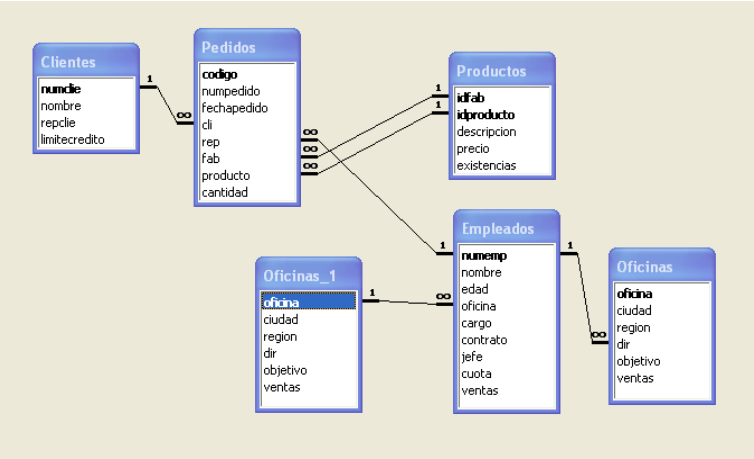
TEMA 7: LENGUAJE SQL: LA SENTENCIA SELECT

0.- BASE DE DATOS PARA LOS EJEMPLOS.

Para el desarrollo de este tema se utilizará la siguiente base de datos. Aparece implementada en el archivo datos.accdb de Microsoft Access 2007. (NOTA: La tabla Oficinas_1 y Oficinas en el dibujo representan la misma tabla.)



Las tablas de la figura tendrán el siguiente contenido:

Productos				
idfab	idproducto	descripcion	precio	existencias
aci	41001	Mesa estudio	34,95 €	277
aci	41002	Mesa de ordenador	25,90 €	107
aci	41003	Estantería (dos estantes)	36,70 €	207
aci	41004	Estantería simple + Dos puertas	58,95 €	139
aci	41005	Estantería (5 estantes)	55,00 €	37
aci	41006	Estantería (tres estantes) + Dos puertas	80,95 €	25
aci	41007	Armario (dos puertas)	90,95 €	28
bic	41003	Mesa despacho con 3 cajones	89,00 €	3
bic	41089	Carro informático	84,95 €	78
bic	41672	Estantería (3 estantes)	42,70 €	0
fea	11200	Silla "Escocia"	56,95 €	115
fea	11400	Sillón "Confidence"	59,95 €	15
imm	77300	Estantería "BIBLIOS"	29,95 €	25
imm	77500	Estantería "BASICA"	19,95 €	29
imm	77900	Armario "BRISE"	89,95 €	0
imm	88700	Armario "MAX"	121,00 €	223
imm	88701	Armario puertas correderas	74,95 €	0
imm	88702	Zapatero	52,95 €	24
qsa	00147	Lámpara serie "ARABIA"	189,00 €	32
qsa	00148	Lámpara serie "TUBE"	41,95 €	115
qsa	00149	Lámpara serie "MEDINA"	135,00 €	37
rei	20441	Video portero de 2 hilos	129,00 €	14
rei	20442	Base multimedia con protección	18,95 €	12
rei	20443	Programador digital	11,95 €	12
rei	20444	Kit 3 enchufes con mando a distancia	19,95 €	210

Pedidos							
codigo	numpedido	fechapedido	cli	rep	fab	producto	cantidad
1	110036	02/01/2005	2107	109	aci	41001	10
2	110036	02/01/2005	2107	109	aci	41002	5
3	112963	20/02/2005	2103	105	bic	41003	3
4	112965	20/02/2005	2105	102	rei	20441	3
5	112965	20/02/2005	2105	102	rei	20442	5
6	112965	20/02/2005	2105	102	rei	20443	5
7	113001	25/02/2005	2120	105	imm	88700	4
8	113003	02/03/2005	2122	107	fea	11200	10
9	113003	02/03/2005	2122	107	fea	11400	12
10	113003	02/03/2005	2122	107	imm	77300	4
11	113005	07/03/2005	2113	101	qsa	00147	2
12	113006	08/03/2005	2113	101	rei	20443	3
13	113010	12/04/2005	2102	101	imm	88700	2
14	113020	15/04/2005	2108	103	bic	41089	15
15	113025	15/04/2005	2110	108	imm	88702	4
16	113028	18/04/2005	2112	102	rei	20441	5
17	114010	20/05/2005	2115	108	qsa	00147	10
18	114010	20/05/2005	2115	108	qsa	00148	15
19	114010	20/05/2005	2115	108	qsa	00149	5
20	114015	22/06/2005	2120	105	aci	41006	30

Clientes			
numclie	nombre	repclie	limitecredito
2101	Luis García Antón	106	65.000,00 €
2102	Álvaro Rodríguez	101	65.000,00 €
2103	Jaime Llorens	105	50.000,00 €
2104	Antonio Canales	101	45.000,00 €
2105	Rafael Serrano	102	85.000,00 €
2106	Juan Suárez	110	35.000,00 €
2107	Julian López	109	50.000,00 €
2108	Cristóbal García	103	20.000,00 €
2109	Maria Silva	103	20.000,00 €
2110	Cristina Bulini	108	60.000,00 €
2111	Vicente Martínez	104	35.000,00 €
2112	Juan Malo	102	40.000,00 €
2113	Vicente Ríos	101	55.000,00 €
2114	Ana López	106	60.000,00 €
2115	Antonio Fernández	108	25.000,00 €
2116	Alejandro Ruiz	109	30.000,00 €
2117	Francisco Gutiérrez	110	40.000,00 €
2118	Inmaculada Santos	103	65.000,00 €
2119	Mercedes Castro	102	30.000,00 €
2120	Julio Torres	105	40.000,00 €
2121	Lucia Campos	102	40.000,00 €
2122	Paula Álvarez	107	20.000,00 €
2123	Miguel Ángel Ruiz	103	20.000,00 €
2124	José Antonio Flores	106	60.000,00 €

Empleados								
numemp	nombre	edad	oficina	cargo	contrato	jefe	cuota	ventas
101	Antonio Viguer	45	12	representante ventas	20/10/1986	104	300.000,00 €	305.673,00 €
102	Jesús Torres	48	21	representante ventas	10/12/1986	108	350.000,00 €	474.050,00 €
103	Juan Rovira	29	12	representante ventas	01/03/1987	104	275.000,00 €	286.775,00 €
104	José González	33	12	director ventas	19/05/1987	106	200.000,00 €	142.594,00 €
105	Vicente Pérez	37	13	representante ventas	12/02/1988	104	350.000,00 €	367.911,00 €
106	Luis Díaz	52	11	director gene-ral	14/06/1988		275.000,00 €	299.912,00 €
107	Jorge Gutiérrez	49	22	representante ventas	14/11/1988	108	300.000,00 €	186.042,00 €
108	Ana Fernández	62	21	director ventas	12/10/1989	106	350.000,00 €	361.865,00 €
109	Maria García	31	11	representante ventas	12/10/1999	106	300.000,00 €	392.725,00 €
110	Juan Gómez	41		representante ventas	10/01/1990	101	0,00 €	75.985,00 €

Oficinas					
oficina	ciudad	region	dir	objetivo	ventas
11	Valencia	este	106	575.000,00 €	692.637,00 €
12	Alicante	este	104	800.000,00 €	735.042,00 €
13	Castellón	este	105	350.000,00 €	367.911,00 €
21	Badajoz	oeste	108	725.000,00 €	835.915,00 €
22	La Coruña	oeste	108	300.000,00 €	184.042,00 €
23	Madrid	centro	108	0,00 €	0,00 €
24	Madrid	centro	108	250.000,00 €	150.000,00 €
26	Pamplona	norte		0,00 €	0,00 €
28	Valencia	este		900.000,00 €	0,00 €

1.- ¿QUÉ ES EL SQL?

