARAMETERS < nombre1 > < tipo1 >, ... , < nombreN > < tipoN >

< Consulta SQL >

Donde:

< nombre1 >,... < nombreN > representan los nombres de los parámetros.

< tipo1 >,..., < tipoN > representan los tipos de datos asociados a los nombres anteriores, respectivamente.

< Consulta SQL > es la consulta SQL que queremos realizar.

- ✓ Se puede utilizar nombres pero no tipos de datos en una cláusula WHERE o HAVING.
- ✓ En Microsoft Access los parámetros se definen directamente con el nombre que se desee en la consulta SQL. La única condición es que los nombres de los parámetros no coincidan con el nombre de ningún campo de las tablas definidas en la consulta SQL.

EJEMPLO

Recuperar los datos del empleado cuyo nombre se introduzca como parámetro

SELECT * FROM empleados WHERE nombre = nombre_param

3.11 EL OPERADOR JOIN

El operador JOIN es uno de los más usados en el lenguaje SQL. Sirve para combinar dos tablas entre sí, utilizando un atributo común. Lo que se realiza es una unión de cada fila de una tabla, con todas las filas de la tabla con la que se hace el JOIN, cuyos atributos comunes coincidan.

La sintaxis general es:

```
SELECT      (campos)
FROM        <tabla1> [ INNER / LEFT / RIGHT ] JOIN <tabla2>
ON  <tabla1.campo1> <operador de comparación> <tabla2.campo2>
```

Donde:

<tabla1> y <tabla2> son los nombres de las tablas de las que se combinan los registros.

<tabla1.campo1> y <tabla2.campo2> son los nombres de los campos por los que se realiza la combinación. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.

<operador de comparación> es cualquier operador de comparación relacional: =, <, >, <=, >=, o <>.

Se puede utilizar una operación INNER JOIN en cualquier cláusula FROM. Esto crea una combinación por equivalencia. Es decir, si empleamos la cláusula INNER en la consulta se seleccionarán sólo aquellos registros de <tabla1> que contengan al menos un registro en <tabla2>.

Las combinaciones utilizando el operador "=" son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas

Si empleamos la cláusula LEFT, se seleccionarán todos los registros de <tabla1> aunque no tengan ningún registro en <tabla2>.

RIGHT realiza la misma operación pero al contrario, toma todos los registros de <tabla2> aunque no tenga ningún registro en <tabla1>.

El tipo más común de combinación es INNER JOIN, por lo que es considerado el tipo de combinación predeterminado.

Es tan común la utilización del operador JOIN que la forma más habitual de realizar una consulta combinada es simplemente especificando las tablas y los campos por los que se realiza la combinación:

```
SELECT      (campos)
FROM        <tabla1>, <tabla2>
WHERE  <tabla1.campo1> <operador de comparación> <tabla2.campo2>
```

Como se puede apreciar no se especifica el operador JOIN. A esta forma de expresar la consulta se llama forma implícita.

EJEMPLO

Recuperar nombre de empleado y nombre de departamento asociado.

```
SELECT empleados.nombre, empleados.num_dpto,
departamentos.nombre_dpto, departamentos.num_dpto
FROM empleados, departamentos
WHERE empleados.num_dpto=departamentos.num_dpto;
```

Utilizando la sintaxis explícita la consulta quedaría:

```
SELECT empleados.nombre, empleados.num_dpto,
departamentos.nombre_dpto, departamentos.num_dpto
FROM empleados INNER JOIN departamentos
ON empleados.num_dpto=departamentos.num_dpto;
```

Consideraciones a tener en cuenta:

- ✓ Se pueden enlazar varias cláusulas ON en una instrucción JOIN, utilizando la sintaxis siguiente:

```
SELECT campos
FROM tabla1 INNER JOIN tabla2
ON tabla1.campo1 <op. comparación> tabla2.campo1 <operador>
ON tabla1.campo2 <op. comparación> tabla2.campo2)
```

- ✓ También puede anidarse instrucciones JOIN utilizando la siguiente sintaxis:

```
SELECT campos
FROM tabla1 INNER JOIN
(tabla2 INNER JOIN tabla3
[INNER JOIN tablaN [INNER JOIN ...)]
ON tablaN.campoN <op. comparación> tabla3.campo3
ON tabla3.campo2 <op. comparación> tabla2.campo2
ON tabla2.campo2 <op. comparación> tabla1.campo1;
```

- ✓ Las operaciones LEFT JOIN y RIGHT JOIN se pueden anidar en una operación INNER JOIN, pero una operación INNER JOIN no se puede anidar en una operación LEFT JOIN o RIGHT JOIN
- ✓ Si se intenta combinar campos que contienen datos de tipo Memo u Objeto OLE, se produce un error.
- ✓ Se pueden combinar dos campos numéricos de tipos similares. Por ejemplo, puede combinar campos de tipo AutoNumber y Long porque son similares. Sin embargo, no puede combinar campos de tipo Single y Double.