El **SQL** (Structured query language), **lenguaje de consulta estructurado**, es un lenguaje surgido de un proyecto de investigación de IBM para el acceso a bases de datos relacionales. Actualmente se ha convertido en un **estándar** de lenguaje de bases de datos, y la mayoría de los sistemas de bases de datos lo soportan, desde sistemas para ordenadores personales, hasta grandes ordenadores.
Por supuesto, a partir del estándar cada sistema ha desarrollado su propio SQL (SQL de Access, SQL de SQL Server, SQL de Oracle, SQL de mySQL…) que puede variar de un sistema a otro, pero con cambios que no suponen ninguna complicación para alguien que conozca un SQL concreto.
Como su nombre indica, el SQL nos **permite** realizar **consultas a la base de datos**. Pero el nombre se queda corto ya que SQL además realiza funciones de **definición, control y gestión de la base de datos.** Las sentencias SQL se clasifican según su finalidad dando origen a tres 'lenguajes' o mejor dicho sublenguajes:

- El **DDL** (Data Description Language), **lenguaje de definición** de datos, incluye órdenes para definir, modificar o borrar las tablas en las que se almacenan los datos y de las relaciones entre estas. (Es el que más varia de un sistema a otro).
- El **DML** (Data Manipulation Language), **lenguaje de manipulación** de datos, nos permite recuperar los datos almacenados en la base de datos y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo, suprimiendo o modificando datos.
- El **DCL** (Data Control Language), **lenguaje de control** de datos, contiene elementos útiles para trabajar en un entorno multiusuario, en el que es importante la protección de los datos, la seguridad de las tablas y el establecimiento de restricciones en el acceso, así como elementos para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieran unos con otros.

2.- CARACTERÍSTICAS DEL LENGUAJE

Una sentencia SQL es como una **frase** (escrita en **inglés**) con la que decimos **lo que queremos obtener y de donde obtenerlo**. Todas las sentencias empiezan con un **verbo** (palabra reservada que indica la acción a realizar), seguido del resto de **cláusulas**, algunas **obligatorias** y otras **opcionales** que completan la frase. Todas las sentencias siguen una **sintaxis** para que se puedan ejecutar correctamente. Para describir esa sintaxis utilizaremos la siguiente notación sintáctica.

SELECT [**ALL** | **DISTINCT**] *nbcolumna* {[, *nbcolumna*]}
FROM *expression-tabla*
[**WHERE** *condición-de-búsqueda*]

Las palabras que aparecen en mayúsculas son palabras reservadas se tienen que poner tal cual y no se pueden utilizar para otro fin, por ejemplo, en el diagrama de la figura tenemos las palabras reservadas **SELECT**, **ALL**, **DISTINCT**, **FROM**, **WHERE**.

Las palabras en minúsculas son variables que el usuario deberá sustituir por un dato concreto. En el ejemplo tenemos *nbcolumna*, *expresion-tabla* y *condicion-de-busqueda*.

Lo opcional se encierra entre corchetes.

Las distintas alternativas se separan por una barra vertical. La alternativa por defecto se subraya.

Lo que se puede repetir de cero a varias veces se encierra entre llaves.

¿Cómo se interpretaría la notación sintáctica de la figura?

Hay que empezar por la palabra **SELECT**, después puedes poner **ALL** o bien **DISTINCT** o nada, a continuación un nombre de columna, o varios separados por comas, a continuación la palabra **FROM** y una expresión-tabla, y por último de forma opcional puedes incluir la cláusula **WHERE** con una condición-de-búsqueda.

EJEMPLOS

SELECT ALL *col1, col2, col3* **FROM** *mitabla*
SELECT *col1, col2, col3* **FROM** *mitabla*
SELECT DISTINCT *col1* **FROM** *mitabla*
SELECT *col1, col2* **FROM** *mitabla* **WHERE** *col2 = 0*

Todas estas sentencias se podrían escribir y no darían lugar a errores sintácticos.

Cuando una palabra opcional está **subrayada**, esto indica que ese es el **valor por defecto** (el valor que se asume si no se pone nada). En el ejemplo anterior las dos primeras sentencias son equivalentes (en la notación sintáctica **ALL** aparece subrayada).

3.- LENGUAJE DE MANIPULACIÓN DE DATOS (DML).

El **DML** (Data Manipulation Language), **lenguaje de manipulación** de datos, nos permite recuperar los datos almacenados en la base de datos (hacer consultas) y también incluye órdenes para permitir al usuario actualizar la base de datos añadiendo nuevos datos (insertando registros), suprimiendo datos antiguos (eliminando registros) o modificando datos previamente almacenados (modificando registros).

Las sentencias que constituyen el **DML** son:

- **SELECT** para realizar consultas (recuperar datos).
- **INSERT** para añadir registros a una tabla.
- **UPDATE** para modificar los valores de un registro.
- **DELETE** para eliminar uno o varios registros de una tabla.

En este primer tema de SQL nos centraremos en el estudio de la sentencia SELECT.

4.- LAS CONSULTAS: INTRODUCCIÓN.

Las consultas son el corazón de SQL, de hecho, existen personas que sólo utilizan el lenguaje SQL para recuperar datos de una base de datos.
La sentencia **SELECT** recupera datos de una base de datos y los devuelve en forma de resultados de la consulta.
Su sintaxis completa es:

```
SELECT [ALL|DISTINCT] *|(item-seleccionado [AS aliasitem] {,item-seleccionado [AS aliasitem]})
FROM especificación-tabla[{,especificación-tabla}]
[
    WHERE condición-de-búsqueda
    GROUP BY columna-de agrupación [,{columna-de-agrupación}]
    HAVING condición-de-búsqueda
    ORDER BY especificación-de-ordenación [{,especificación-de-ordenación}]
]
```

- La cláusula **SELECT** lista los datos. Los ítems seleccionados pueden ser **columnas de la base de datos** o **columnas a calcular** por SQL cuando efectúe la consulta. Si se pone un **asterisco**, se recuperarán todas las columnas de la tabla o tablas que se especifican en la cláusula **FROM**. Al incluir la cláusula **DISTINCT** en la **SELECT**, **se eliminan del resultado las repeticiones de filas**. Si por el contrario queremos que aparezcan todas las filas incluidas las duplicadas, podemos incluir la cláusula **ALL** o nada, ya que **ALL** es el valor que SQL asume por defecto.
- La cláusula **FROM** lista las tablas que contienen los datos a recuperar por la consulta.
- La cláusula **WHERE** dice a SQL que incluya sólo ciertas filas de datos en los resultados de la consulta. Se utiliza una condición de búsqueda para especificar las filas deseadas.
- La cláusula **GROUP BY** especifica una consulta sumaria. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y luego produce una fila sumaria de resultados para cada grupo.
- La cláusula **HAVING** dice a SQL que incluya sólo ciertos grupos producidos por la cláusula **GROUP BY** en los resultados de la consulta.
- La cláusula **ORDER BY** ordena los resultados de la consulta en base a los datos de una o más columnas. Si se omite, los resultados de la consulta no aparecen ordenados.

El resultado de una consulta SQL **es siempre una tabla de datos**, semejante a las tablas de la base de datos, aunque conste de una sola fila y columna.

5.- OPERADORES QUE SE VINCULAN A LA CLÁUSULA WHERE

Operadores de relación: **<, >, =, <>, <=, >=**
Operador BETWEEN Vi AND Vf ; donde Vi representa el valor inicial de un rango y Vf el valor final.
Operador LIKE sirve para comparar valores que se asemejan a un patrón. El patrón debe construirse empleando el carácter comodín %, que representa a uno o varios caracteres.
Operador IN sirve para preguntar si un valor se corresponde con alguno de los valores de un conjunto.
Operador IS NULL sirve para preguntar si un atributo toma o no el valor nulo. (El valor nulo lo toma un atributo, cuando no se le ha dado ningún valor).
Operadores lógicos: **AND**, **OR** y **NOT**.

6.- ESTUDIO DETALLADO DE LA SENTENCIA SELECT

A) Selección de columnas

La lista de **columnas** que queremos que **aparezcan** en el **resultado** es lo que llamamos **lista de selección** y se especifica delante de la cláusula **FROM**. En el diagrama aparecen con el nombre de ***item-seleccionado***.

- **Utilización del ***

  Se utiliza el asterisco * en la lista de selección para indicar **'todas las columnas de la tabla'**.
  Tiene dos **ventajas**:
  a) Evitar nombrar las columnas una a una (es más corto).
  b) Si añadimos una columna nueva en la tabla, esta nueva columna saldrá sin tener que modificar la consulta.

  Se puede combinar el * con el nombre de una tabla (ej. oficinas.*), pero esto se suele utilizar cuando el origen de la consulta son dos tablas.

EJEMPLOS

```
SELECT * FROM Oficinas
```

o bien

```
SELECT Oficinas.* FROM Oficinas
```

Lista todos los datos de las oficinas

- **Columnas de la tabla origen**
  Las columnas se pueden especificar mediante su **nombre simple** (nbcol) o su **nombre cualificado** (nbtabla.nbcol), el nombre de la columna precedido del nombre de la tabla que contiene la columna y separados por un punto).
  El nombre cualificado se puede emplear siempre que queramos y es obligatorio en algunos casos que veremos más adelante.

EJEMPLOS

```
SELECT nombre, oficina, contrato
FROM Empleados
```

o bien

```
SELECT Empleados.nombre, Empleados.oficina,
       Empleados.contrato
FROM Empleados
```

Lista el nombre, oficina, y fecha de contrato de todos los empleados.

```
SELECT idfab, idproducto, descripcion, precio
FROM Productos
```

Lista el fabricante, el identificador de producto, la descripción y el precio de todos los productos.

- **Alias de columna**.
  Cuando se visualiza el resultado de la consulta, normalmente las columnas toman el nombre que tiene la columna en la tabla, si queremos cambiar ese nombre lo podemos hacer definiendo un alias de columna mediante la cláusula **AS** será el nombre que aparecerá como **título de la columna.**

EJEMPLO

SELECT idfab AS fabricante, idproducto, descripcion **FROM Productos**	Como título de la primera columna aparecerá fabricante en vez de idfab

- **Columnas calculadas**.

  Además de las columnas que provienen directamente de la tabla origen, una consulta SQL puede incluir **columnas calculadas** cuyos valores se calculan a partir de los valores de los datos almacenados. Para solicitar una columna calculada, se especifica en la lista de selección una **expresión** en vez de un nombre de columna. La expresión puede contener sumas, restas, multiplicaciones, divisiones, paréntesis, funciones predefinidas, etc.

EJEMPLOS

SELECT ciudad, región, **(ventas-objetivo) AS superavit** **FROM Oficinas**	Lista la ciudad, región y el superavit de cada oficina.
SELECT idfab, idproducto, descripcion, **(existencias * precio) AS valoracion** **FROM Productos**	De cada producto obtiene su fabricante, idproducto, su descripción y el valor del inventario
SELECT nombre, MONTH(contrato), YEAR(contrato) **FROM Empleados**	Lista el nombre, mes y año del contrato de cada vendedor. La función MONTH() devuelve el mes de una fecha La función YEAR() devuelve el año de una fecha
SELECT oficina, 'tiene ventas de ', ventas **FROM Oficinas**	Listar las ventas en cada oficina con el formato: 22 tiene ventas de 186,042.00 ptas

B) La tabla origen -FROM-

Con la cláusula **FROM** indicamos **en qué tabla o tablas** hay que **buscar la información**.

FROM *especificación-tabla* [**AS** *aliastabla*]

- **Aliastabla** es un nombre de **alias**, es como un **segundo nombre** que asignamos a la **tabla**, si en una consulta definimos un alias para la tabla, esta se deberá nombrar utilizando ese nombre y no su nombre real, además **ese nombre sólo es válido en la consulta** donde se define. El alias se suele emplear en consultas basadas en más de una tabla. La palabra **AS** que se puede poner delante del nombre de alias es opcional y es el valor por defecto por lo que no tienen ningún efecto.

EJEMPLO

SELECT .....FROM Oficinas ofi equivalente a **SELECT .....FROM Oficinas AS ofi**	esta sentencia me indica que se van a buscar los datos en la tabla *Oficinas* que queda renombrada en esta consulta con *ofi*.

C) Selección de filas

A continuación veremos las cláusulas que nos permiten indicar **qué filas** queremos **visualizar**.

SELECT [**ALL**|**DISTINCT**] …

…

WHERE *condición-de-búsqueda*

- **Las cláusulas DISTINCT / ALL**
  Al incluir la cláusula **DISTINCT** en la SELECT, se **eliminan** del resultado las **repeticiones** de filas. Si por el contrario queremos que aparezcan todas las filas **incluidas las duplicadas**, podemos incluir la cláusula **ALL** o nada, ya que ALL es el valor que SQL asume por defecto.

EJEMPLO

Queremos saber los códigos de los directores de oficina.

SELECT dir FROM Oficinas **SELECT ALL dir FROM Oficinas**	Lista los códigos de los directores de las oficinas. El director 108 aparece en cuatro oficinas, por lo tanto aparecerá cuatro veces en el resultado de la consulta.
SELECT DISTINCT dir FROM Oficinas	En este caso el valor 108 aparecerá una sola vez ya que le decimos que liste los distintos valores de directores.

- **La cláusula WHERE**

La cláusula **WHERE** selecciona únicamente las **filas** que **cumplan** la **condición de búsqueda** especificada.
En la consulta sólo aparecerán las filas para las cuales la condición es verdadera (TRUE), los valores nulos (NULL) no se incluyen por lo tanto en las filas del resultado. La **condición de búsqueda** puede ser cualquier **condición válida** o **combinación de condiciones** utilizando los operadores **NOT** (no) **AND** (y) y **OR** (ó).

EJEMPLOS

`SELECT nombre`
`FROM empleados`
`WHERE oficina = 12`
	Lista el nombre de los empleados de la oficina 12.

`SELECT nombre`
`FROM empleados`
`WHERE oficina = 12 AND edad > 30`
	Lista el nombre de los empleados de la oficina 12 que tengan más de 30 años. (oficina igual a 12 y edad mayor que 30)

Condiciones de búsqueda

Las **condiciones de búsqueda** son las condiciones que pueden aparecer en la cláusula **WHERE**.
En SQL tenemos cinco condiciones básicas: test de comparación, test de rango, test de pertenencia a un conjunto, test de valor nulo y test de correspondencia con patrón.

a) El test de comparación: compara el valor de una expresión con el valor de otra.
La sintaxis es la siguiente:

expresion1 = | <> | < | <= | > | >= *expresion2*

EJEMPLOS

`SELECT numemp, nombre`
`FROM Empleados`
`WHERE ventas > cuota`
	Lista los empleados cuyas ventas superan su cuota

`SELECT numemp, nombre`
`FROM Empleados`
`WHERE contrato < #01/01/1988#`
	Lista los empleados contratados antes del año 88 (cuya fecha de contrato sea anterior al 1 de enero de 1988).

`SELECT numemp, nombre`
`FROM Empleados`
`WHERE YEAR(contrato) < 1988`
	Este ejemplo obtiene lo mismo que el anterior pero utiliza la función year(). Obtiene los empleados cuyo año de la fecha de contrato sea menor que 1988.

`SELECT oficina`
`FROM Oficinas`
`WHERE ventas < objetivo * 0.8`
	Lista las oficinas cuyas ventas estén por debajo del 80% de su objetivo.
	Hay que utilizar siempre el punto decimal aunque tengamos definida la coma como separador de decimales.

`SELECT oficina`
`FROM Oficinas`
`WHERE dir = 108`
	Lista las oficinas dirigidas por el empleado 108.

b) El test de rango (BETWEEN): examina si el **valor** de la expresión está **comprendido entre dos valores.**
Tiene la siguiente sintaxis:

Expresión [NOT] BETWEEN *exp1* AND *exp2*

EJEMPLOS

SELECT numemp, nombre FROM Empleados WHERE ventas BETWEEN 100000 AND 500000	Lista los empleados cuyas ventas estén comprendidas entre 100.000 y 500.00
SELECT numemp, nombre FROM Empleados WHERE (ventas >= 100000) AND (ventas <= 500000)	Obtenemos lo mismo que en el ejemplo anterior. Los paréntesis son opcionales.

c) El test de pertenencia a conjunto (IN): examina si el **valor** de la expresión es uno de los valores **incluidos en** la **lista de valores**.
Tiene la siguiente sintaxis:

Expresión [NOT] IN (valor {,valor})

EJEMPLOS

SELECT numemp, nombre, oficina FROM Empleados WHERE oficina IN (12,14,16)	Lista los empleados de las oficinas 12, 14 y 16
SELECT numemp, nombre FROM Empleados WHERE (oficina = 12) OR (oficina = 14) OR (oficina = 16)	Obtenemos lo mismo que en el ejemplo anterior. Los paréntesis son opcionales.

d) El test de valor nulo (IS NULL)
Una condición de selección puede dar como resultado el valor verdadero TRUE, falso FALSE o nulo NULL. Cuando una **columna** que interviene en una condición de selección **contiene** el **valor nulo,** el **resultado** de la condición no es verdadero ni falso, sino **nulo**, **sea cual sea el test** que se haya utilizado. Por eso si queremos listar las filas que tienen valor en una determinada columna, no podemos utilizar el test de comparación, la condición oficina = NULL devuelve el valor nulo sea cual sea el valor contenido en oficina. Si queremos preguntar si una columna contiene el valor nulo debemos utilizar un **test especial**, el test de valor nulo.
Tiene la siguiente sintaxis:

nbcolumna IS [NOT] NULL

EJEMPLOS

SELECT oficina, ciudad	Lista las oficinas que no tienen director.