3.11.1 TIPOS DE JOIN

3.11.1.1 Combinaciones internas

Son las que tienen la cláusula INNER.

Como ya se ha comentado anteriormente con esta operación se calcula el producto cruzado de todos los registros; así cada registro en la <tabla1> es combinado con cada registro de la <tabla2>; pero sólo permanecen aquellos registros en la tabla combinada que satisfacen las condiciones que se especifiquen.

Este es el tipo de JOIN más utilizado por lo que es considerado el tipo de combinación predeterminado.

También hemos visto que SQL especifica dos formas diferentes para expresar estas combinaciones:

- ✓ Explícita: usa la palabra JOIN.

```
SELECT    (campos)
FROM      <tabla1> INNER JOIN <tabla2>
ON  <tabla1.campo1> <operador de comparación> <tabla2.campo2>
```

- ✓ Implícita: usa ',' para separar las tablas a combinar en la sentencia FROM de la declaración SELECT y las condiciones se indican en la cláusula WHERE.

```
SELECT    (campos)
FROM      <tabla1>, <tabla2>
WHERE  <tabla1.campo1> <operador de comparación> <tabla2.campo2>
```

Es necesario tener especial cuidado cuando se combinan columnas con valores nulos (NULL) ya que el valor nulo no se combina con otro valor o con otro nulo, excepto cuando se le agregan predicados tales como IS NULL o IS NOT NULL.

Las operaciones INNER JOIN pueden ser clasificadas como de equivalencia, naturales, y cruzadas.

DE EQUIVALENCIA (equi-join)

Es una combinación que usa comparaciones de igualdad (=) en el predicado JOIN

EJEMPLO

```
SELECT empleados.nombre, empleados.num_dpto,
departamentos.nombre_dpto, departamentos.num_dpto
FROM empleados INNER JOIN departamentos
ON empleados.num_dpto = departamentos.num_dpto;
```

NATURALES (Natural join)

Es una especialización de la combinación de equivalencia, anteriormente mencionada. En este caso se comparan todas las columnas que tengan el mismo nombre en ambas tablas. La tabla resultante contiene sólo una columna por cada par de columnas con el mismo nombre.

EJEMPLO

```
SELECT *
FROM empleados NATURAL JOIN departamentos;
```

El uso de esta sentencia puede producir resultados ambiguos y generar problemas si la base de datos cambia, porque al añadir, quitar, o renombrar las columnas, puede perder el sentido la sentencia; por esta razón su uso no es muy común. Por ejemplo en Access no están soportadas.

CRUZADAS (Cross join)

Presenta el producto cartesiano de todos los registros de las dos tablas.

El código SQL para realizar este producto cartesiano enuncia las tablas que serán combinadas, pero no incluye algún predicado que filtre el resultado.

EJEMPLO

```
SELECT * FROM empleados CROSS JOIN departamentos;
SELECT * FROM empleados, departamentos;
```

Esta clase de combinaciones son usadas pocas veces ya que generalmente se agregan condiciones de filtrado.

3.11.1.2 Combinaciones externas

Son las que tienen la cláusula OUTER.

En éstas se incluyen:

- ✓ Combinación externa izquierda: LEFT OUTER JOIN o LEFT JOIN
- ✓ Combinación externa derecha RIGHT OUTER JOIN o RIGHT JOIN
- ✓ Combinación externa completa: FULL OUTER JOIN

Combinación externa izquierda (LEFT JOIN)

El resultado de esta operación siempre contiene todos los registros de la tabla de la izquierda (la primera tabla que se menciona en la consulta), aun cuando no exista un registro correspondiente en la tabla de la derecha. Su sintaxis es:

```
SELECT    (campos)
FROM      <tabla1> LEFT [ OUTER ] JOIN <tabla2>
ON  <tabla1.campo1> <operador de comparación> <tabla2.campo2>
```

La sentencia LEFT JOIN retorna la pareja de todos los valores de la tabla izquierda con los valores de la tabla de la derecha correspondientes, o retorna un valor nulo en caso de no correspondencia.