FROM Oficinas WHERE dir IS NULL	
SELECT numemp, nombre FROM Empleados WHERE oficina IS NOT NULL	Lista los empleados asignados a alguna oficina (los que tienen un valor en la columna oficina).

e) El test de correspondencia con patrón (LIKE): se utiliza cuando queremos **utilizar caracteres comodines** para formar el valor con el comparar.
Tiene la siguiente sintaxis:

nbcolumna **[NOT] LIKE** *patrón*

Los comodines más usados son los siguientes:

? ó _ representa un carácter cualquiera.
*** ó %** representa cero o más caracteres.
representa un dígito cualquiera (0-9).

EJEMPLOS

SELECT numemp, nombre FROM Empleados WHERE nombre LIKE 'Luis*'	Lista los empleados cuyo nombre empiece por Luis (Luis seguido de cero o más caracteres).
SELECT numemp, nombre FROM Empleados WHERE nombre LIKE '*Luis*'	Lista los empleados cuyo nombre contiene Luis, en este caso también saldría los empleados José Luis (cero o más caracteres seguidos de LUIS y seguido de cero o más caracteres).
SELECT numemp, nombre FROM Empleados WHERE nombre LIKE '??a*'	Lista los empleados cuyo nombre contenga una a como tercera letra (dos caracteres, la letra a, y cero o más caracteres.

D) Ordenación de las filas - ORDER BY -

ORDER BY *nbcolumna* [**ASC** | **DESC**] { , *nbcolumna* [**ASC** | **DESC**] }

Para **ordenar** las **filas** del resultado de la consulta, tenemos la cláusula **ORDER BY**.
Con esta cláusula se altera el orden de visualización de las filas de la tabla pero en ningún caso se modifica el orden de las filas dentro de la tabla. La tabla no se modifica.
Podemos indicar la columna por la que queremos ordenar utilizando su **nombre de columna** (nbcolumna) o utilizando su **número de orden** que ocupa en la lista de selección (Nºcolumna).

EJEMPLOS

```
SELECT nombre, oficina, contrato
FROM Empleados
ORDER BY oficina
```

es equivalente a

```
SELECT nombre, oficina, contrato
FROM Empleados
ORDER BY 2
```

Por defecto el **orden** será **ascendente** (**ASC**) (de menor a mayor si el campo es numérico, por orden alfabético si el campo es de tipo texto, de anterior a posterior si el campo es de tipo fecha/hora, etc...

EJEMPLOS

`SELECT nombre, numemp, oficinarep`
`FROM Empleados`
`ORDER BY nombre`
	Obtiene un listado alfabético de los empleados.

`SELECT nombre, numemp, contrato`
`FROM Empleados`
`ORDER BY contrato`
	Obtiene un listado de los empleados por orden de antiguedad en la empresa (los de más antiguedad aparecen primero).

`SELECT nombre, numemp,ventas`
`FROM Empleados`
`ORDER BY ventas`
	Obtiene un listado de los empleados ordenados por volúmen de ventas sacándo los de menores ventas primero.

Si queremos podemos alterar ese orden utilizando la cláusula **DESC** (DESCendente), en este caso el orden será el inverso al ASC.

EJEMPLOS

`SELECT nombre, numemp, contrato`
`FROM Empleados`
`ORDER BY contrato DESC`
	Obtiene un listado de los empleados por orden de antiguedad en la empresa empezando por los más recientemente incorporados.

`SELECT nombre, numemp, ventas`
`FROM Empleados`
`ORDER BY ventas DESC`
	Obtiene un listado de los empleados ordenados por volúmen de ventas sacando primero los de mayores ventas.

También podemos **ordenar** por **varias columnas**, en este caso se indican las columnas separadas por comas. Se ordenan las filas por la primera columna de ordenación, para un mismo valor de la primera columna, se ordenan por la segunda columna, y así sucesivamente.
La cláusula **DESC** o **ASC** se puede indicar para cada columna y así utilizar una ordenación distinta para cada columna. Por ejemplo ascendente por la primera columna y dentro de la primera columna, descendente por la segunda columna.

EJEMPLOS

`SELECT region, ciudad, ventas`
`FROM Oficinas`
`ORDER BY region, ciudad`
	Muestra las ventas de cada oficina , ordenadas por orden alfabético de región y dentro de cada región por ciudad.

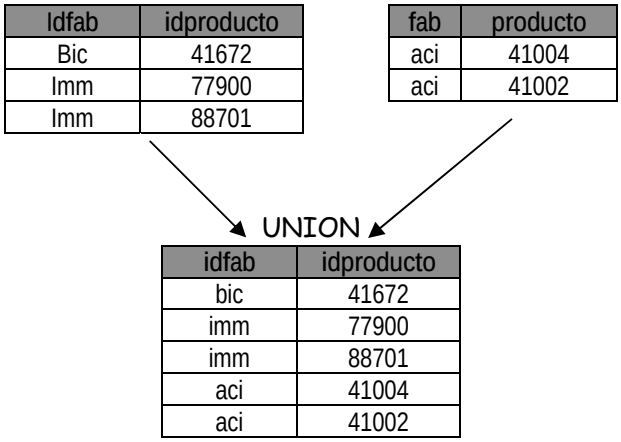
`SELECT region, ciudad, (ventas - objetivo) AS superavit`
`FROM Oficinas`
`ORDER BY region, 3 DESC`
	Lista las oficinas clasificadas por región y dentro de cada región por superavit de modo que las de mayor superavit aparezcan las primeras.

7.- LAS CONSULTAS MULTITABLA

Las **consultas multitabla** están **basadas** en **más de una tabla**. Existen dos grupos de consultas multitabla que son la **unión** de tablas y la **composición** de tablas.

a) La unión de tablas

Esta operación se utiliza cuando tenemos **dos tablas** con las **mismas columnas** y queremos obtener una **nueva tabla** con las **filas de la primera y las filas de la segunda**. En este caso la tabla resultante tiene las mismas columnas que la primera tabla (que son las mismas que las de la segunda tabla).
Por ejemplo tenemos una tabla de libros nuevos y una tabla de libros antiguos y queremos una lista con todos los libros que tenemos. En este caso las dos tablas tienen las mismas columnas, lo único que varía son las filas, además queremos obtener una lista de libros (las columnas de una de las tablas) con las filas que están tanto en libros nuevos como las que están en libros antiguos, en este caso utilizaremos este tipo de operación.
Por ejemplo queremos en un sólo listado los productos cuyas existencias sean iguales a cero y también los productos que aparecen en pedidos de enero de 2005. En este caso tenemos unos productos en la tabla de productos y los otros en la tabla de pedidos, las tablas no tienen las mismas columnas no se puede hacer una unión de ellas pero lo que interesa realmente es el identificador del producto (idfab,idproducto), luego por una parte sacamos los códigos de los productos con existencias cero (con una consulta), por otra parte los códigos de los productos que aparecen en pedidos de enero de 2005 (con otra consulta), y luego unimos estas dos tablas lógicas.
El operador que permite realizar esta operación es el operador **UNION**.



El operador UNION

El operador **UNION** sirve para obtener a partir de dos **tablas** con las **mismas columnas**, una nueva tabla con las **filas** de la **primera y** las **filas** de la **segunda**.
La sintaxis es la siguiente:

```
CONSULTA
  UNION
CONSULTA
```

Consulta puede ser un **nombre de tabla**, un **nombre de consulta** (en estos dos casos el nombre debe estar precedido de la palabra **TABLE**), o una **sentencia SELECT** completa (en este caso no se puede poner **TABLE**). La sentencia **SELECT** puede ser cualquier sentencia **SELECT** con la única restricción de que no puede contener la cláusula **ORDER BY**.
Después de la primera consulta viene la palabra **UNION** y a continuación la segunda consulta. La segunda consulta sigue las mismas reglas que la primera consulta.
Deben cumplirse las siguientes condiciones:

- Las dos consultas deben tener el **mismo número** de **columnas** pero las columnas pueden **llamarse** de **diferente** forma y ser de **tipos** de datos **distintos**.
- Las **columnas** del **resultado** se **llaman** como las de la **primera consulta**.
- Por **defecto** la unión **no incluye filas repetidas**, si alguna fila está en las dos tablas, sólo aparece una vez en el resultado. Si **queremos** que aparezcan todas las filas incluso las **repeticiones** de **filas**, incluimos la palabra **ALL** (*todo* en inglés). El empleo de **ALL** tienen una **ventaja**, la consulta **se ejecutará más rapidamente**. Puede que la diferencia no se note con tablas pequeñas, pero si tenemos tablas con muchos registros (filas) la diferencia puede ser notable.
- Se puede **unir más** de **dos** tablas, para ello después de la segunda consulta **repetimos** la palabra **UNION** ... y así sucesivamente.
- También podemos indicar que queremos el **resultado ordenado** por algún criterio, en este caso se incluye la cláusula **ORDER BY** que ya vimos en el tema anterior. La cláusula **ORDER BY** se escribe **después** de la **última consulta**, al final de la sentencia; para indicar las **columnas de ordenación** podemos utilizar su **número de orden** o el **nombre de la columna**, en este último caso se deben de utilizar los nombres de columna de la **primera consulta** ya que son los que se van a utilizar para nombrar las columnas del resultado.

EJEMPLO

```
SELECT idfab, idproducto
FROM Productos
WHERE existencias = 0

    UNION ALL

SELECT fab, producto
FROM Pedidos
WHERE year(fechapedido) = 2005
AND month(fechapedido) = 1

ORDER BY idproducto
```

Obtiene los códigos de los productos que tienen existencias iguales a cero o que aparezcan en pedidos de enero del 2005.

Se ha incluido la cláusula **ALL** porque no nos importa que salgan filas repetidas.
Se ha incluido **ORDER BY** para que el resultado salga ordenado por idproducto, observar que hemos utilizado el nombre de la columna de la primera **SELECT**, también podíamos haber puesto **ORDER BY 2** pero no **ORDER BY producto** (es el nombre de la columna de la segunda tabla).

B) La composición de tablas

La composición de tablas consiste en **concatenar** filas de una tabla con filas de otra. En este caso obtenemos una tabla con las **columnas** de la **primera tabla unidas** a las **columnas** de la **segunda tabla**, y las filas de la tabla resultante son **concatenaciones** de **filas** de la **primera tabla con** filas de la **segunda tabla**. El ejemplo anterior quedaría de la siguiente forma con la composición:

Idfab	idproducto
Bic	41672
Imm	77900
Imm	88701

fab	producto
aci	41004
aci	41002

COMPOSICIÓN

Idfab	idproducto	fab	producto
Bic	41672	aci	41004
Imm	77900	aci	41004
Imm	88701	aci	41004
Bic	41672	aci	41002
Imm	77900	aci	41002
Imm	88701	aci	41002

A diferencia de la unión la composición permite obtener una fila con datos de las dos tablas, esto es muy útil cuando queremos visualizar filas cuyos datos se encuentran en dos tablas distintas.
Por ejemplo, queremos listar los pedidos con el nombre del representante que ha hecho el pedido, pues los datos del pedido los tenemos en la tabla de pedidos pero el nombre del representante está en la tabla de empleados y además queremos que aparezcan en la misma línea; en este caso necesitamos componer las dos tablas (Nota: en el ejemplo expuesto a continuación, hemos seleccionado las filas que nos interesan).

numpedido	rep
110036	109
112963	105
...	...

numemp	nombre
109	Maria García
105	Vicente Pérez
...	...

COMPOSICIÓN

numpedido	rep	numemp	nombre
110036	109	109	Maria García
112963	105	105	Vicente Pérez
...	...	...	...

Existen distintos tipos de composición, aprenderemos a utilizarlos todos y a elegir el tipo más apropiado a cada caso.
Los **tipos de composición** de tablas son: el **producto cartesiano,** el **INNER JOIN** y el **LEFT / RIGHT JOIN.**

El producto cartesiano

El producto cartesiano es un tipo de composición de tablas, aplicando el producto cartesiano a dos tablas se obtiene una tabla con las **columnas** de la **primera tabla unidas** a las **columnas** de la **segunda tabla**, y las filas de la tabla resultante son **todas las posibles concatenaciones** de **filas** de la **primera tabla** con filas de la **segunda tabla**. La sintaxis es la siguiente:

FROM *nbtabla* [**AS** *aliastabla*] {, *nbtabla* [**AS** *aliastabla*]}

- El **producto cartesiano** se indica **poniendo** en la **FROM** las **tablas** que queremos componer **separadas por comas**, podemos obtener así el producto cartesiano de dos, tres, o más tablas.
- Hay que tener en cuenta que el producto cartesiano **obtiene** todas las **posibles combinaciones** de filas por lo tanto si tenemos dos tablas de 100 registros cada una, el resultado tendrá 100x100 filas, si el producto lo hacemos de estas dos tablas con una tercera de 20 filas, el resultado tendrá 200.000 filas (100x100x20) y estamos hablando de tablas pequeñas. Se ve claramente que el producto cartesiano es una **operación costosa** sobre todo si operamos con más de dos tablas o con tablas voluminosas.
- Se puede **componer** una **tabla consigo misma**, en este caso es **obligatorio** utilizar un nombre de **alias** por lo menos para una de las dos. Por ejemplo: **SELECT * FROM Empleados, Empleados emp**. En este ejemplo obtenemos el producto cartesiano de la tabla de empleados con ella misma. Todas las posibles combinaciones de empleados con empleados.
- Para ver cómo funciona el producto cartesiano cogemos las consultas [existencias cero] y [pedidos ene-2005] creadas en la página anterior, y creamos una consulta que halle el producto cartesiano de las dos. Obtenemos la siguiente tabla:

Idfab	idproducto	fab	producto
Bic	41672	aci	41004
Imm	77900	aci	41004
Imm	88701	aci	41004
Bic	41672	aci	41002
Imm	77900	aci	41002
Imm	88701	Aci	41002

Se observa que tenemos las dos filas de la primera consulta combinadas con las dos filas de la segunda.

- Esta operación **no** es de las **más utilizadas**, normalmente cuando queremos componer dos tablas es para añadir a las filas de una tabla, una fila de la otra tabla, por ejemplo añadir a los pedidos los datos del cliente correspondiente, o los datos del representante, esto equivaldría a un producto cartesiano con una selección de filas:

```
SELECT *
FROM Pedidos, Clientes
WHERE Pedidos.clie = Clientes.numclie
```

Combinamos todos los pedidos con todos los clientes pero luego seleccionamos los que cumplan que el código de cliente de la tabla de pedidos sea igual al código de cliente de la tabla de clientes, por lo tanto nos quedamos con los pedidos combinados con los datos del cliente correspondiente.

- Las columnas que aparecen en la cláusula WHERE de nuestra consulta anterior se denominan **columnas de emparejamiento** ya que permiten emparejar las filas de las dos tablas. Las columnas de emparejamiento no tienen por qué estar incluidas en la lista de selección.

- Normalmente, emparejamos tablas que están relacionadas entre sí y una de las columnas de emparejamiento actúa como clave principal. Sin embargo, cuando **una** de las **columnas de emparejamiento** tiene un **índice** definido es más eficiente utilizar otro tipo de composición, el **INNER JOIN**.

El INNER JOIN

El **INNER JOIN** es otro tipo de composición de tablas, permite **emparejar filas** de distintas tablas de forma **más eficiente** que con el producto cartesiano **cuando** una de las **columnas de emparejamiento** está **indexada**. Ya que en vez de hacer el producto cartesiano completo y luego seleccionar la filas que cumplen la condición de emparejamiento, para cada fila de una de las tablas **busca directamente** en la otra tabla **las filas que** cumplen la condición, con lo cual **se emparejan** sólo las filas que luego aparecen en el resultado.
La sintaxis es la siguiente:

FROM *tabla1* **INNER JOIN** *tabla2* **ON** *tabla1.col1* operador-comparación t*abla2.col2*

EJEMPLO

```
SELECT *
FROM Pedidos INNER JOIN Clientes ON Pedidos.clie = Clientes.numclie
```

- *tabla1* y *tabla2* son **especificaciones de tabla** (nombre de tabla con alias o no, nombre de consulta guardada), de las tablas cuyos registros se van a combinar.
- Pueden ser las dos la **misma tabla**, en este caso es **obligatorio** definir al menos un **alias** de tabla.
- *col1, col2* son las **columnas de emparejamiento**.
- Observar que dentro de la cláusula **ON** los nombres de **columna** deben ser **nombres cualificados** (llevan delante el nombre de la tabla y un punto).
- Las **columnas** de **emparejamiento** deben contener la **misma clase de datos**, las dos de tipo texto, de tipo fecha etc... los campos numéricos deben ser de tipos similares. Por ejemplo, se puede combinar campos AutoNumérico y Long puesto que son tipos similares, sin embargo, no se puede combinar campos de tipo Simple y Doble. Además las columnas no pueden ser de tipo Memo ni OLE.
- **Operador-comparación** representa cualquier operador de **comparación** (=, <, >, <=, >=, o <>) y se utiliza para establecer la condición de emparejamiento. Se pueden definir **varias condiciones** de emparejamiento **unidas** por los operadores **AND** y **OR** poniendo cada condición entre **paréntesis**.
  Ejemplo:
  ```
  SELECT *
  FROM Pedidos INNER JOIN Productos ON (Pedidos.fab = Productos.idfab)
                                    AND (Pedidos.producto = Productos.idproducto)
  ```

- Se pueden **combinar más** de **dos tablas.** En este caso hay que **sustituir** en la sintaxis una **tabla** por un **INNER JOIN completo**.
  Ejemplo:
  ```
  SELECT *
  FROM (Pedidos INNER JOIN Clientes ON Pedidos.clie = Clientes.numclie)
        INNER JOIN Empleados ON Pedidos.rep = Empleados.numemp
  ```

En vez de *tabla1* hemos escrito un INNER JOIN completo, también podemos escribir:

```
SELECT *
FROM Clientes INNER JOIN
        (Pedidos INNER JOIN Empleados ON Pedidos.rep = Empleados.numemp)
        ON Pedidos.clie = Clientes.numclie
```

En este caso hemos sustituido *tabla2* por un INNER JOIN completo.