En Access no está soportada la cláusula OUTER.

EJEMPLO

Recuperar el nif, nombre y apellidos de TODOS los empleados y los nombres de los departamentos a los que pertenecen.

```
SELECT empleados.nif, empleados.nombre, empleados.apellidos,
departamentos.nombre_dpto
FROM empleados LEFT JOIN departamentos
ON empleados.num_dpto=departamentos.num_dpto
```

Combinación externa derecha (RIGHT JOIN)

Esta operación es inversa a la anterior; el resultado de esta operación siempre contiene todos los registros de la tabla de la derecha (la segunda tabla que se menciona en la consulta), aun cuando no exista un registro correspondiente en la tabla de la izquierda. Su sintaxis es:

```
SELECT      (campos)
FROM        <tabla1> RIGHT [ OUTER ] JOIN <tabla2>
ON  <tabla1.campo1> <operador de comparación> <tabla2.campo2>
```

La sentencia RIGHT JOIN retorna la pareja de todos los valores de la tabla derecha con los valores de la tabla de la izquierda correspondientes, o retorna un valor nulo en caso de no correspondencia.

En Access no está soportada la cláusula OUTER.

Combinación externa completa (FULL OUTER JOIN)

Esta operación presenta los resultados de tabla izquierda y tabla derecha aunque no tengan correspondencia en la otra tabla. La tabla combinada contendrá, entonces, todos los registros de ambas tablas y presentará valores nulos para registros sin pareja. Su sintaxis es:

```
SELECT      (campos)
FROM        <tabla1> FULL OUTER JOIN <tabla2>
ON  <tabla1.campo1> <operador de comparación> <tabla2.campo2>
```

Algunos sistemas de bases de datos no soportan esta funcionalidad, pero esta puede ser emulada a través de las combinaciones de tabla izquierda, tabla derecha y de la sentencia de unión UNION.

En Access no está soportado.

Ejercicios

5.1 Recuperar el nif, nombre y apellidos de los empleados que son jefes de departamento.

5.2 Suponiendo que todos los hijos/as viven con sus padres, recuperar los nombres ordenados descendentes de los hijos/as cuyos padres no trabajan en Pamplona.

5.3 Recuperar el nif, salario y fecha de nacimiento de los empleados cuyos sueldos son superiores a la media de los sueldos de la empresa y su fecha de nacimiento está comprendida entre el 7 de Marzo de 1962 y el 18 de Noviembre de 1993

5.4 Recuperar el nif, nombre, apellidos, horas trabajadas y número de proyecto en el que las han realizado, ordenando por nombre y apellidos.

5.5 Recuperar los nombres de proyecto, localización y las horas asociadas para los proyectos cuyo nombre empiece por D. Ordenar por horas descendente.

5.6 Recuperar los nombres de los departamentos, fecha de inicio jefe, nif del jefe y nombre y apellidos para los departamentos ubicados en Pamplona y Donosti ordenados por nombre de departamento descendente.

5.7 Recuperar nombre, apellidos del empleado, los nombres de sus familiares y su parentesco tomando como parámetros de entrada el nif del empleado. Mostrar los resultados ordenados por nombre del familiar.

5.8 Recuperar nif, nombre, apellidos y horas trabajadas para el número de proyecto que se introducirá por teclado.

5.9 Recuperar nif del empleado, nombre, apellidos, los nombres de sus familiares y su parentesco, en caso de que los tuviera, para los jefes de departamento de Desarrollo e Investigación. Mostrar los resultados ordenando por nombre y apellidos del empleado.

5.10 Recuperar nif, nombre, apellidos, horas trabajadas y números de proyecto asociados para el departamento que se introducirá por teclado.

5.11 Recuperar para cada nombre de departamento el número de empleados que trabajan en él.

5.12 Recuperar para cada empleado su nombre y apellidos y el número de familiares.

Ejercicios

6.1 Recuperar matricula, fecha de venta y precio de venta de los coches vendidos en el primer semestre del año pasado

6.2 Recuperar matrícula y fecha de revisión de los coches cuyo código de revisión incluye una A.

6.3 Recuperar el número de coches vendidos agrupando por sexo.

6.4 Recuperar código de revisión y matrícula de las revisiones de Junio del 2010.

6.5 Recuperar para el nombre introducido por teclado los datos de los coches que ha comprado (mostrar matrícula, fecha de venta y precio de venta).