El LEFT JOIN y RIGHT JOIN

El **LEFT JOIN** y **RIGHT JOIN** son otro tipo de composición de tablas, también denominada **composición externa**. Son una extensión del **INNER JOIN**.
Las composiciones vistas hasta ahora (el **producto cartesiano** y el **INNER JOIN**) son **composiciones internas** ya que todos los valores de las filas del resultado son valores que están en las tablas que se combinan.
Con una composición interna sólo se obtienen las filas que tienen al menos una fila de la otra tabla que cumpla la condición, veamos un ejemplo:
Queremos combinar los empleados con las oficinas para saber la ciudad de la oficina donde trabaja cada empleado, si utilizamos un producto cartesiano tenemos:

```
SELECT Empleados.*,ciudad
FROM Empleados, Oficinas
WHERE Empleados.oficina = Oficinas.oficina
```

Observar que hemos cualificado el nombre de columna oficina ya que ese nombre aparece en las dos tablas de la FROM.
Con esta sentencia los **empleados** que **no tienen** una **oficina** asignada (un valor nulo en el campo oficina de la tabla empleados) **no aparecen en el resultado** ya que la condición **Empleados.oficina = Oficinas.oficina** será siempre nula para esos empleados.

Si utilizamos el **INNER JOIN**:

```
SELECT Empleados.*, ciudad
FROM Empleados INNER JOIN Oficinas ON Empleados.oficina = Oficinas.oficina
```

Nos pasa lo mismo, el empleado 110 tiene un valor nulo en el campo oficina y no aparecerá en el resultado.
Pues en los casos en que **queremos** que **también aparezcan** las **filas que no tienen una fila coincidente** en la otra tabla, **utilizaremos** el **LEFT** o **RIGHT JOIN**.

La sintaxis del **LEFT JOIN** es la siguiente:

```
FROM tabla1 LEFT JOIN tabla2 ON tabla1.col1 operador-comparación tabla2.col2
```

La descripción de la sintaxis es la **misma** que la del **INNER JOIN** lo único que cambia es la palabra **LEFT** (**izquierda** en inglés).
Esta operación consiste en **añadir al resultado** del **INNER JOIN** las **filas** de la **tabla** de la **izquierda** que **no tienen correspondencia** en la otra tabla, y **rellenar** en esas filas los **campos** de la **tabla** de la **derecha** con **valores nulos**.

Ejemplo

```
SELECT *
FROM Empleados LEFT JOIN Oficinas ON Empleados.oficina = Oficinas.oficina
```

Con el ejemplo anterior obtenemos una lista de los empleados con los datos de su oficina, y el empleado 110 que no tiene oficina aparece con sus datos normales y los datos de su oficina a nulos.

La sintaxis del **RIGHT JOIN** es la siguiente:

FROM tabla1 **RIGHT JOIN** tabla2 **ON** tabla1.col1 **operador-comparación** tabla2.col2

La descripción de la sintaxis es la **misma** que la del **INNER JOIN** lo único que cambia es la palabra **RIGHT** (**derecha** en inglés).
Esta operación consiste en **añadir al resultado** del **INNER JOIN** las **filas** de la **tabla** de la **derecha** que **no tienen correspondencia** en la otra tabla, y **rellenar** en esas filas los **campos** de la **tabla** de la **izquierda** con **valores nulos**.

Ejemplo:

```
SELECT *
FROM Empleados RIGHT JOIN Oficinas ON Empleados.oficina = Oficinas.oficina
```

Con el ejemplo anterior obtenemos una lista de los empleados con los datos de su oficina, y además aparece una fila por cada oficina que no está asignada a ningún empleado con los datos del empleado a nulos.

Una operación **LEFT JOIN** o **RIGHT JOIN se puede anidar** dentro de una operación **INNER JOIN**, pero una operación **INNER JOIN no** se puede **anidar dentro** de **LEFT JOIN** o **RIGHT JOIN**. Los anidamientos de **JOIN** de distinta naturaleza no funcionan siempre, a veces depende del orden en que colocamos las tablas, en estos casos lo mejor es probar y si no permite el anudamiento, cambiar el orden de las tablas (y por tanto de los **JOIN**s) dentro de la cláusula **FROM**.

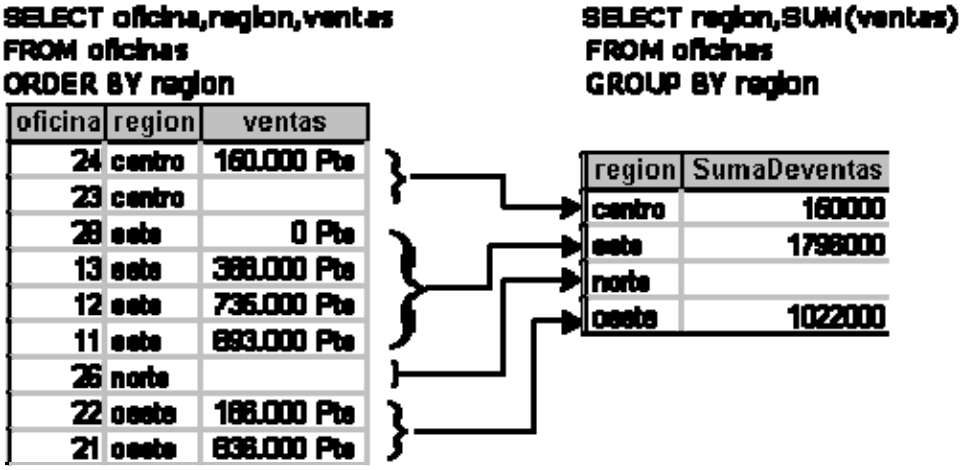
Por ejemplo podemos tener:

```
SELECT *
FROM Clientes INNER JOIN
   (Empleados LEFT JOIN Oficinas ON Empleados.oficina = Oficinas.oficina)
   ON Clientes.repclie = Empleados.numclie
```

Combinamos empleados con oficinas para obtener los datos de la oficina de cada empleado, y luego añadimos los clientes de cada representante, así obtenemos los clientes que tienen un representante asignado y los datos de la oficina del representante asignado.
Si hubiéramos puesto **INNER** en vez de **LEFT** no saldrían los clientes que tienen el empleado 110 (porque no tiene oficina y por tanto no aparece en el resultado del **LEFT JOIN** y por tanto no entrará en el cálculo del **INNER JOIN** con clientes).

8.- LAS CONSULTAS DE RESUMEN/SUMARIAS.

En la mayoría de los SGBD relacionales, podemos definir un **tipo de consultas** cuyas filas resultantes **son un resumen de las filas de la tabla origen**, por eso las denominamos **consultas de resumen**, también se conocen como consultas sumarias.
Es importante entender que las filas del resultado de una consulta de resumen tienen una **naturaleza distinta** a las filas de las demás tablas resultantes de consultas, ya que corresponden a varias filas de la tabla origen. Para simplificar, veamos el caso de una consulta basada en una sola tabla, una fila de una consulta 'no resumen' corresponde a una fila de la tabla origen, contiene datos que se encuentran en una sola fila del origen, mientras que **una fila de una consulta de resumen corresponde** a un **resumen de varias filas** de la **tabla origen**, esta diferencia es lo que va a originar una serie de restricciones que sufren las consultas de resumen.
En el ejemplo que viene a continuación tienes un ejemplo de consulta normal en la que se visualizan las filas de la tabla oficinas ordenadas por región, en este caso cada fila del resultado se corresponde con una sola fila de la tabla oficinas, mientras que la segunda consulta es una consulta resumen, cada fila del resultado se corresponde con una o varias filas de la tabla oficinas.