6.6 Recuperar el número de coches vendidos en el año 2009.

6.7 Recuperar los datos de los coches cuyas matrículas contienen una B y terminan con K.

6.8 Recuperar por cada cliente (nombre), el número de coches comprados.

6.9 Recuperar de los clientes nacidos en el año 66 en adelante su nombre y sexo.

6.10 Recuperar los diferentes códigos de revisión (omitiendo duplicados),

6.11 Recuperar el total de ventas de coches, en euros y en pesetas, realizadas en el año 2009

6.12 Recuperar para cada cliente su código de cliente, nombre y número total de revisiones que se le han hecho a todos sus vehículos.

6.13 Recuperar el total de revisiones por código.

6.14 Recuperar el valor medio de los coches vendidos el año pasado.

6.15 Recuperar matrícula y precio de venta de los vehículos que tengan la revisión realizada en el primer semestre de este año.

6.16 Recuperar los nombres de los clientes que hayan comprado coches cuyo valor total sea superior a 30000€.

6.17 Recuperar el nombre de aquellos clientes que han pasado la revisión a su coche y se le ha entregado el vehículo en el primer trimestre del año 2010.

6.18 Recuperar el número de revisiones que se han hecho en total de los códigos AR y RF.

6.19 Averiguar qué porcentaje medio representa la tasa sobre el valor de venta del coche.

6.20 Suponiendo que el beneficio de un coche equivale al 15% de su precio de venta, recuperar para cada coche vendido su precio de venta y el beneficio obtenido.

4 SENTENCIAS DE ACTUALIZACIÓN

Las sentencias de actualización son las encargadas de realizar acciones como añadir, modificar y borrar datos de la base de datos. Los comandos que proporciona SQL para realizar estas acciones son:

Comando	Descripción
INSERT	Este comando agrega uno o más registros a una tabla en una base de datos relacional
UPDATE	Este permite modificar los valores de un conjunto de registros existentes en una tabla
DELETE	Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla

4.1 SENTENCIAS DE AGREGACIÓN

Mediante estas sentencias se añaden uno o más registros a una (y sólo una) tabla en una base de datos relacional. El comando que proporciona SQL para realizar la acción de agregación de datos en las tablas es INSERT INTO.

La sintaxis básica es:

```
INSERT INTO    <tabla> (<campo1>, <campo2>, ... ,<campoN>)
VALUES         (<valor1>, ... , <valorN>)
```

Esta sentencia añade un registro en la tabla <tabla> con el valor1 en el campo1, con el valor2 en el campo2 y así sucesivamente.

Características a tener en cuenta:

- ✓ El número de campos y valores debe ser igual.
- ✓ Si no se especifica valor para una columna, le será asignado el valor por defecto (si está definido). En caso de que no esté definido un valor por defecto se asignará NULL.
- ✓ Los valores especificados deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.
- ✓ Si se agregan registros a una tabla con un campo de tipo autonúmerico, no se debe incluir dicho campo en la sentencia. Si se inserta un valor en un campo de este tipo se escribe el valor, no incrementándose como le corresponde.
- ✓ Acotar entre comillas simples (') los valores literales (cadenas de caracteres)
- ✓ Indicar las fechas en formato mm-dd-aaaa y entre almohadillas (#).

EJEMPLO

Añadir una nueva localización 'Zaragoza' asociada al número de departamento 5.

```
INSERT INTO localizaciones (num_dpto, localizacion)
VALUES (5,'Zaragoza');
```

Para facilitar la inserción de datos en las bases de datos SQL proporciona diversas variantes.

1. Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

   INSERT INTO <tabla>

   VALUES (<valor1>, ... , <valorN>)

2. Inserciones en múltiples filas:

   INSERT INTO <tabla> (<campo1>, <campo2>, ... ,<campoN>)

   VALUES (<valor1>,...,<valorN>), ... ,(<valorO>,..., <valorR>)

   Esta funcionalidad no está soportada por Access.

3. Copia de filas de los datos recuperados mediante una consulta de selección:

   INSERT INTO <tabla1> (<campo1>, <campo2>, ... ,<campoN>)

   SELECT <campo11>, <campo21>, ... ,<campoN1>

   FROM <tabla11>

   WHERE <condiciones>

4. Copia de filas de otras tablas que tienen la misma estructura, es decir, los mismos campos (igual nombre y tipo):

   INSERT INTO <tabla1>

   SELECT <campos>

   FROM <tabla11>

   WHERE <condiciones>

4.2 SENTENCIAS DE ACTUALIZACIÓN

Mediante estas sentencias se actualizan uno o más registros de una tabla en función de un criterio. El comando que proporciona SQL para realizar la acción de actualización es UPDATE.

La sintaxis básica es:

```
UPDATE    <tabla>
SET       Campo1=Valor1, Campo2=Valor2, ... CampoN=ValorN
WHERE     <condiciones>
```

Si en una consulta de actualización suprimimos la cláusula WHERE todos los registros de la tabla señalada serán actualizados.

Una vez que se han actualizado los registros no puede deshacerse la operación. Si se desea saber qué registros se actualizarán, primero hay que examinar los resultados mediante una consulta de selección que utilice el mismo criterio.

También es muy importante realizar una copia de seguridad antes de ejecutar la acción ya que así se podrán recuperar los datos si se produce algún error.

EJEMPLO

Actualizar la localización con el número de departamento 5 por Bilbao .

```
UPDATE localizaciones SET localizacion='Bilbao' WHERE num_dpto=5;
```

4.3 SENTENCIAS DE ELIMINACIÓN

Mediante estas sentencias se eliminan uno o más registros de una tabla en función de un criterio. El comando que proporciona SQL para realizar la acción de actualización es DELETE.

La sintaxis básica es:

```
DELETE
FROM    <tabla>
WHERE   <condiciones>
```

Esta sentencia elimina los registros completos, es decir, no es posible eliminar el contenido de algún campo en concreto.

Una vez que se han eliminado los registros utilizando una sentencia de borrado, no puede deshacer la operación. Si se desea saber qué registros se eliminarán, primero hay que examinar los resultados mediante una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de borrado.

También es muy importante realizar una copia de seguridad antes de ejecutar la acción de borrado ya que así se podrán recuperar los datos si se produce algún error.

EJEMPLO

Eliminar la localización con número de departamento 5.

```
DELETE FROM localizaciones WHERE num_dpto=5;
```

5 SENTENCIAS DE GESTIÓN DE ESTRUCTURA

Las sentencias de gestión de la estructura del sistema de gestión de base de datos son las encargadas modificar la estructura de los objetos de la base de datos.

Estas sentencias son las que forman el lenguaje de definición de datos (DDL).

Existen cuatro operaciones básicas:

Comando	Descripción
CREATE	Este comando crea un objeto dentro de la base de datos
ALTER	Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.
DROP	Este comando elimina un objeto de la base de datos. Se puede combinar con la sentencia ALTER
TRUNCATE	Este comando borra todo el contenido de una tabla

5.1 CREACIÓN DE TABLAS

Mediante estas sentencias se crean tablas nuevas en la base de datos. El comando que proporciona SQL para realizar la acción de creación es CREATE TABLE.

La sintaxis general es:

```
CREATE TABLE    <Tabla> (
        <Campo1>    <Tipo_dato>    [NOT NULL]    [DEFAULT <valor>] ,
        <Campo2>    <Tipo_dato>    [NOT NULL]    [DEFAULT <valor>] ,
        ..... ,
        [CONSTRAINT <Nombre>] [PRIMARY KEY (<Campo1>, <Campo2>, ...)] ,
        [UNIQUE (<Campo1>, <Campo2>, ...) ],
        [FOREIGN KEY (<Campo>) REFERENCES <Tabla (Campo)>
        [ON DELETE CASCADE | SET DEFAULT | SET NULL]
        [ON UPDATE CASCADE | SET DEFAULT | SET NULL] ]
        ) ;
```

Donde:

<Tabla> es el nombre de la nueva tabla que se va a crear.

<Campo1>, <Campo2> son los campos que va a contener la nueva tabla.

<Tipo_dato> es el tipo de dato que van a tener los nuevos campos.

NOT NULL es una cláusula opcional que se añade a cada campo al que no se quiera permitir el valor nulo.