Las consultas de resumen introducen dos **nuevas cláusulas** a la sentencia SELECT, la **cláusula GROUP BY** y la **cláusula HAVING**, son cláusulas que **sólo** se pueden utilizar en una consulta de resumen, se tienen que escribir **entre** la cláusula **WHERE** y la cláusula **ORDER BY.**

Funciones de columna

En la lista de selección de una consulta de resumen aparecen **funciones de columna** también denominadas funciones de dominio agregadas. Una función de columna **se aplica a una columna** y obtiene un **valor que resume el contenido de la columna**.
Tenemos las siguientes funciones de columna:

SUM (expresión)	MIN (expresión)
AVG (expresión)	MAX (expresión)
STDEV (expresión)	COUNT (nbcolumns)
STDEVP (expresión)	COUNT (*)

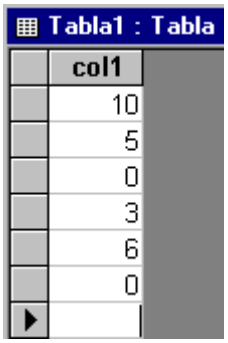
El **argumento** de la función indica **con qué valores** se tiene que operar, por eso ***expresión*** suele ser un **nombre de columna**, columna que contiene los valores a resumir, pero también puede ser cualquier expresión válida que devuelva una lista de valores.

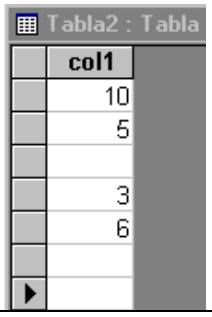
- La función **SUM()** calcula la **suma** de los valores indicados en el argumento. Los datos que se suman deben ser de **tipo numérico** (entero, decimal, coma flotante o monetario...). El resultado será del mismo tipo aunque puede tener una precisión mayor.
  Ejemplo:

  ```
  SELECT SUM(ventas)
  FROM Oficinas
  ```

  Obtiene una sola fila con el resultado de sumar todos los valores de la columna ventas de la tabla oficinas.

- La función **AVG()** calcula el **promedio** (la media arimética) de los valores indicados en el argumento, también se aplica a **datos numéricos**, y en este caso el tipo de dato del resultado puede cambiar según las necesidades del sistema para representar el valor del resultado.
- **StDev()** y **StDevP()** calculan la **desviación estándar** de una población o de una muestra de la población representada por los valores contenidos en la columna indicada en el argumento. Si la consulta base (el origen) tiene menos de dos registros, el resultado es nulo.
  Es interesante destacar que el **valor nulo no equivale al valor 0**, las **funciones de columna no consideran** los **valores nulos** mientras que consideran el valor 0 como un valor, por lo tanto en las funciones AVG(), STDEV(), STDEVP() los resultados no serán los mismos con valores 0 que con valores nulos. Veámoslo con un ejemplo:

Si tenemos esta tabla:
 
 	
	La consulta
	`SELECT AVG(col1) AS media`
	`FROM tabla1`
		devuelve:
		 
		
			En este caso los ceros entran en la media por lo que sale igual a 4
			(10+5+0+3+6+0)/6 = 4

Con esta otra tabla:  	`SELECT AVG(col1) AS media` `FROM tabla2`	devuelve:  	En este caso los ceros se han sustituido por valores nulos y no entran en el cálculo por lo que la media sale igual a 6 (10+5+3+6)/4 = 4

- Las funciones **MIN()** y **MAX()** determinan los **valores menores** y **mayores** respectivamente. Los valores de la columna pueden ser de **tipo numérico**, **texto** o **fecha**. El resultado de la función tendrá el mismo tipo de dato que la columna. Si la columna es de **tipo numérico MIN()** devuelve el **valor menor** contenido en la columna, si la columna es de **tipo texto MIN()** devuelve el **primer valor** en **orden alfabético**, y si la columna es de **tipo fecha**, **MIN()** devuelve la **fecha más antigua** y **MAX()** la **fecha más reciente**.
- La función **COUNT(nb columna)** cuenta el **número de valores** que hay **en la columna**, los datos de la columna pueden ser de **cualquier tipo**, y la función siempre devuelve un número entero. Si la columna contiene **valores nulos** esos valores **no se cuentan**, si en la columna aparece un **valor repetido**, lo **cuenta varias veces**.
- **COUNT(*)** permite **contar filas** en vez de valores. Si la columna no contiene ningún valor nulo, **COUNT**(nbcolumna) y **COUNT(*)** devuelven el mismo resultado, mientras que si hay valores nulos en la columna, **COUNT(*)** cuenta también esos valores mientras que **COUNT**(nb columna) no los cuenta.

Ejemplo:

¿Cuántos empleados tenemos?
```
SELECT COUNT(numemp)
FROM Empleados
```

o bien

```
SELECT COUNT(*)
FROM Empleados
```

En este caso las dos sentencias devuelven el mismo resultado ya que la columna numemp no contiene valores nulos (es la clave principal de la tabla empleados).

¿Cuántos empleados tienen una oficina asignada?
```
SELECT COUNT(oficina)
FROM Empleados
```

Esta sentencia por el contrario, nos devuelve el número de valores no nulos que se encuentran en la columna oficina de la tabla empleados, por lo tanto nos dice cuántos empleados tienen una oficina asignada.

- Se pueden combinar varias funciones de columna en una expresión pero no se pueden anidar funciones de columna, es decir:

```
SELECT (AVG(ventas) * 3) + SUM(cuota)
FROM ...
```

es correcto

```
SELECT AVG(SUM(ventas))
FROM ...
```

NO es correcto, no se puede incluir una función de columna dentro de una función de columna

Selección en el origen de datos

Si queremos **eliminar** del origen de datos algunas **filas**, basta incluir la cláusula **WHERE** que ya conocemos después de la cláusula **FROM**.

Ejemplo: Queremos saber el acumulado de ventas de los empleados de la oficina 12.

```
SELECT SUM(ventas)
FROM Empleados
WHERE oficina = 12
```

Origen múltiple

Si los **datos** que necesitamos utilizar para obtener nuestro resumen **se encuentran** en **varias tablas**, formamos el origen de datos adecuado en la cláusula **FROM** como si fuera una consulta **multitabla** normal.

Ejemplo: Queremos obtener el importe total de ventas de todos los empleados y el mayor objetivo de las oficinas asignadas a los empleados:

```
SELECT SUM(Empleados.ventas), MAX(objetivo)
FROM Empleados LEFT JOIN Oficinas ON Empleados.oficina = Oficinas.oficina
```

NOTA: combinamos empleados con oficinas por un **LEFT JOIN** para que aparezcan en el origen de datos todos los empleados incluso los que no tengan una oficina asignada, así el origen de datos estará formado por una tabla con tantas filas como empleados hayan en la tabla empleados, con los datos de cada empleado y de la oficina a la que está asignado. De esta tabla sacamos la suma del campo ventas (importe total de ventas de todos los empleados) y el objetivo máximo. Observar que el origen de datos no incluye las oficinas que no tienen empleados asignados, por lo que esas oficinas no entran a la hora de calcular el valor máximo del objetivo.

La cláusula GROUP BY

GROUP BY ***columna de agrupación {, columna de agrupación}***

Hasta ahora las consultas de resumen que hemos visto utilizan todas las filas de la tabla y producen una única fila resultado.

Se pueden obtener **subtotales** con la cláusula **GROUP BY**. Una consulta con una cláusula **GROUP BY** se denomina **consulta agrupada** ya que agrupa los datos de la tabla origen y **produce una única fila resumen por cada grupo formado**. Las columnas indicadas en el **GROUP BY** se llaman **columnas de agrupación**.

EJEMPLOS

`SELECT SUM(ventas)` `FROM Empleados`	Obtiene la suma de las ventas de todos los empleados.

SELECT SUM(ventas) **FROM Empleados** **GROUP BY oficina**	Se forma un grupo para cada oficina, con las filas de la oficina, y la suma se calcula sobre las filas de cada grupo. El ejemplo anterior obtiene una lista con la suma de las ventas de los empleados de cada oficina.