DEFAULT <valor> es una cláusula opcional que se añade a cada campo al que se le quiera especificar el valor por defecto a <valor>.

PRIMARY KEY es la cláusula que permite especificar uno o más campos que constituyen la clave primaria de una tabla. Con la cláusula opcional CONSTRAINT se puede indicar un nombre a la restricción.

UNIQUE es la cláusula que permite especificar uno o más campos que constituyen la clave secundaria de una tabla. También se puede utilizar conjuntamente con la cláusula opcional CONSTRAINT.

FOREIGN KEY es la cláusula que nos permite indicar el campo <Campo> que es clave externa del campo de la tabla especificado en <Tabla (Campo)> mediante la cláusula REFERENCES.

Con la cláusula opcional ON DELETE se indica la acción a realizar CASCADE o SET DEFAULT o SET NULL sobre el campo de clave externa, en caso de que se elimine el registro de referencia. En este caso CASCADE implica borrado del registro, SET DEFAULT fijará al valor por defecto y SET NULL fijará a vacío.

Con la cláusula opcional ON UPDATE se indica la acción a realizar CASCADE o SET DEFAULT o SET NULL sobre el campo de clave externa en caso de que se actualice el registro de referencia. En este caso CASCADE implica actualización del registro al valor de referencia, SET DEFAULT fijará al valor por defecto y SET NULL fijará a vacío.

También se puede utilizar conjuntamente con la cláusula opcional CONSTRAINT.

EJEMPLO

Crear una tabla Bajas que tenga los campos NIF, FechaBaja y FechaAlta

```
CREATE TABLE Bajas (NIF TEXT(9), FechaBaja DATETIME, FechaAlta DATETIME);
```

5.2 MODIFICACIÓN DE TABLAS

Mediante estas sentencias se modifican las estructuras de las tablas en la base de datos. El comando que proporciona SQL para realizar la acción de modificación es ALTER TABLE.

Existen diferentes sintaxis según lo que se quiera hacer:

5.2.1 AÑADIR UN CAMPO

ALTER TABLE <Tabla> ADD [COLUMN] <Campo> <Tipo_dato>;

Donde:

<Tabla> es el nombre de la tabla que se va a modificar.

<Campo> es el nuevo campo que va a contener la tabla <Tabla>.

<Tipo_dato> es el tipo de dato que va a tener el campo <Campo>

La cláusula COLUMN es opcional.

EJEMPLO

Crear un campo Nombre en la tabla Bajas.

```
ALTER TABLE Bajas ADD Nombre TEXT;
```

5.2.2 BORRAR UN CAMPO

ALTER TABLE <Tabla> DROP [COLUMN] <Campo> CASCADE | RESTRICT;

Donde:

<Tabla> es el nombre de la tabla que se va a modificar.

<Campo> es el campo que se va a eliminar de la tabla <Tabla>.

Para eliminar una columna se debe elegir entre CASCADE o RESTRICT.

Si se elige CASCADE todas las restricciones que hagan referencia al campo se eliminarán automáticamente.

Si se elige RESTRICT la sentencia sólo se ejecutará si ninguna restricción hace referencia al campo que se quiere eliminar.

EJEMPLO

Borrar el campo Nombre en la tabla Bajas.

```
ALTER TABLE Bajas DROP Nombre;
```

5.2.3 AÑADIR UN VALOR POR DEFECTO A UN CAMPO

```
ALTER TABLE <Tabla> ALTER [COLUMN] <Campo> SET DEFAULT <Valor>;
```

Donde:

<Tabla> es el nombre de la tabla que se va a modificar.

<Campo> es el nuevo campo que va a contener la tabla <Tabla>.

<Valor> es el nuevo valor por defecto que va a tener el campo <Campo>

No está soportado en Access.

5.2.4 BORRAR UN VALOR POR DEFECTO DE UN CAMPO

```
ALTER TABLE <Tabla> ALTER [COLUMN] <Campo> DROP DEFAULT;
```

Donde:

<Tabla> es el nombre de la tabla que se va a modificar.

<Campo> es el campo del que se va a eliminar el valor por defecto.

No está soportado en Access.

5.2.5 AÑADIR UNA RESTRICCIÓN

ALTER TABLE <Tabla> ADD CONSTRAINT <Nombre_Restricción> <Restricción>;

Donde:

<Tabla> es el nombre de la tabla que se va a modificar.

<Nombre_Restricción> es el nombre de la nueva restricción <Restricción> que va a contener la tabla <Tabla>

EJEMPLO

Modificar la tabla Bajas para que el campo NIF sea clave primaria.

ALTER TABLE Bajas ADD CONSTRAINT Bajas_ClavePrimaria PRIMARY KEY (NIF);

5.2.6 BORRAR UNA RESTRICCIÓN

ALTER TABLE <Tabla> DROP CONSTRAINT <Nombre_Restricción> CASCADE | RESTRICT;

Donde:

<Tabla> es el nombre de la tabla que se va a modificar.

<Nombre_Restricción> es el nombre de la restricción que se va a eliminar.

Para poder eliminar una restricción es preciso haberle dado un nombre cuando fue especificada.

Si se elige CASCADE todas las restricciones que hagan referencia a la restricción se eliminarán automáticamente.

Si se elige RESTRICT la sentencia sólo se ejecutará si ninguna restricción hace referencia a la restricción que se quiere eliminar.

EJEMPLO

Modificar la tabla Bajas para que el campo NIF no sea clave primaria.

ALTER TABLE Bajas DROP CONSTRAINT Bajas_ClavePrimaria CASCADE;

5.3 BORRADO DE TABLA

Mediante esta sentencia se borra una tabla en la base de datos. El comando que proporciona SQL para realizar la acción de borrado es DROP TABLE.

La sintaxis básica es:

DROP TABLE <Tabla> CASCADE | RESTRICT;

Si se escoge la opción RESTRICT se eliminará la tabla si no se hace referencia a ella en ninguna restricción, como por ejemplo en las definiciones de clave externa de otra relación.

Con la opción CASCADE se eliminará la tabla y todas las restricciones que hagan referencia a la misma.

Hay que tener cuidado con esta acción porque no es reversible.

EJEMPLO

Eliminar la tabla Bajas.

DROP TABLE Bajas CASCADE;

5.4 BORRADO DEL CONTENIDO DE UNA TABLA

Mediante esta sentencia se borra el contenido de una tabla en la base de datos. El comando que proporciona SQL para realizar la acción de creación es TRUNCATE TABLE.

La sintaxis básica es:

```
TRUNCATE TABLE <Tabla>;
```

Hay que tener cuidado con esta acción porque no es reversible.

No está soportado en Access.

Ejercicios

7.1 Recuperar el número de conductores que han nacido en Pamplona.

7.2 Recuperar los nombres de las calles que son avenidas o plazas.

7.3 Recuperar la media de plazas de los autobuses de la empresa de transportes.

7.4 Recuperar el número de rutas que pasan por Olite y Tudela.

7.5 Se quiere organizar una excursión extraordinaria. Recuperar el total de plazas disponibles de la empresa de transportes. No tener en cuenta si el autobús está disponible o no.

7.6 Recuperar el código de ruta y el precio de billete de la ruta urbana que más paradas tiene.

7.7 Recuperar la matrícula de los autobuses que han realizado rutas urbanas.

7.8 Recuperar los datos del conductor cuyo nombre se introducirá como parámetro de entrada.

7.9 Recuperar todos los datos de las rutas realizadas el 5 de Septiembre de 2005.

7.10 Recuperar para las rutas interurbanas, por cuantos pueblos pasa cada ruta.

7.11 Recuperar el código de ruta y el precio de billete de la ruta interurbana que menos kilómetros recorre.

7.12 Recuperar para las rutas interurbanas el precio del billete por kilómetro.

7.13 Recuperar para cada ruta urbana, el número total de calles por las que pasa.

7.14 Se ha dado un parte de siniestro y se quiere recuperar las matrículas que contienen una C.

7.15 Recuperar los nombres y direcciones de los conductores que no han nacido en Pamplona.

7.16 Debido al incremento del precio de la gasolina se dispone el incrementar los precios de los billetes un 10%.

7.17 Recuperar los nombres de los pueblos que intervienen en rutas interurbanas con un recorrido superior a 100 Km.

7.18 Recuperar el número de rutas agrupando por urbanas e interurbanas.

7.19 Recuperar los datos personales de los conductores que realizan rutas a Tudela.

7.20 Del nombre de conductor que se introducirá por teclado se quiere saber todos los vehículos que ha utilizado (matrícula) y el código de ruta.