La consulta quedaría mejor incluyendo en la lista de selección la oficina para saber a qué oficina corresponde la suma de ventas:

```
SELECT oficina,SUM(ventas)
FROM Empleados
GROUP BY oficina
```

- Una columna de agrupación no puede ser de tipo memo u OLE.
- La **columna de agrupación** se puede indicar mediante un **nombre de columna** o cualquier expresión válida basada en una columna pero no se pueden utilizar los alias de campo.
- **En la lista de selección sólo pueden aparecer: valores constantes, funciones de columna, columnas de agrupación** (columnas que aparecen en la cláusula **GROUP BY**) **o cualquier expresión basada en las anteriores**.
- Se **pueden agrupar** las filas **por varias columnas**, en este caso se indican las columnas separadas por una coma y en el **orden de mayor a menor agrupación**. Se permite incluir en la lista de agrupación hasta 10 columnas.

Ejemplo: Queremos obtener la suma de las ventas de las oficinas agrupadas por region y ciudad:

```
SELECT SUM(ventas)
FROM Oficinas
GROUP BY region, ciudad
```

Se agrupa primero por región, y dentro de cada región por ciudad.

- **Todas las filas** que tienen **valor nulo** en el campo de agrupación, pasan a formar **un único grupo**. Es decir, considera el valor nulo como un valor cualquiera a efectos de agrupación.

Ejemplo:

```
SELECT oficina, SUM(ventas) AS ventas_totales
FROM Empleados
GROUP BY oficina
```

En el resultado aparece una fila con el campo oficina sin valor y a continuación una cantidad en el campo ventas_totales, esta cantidad corresponde a la suma de las ventas de los empleados que no tienen oficina asignada (campo *oficina* igual a nulo).

La cláusula HAVING

HAVING ***condición de selección***

- La cláusula **HAVING** nos permite **seleccionar filas** de la tabla resultante **de una consulta de resumen**.
- Para la condición de selección se pueden utilizar los mismos tests de comparación descritos en la cláusula **WHERE**, también se pueden escribir condiciones compuestas (unidas por los operadores **OR**, **AND**, **NOT**), pero existe una restricción.
- **En la condición de selección sólo pueden aparecer**: **valores constantes, funciones de columna, columnas de agrupación** (columnas que aparecen en la cláusula GROUP BY) **o cualquier expresión basada en las anteriores**.

Ejemplo: Queremos saber las oficinas con un promedio de ventas de sus empleados mayor que 300000 €.

```
SELECT oficina
FROM Empleados
GROUP BY oficina
HAVING AVG(ventas) > 300000
```

NOTA: Para obtener lo que se pide hay que calcular el promedio de ventas de los empleados de cada oficina, por lo que hay que utilizar la tabla empleados. Tenemos que agrupar los empleados por oficina y calcular el promedio para cada oficina, por último nos queda seleccionar del resultado las filas que tengan un promedio superior a 300000 €.

¿Cómo se ejecuta internamente una consulta de resumen?

1. Se forma la tabla origen de datos según la cláusula **FROM**,
2. se seleccionan del origen de datos las filas según la cláusula **WHERE**,
3. se forman los grupos de filas según la cláusula **GROUP BY**,
4. por cada grupo se obtiene una fila en la tabla resultante con los valores que aparecen en las cláusulas **GROUP BY, HAVING** y en la lista de selección,
5. se seleccionan de la tabla resultante las filas según la cláusula **HAVING**,
6. se eliminan de la tabla resultante las columnas que no aparecen en la lista de selección,
7. se ordenan las filas de la tabla resultante según la cláusula **ORDER BY**

¡¡¡ IMPORTANTE !!!

- Una consulta se convierte en consulta de resumen en cuanto aparece **GROUP BY**, **HAVING** o una función de columna.
- En una consulta de resumen, la lista de selección y la cláusula **HAVING** sólo pueden contener: valores constantes, funciones de columna, columnas de agrupación (columnas que aparecen en la cláusula **GROUP BY**) o cualquier expresión basada en las anteriores.

9.- LAS SUBCONSULTAS

Una **subconsulta es una sentencia SELECT que aparece dentro de otra sentencia SELECT** que llamaremos **consulta principal**.
Se puede encontrar **en la lista de selección**, **en la cláusula WHERE** o **en la cláusula HAVING** de la consulta principal.
Una subconsulta tiene la misma sintaxis que una sentencia SELECT normal exceptuando que aparece **encerrada entre paréntesis**, no puede contener la cláusula **ORDER BY**, ni puede ser la **UNION** de varias sentencias **SELECT**, además tiene algunas restricciones en cuanto a número de columnas según el lugar donde aparece en la consulta principal. Estas restricciones las iremos describiendo en cada caso.
Cuando se ejecuta una consulta que contiene una subconsulta, **la subconsulta se ejecuta por cada fila de la consulta principal**.
Se aconseja no utilizar campos calculados en las subconsultas, ralentizan la consulta.
Las consultas que utilizan subconsultas suelen ser **más fáciles de interpretar por el usuario**.

Referencias externas

A menudo, es necesario, dentro del cuerpo de una subconsulta, hacer referencia al valor de una columna en la fila actual de la consulta principal, ese nombre de columna se denomina referencia externa.
Una **referencia externa** es un nombre de columna que estando en la subconsulta, no se refiere a ninguna columna de las tablas designadas en la **FROM** de la subconsulta sino a una **columna de las tablas designadas en la FROM de la consulta principal.** Como la subconsulta se ejecuta por cada fila de la consulta principal, el valor de la referencia externa irá cambiando.

Ejemplo:

```
SELECT numemp, nombre,
      (SELECT MIN(fechapedido) FROM Pedidos WHERE rep = numemp)
FROM Empleados
```

En este ejemplo la consulta principal es **SELECT... FROM Empleados**.
La subconsulta es **(SELECT MIN(fechapedido) FROM Pedidos WHERE rep = numemp)**.
En esta subconsulta tenemos una referencia externa (*numemp*) es un campo de la tabla empleados (origen de la consulta principal).

¿Qué pasa cuando se ejecuta la consulta principal?

- Se coge el primer empleado y se calcula la subconsulta sustituyendo numemp por el valor que tiene en el primer empleado. La subconsulta obtiene la fecha más antigua en los pedidos del rep = 101,
- Se coge el segundo empleado y se calcula la subconsulta con numemp = 102 (numemp del segundo empleado)... y así sucesivamente hasta llegar al último empleado.
- Al final obtenemos una lista con el número, nombre y fecha del primer pedido de cada empleado.
- Si quitamos la cláusula **WHERE** de la subconsulta obtenemos la fecha del primer pedido de todos los pedidos no del empleado correspondiente.

Anidar subconsultas

Las subconsultas pueden **anidarse** de forma que **una subconsulta aparezca en la cláusula WHERE** (por ejemplo) **de otra subconsulta** que a su vez forma parte de otra consulta principal. En la práctica, una consulta consume mucho más tiempo y memoria cuando se incrementa el número de niveles de anidamiento. La consulta resulta también más difícil de leer, comprender y mantener cuando contiene más de uno o dos niveles de subconsultas.

Ejemplo:

```
SELECT numemp, nombre
FROM Empleados
WHERE numemp = (SELECT rep
                FROM Pedidos
                WHERE clie = (SELECT numclie
                              FROM Clientes
                              WHERE nombre = 'Inmaculada Santos'))
```

En este ejemplo, por cada línea de pedido se calcula la subconsulta de clientes, y esto se repite por cada empleado, en el caso de tener 10 filas de empleados y 200 filas de pedidos (tablas realmente pequeñas), la subconsulta más interna se ejecutaría 2000 veces (10 x 200).

Subconsulta en la lista de selección

Cuando la subconsulta aparece **en la lista de selección** de la consulta principal, en este caso la subconsulta, **no puede devolver varias filas ni varias columnas**, de lo contrario se da un mensaje de error.
Muchos SQLs no permiten que una subconsulta aparezca en la lista de selección de la consulta principal pero eso no es ningún problema ya que normalmente se puede obtener lo mismo utilizando como origen de datos las dos tablas. El ejemplo anterior se puede obtener de la siguiente forma:

```
SELECT numemp, nombre, MIN(fechapedido)
FROM Empleados LEFT JOIN Pedidos ON Empleados.numemp = Pedidos.rep
GROUP BY numemp, nombre
```

En la cláusula FROM

En la cláusula FROM se puede encontrar una sentencia **SELECT** encerrada entre paréntesis pero **más que subconsulta sería una consulta** ya que no se ejecuta para cada fila de la tabla origen sino que se ejecuta una sola vez al principio, su resultado se combina con las filas de la otra tabla para formar las filas origen de la **SELECT** primera y no admite referencias externas.
En la cláusula **FROM** vimos que se podía poner un nombre de tabla o un nombre de consulta, pues en vez de poner un nombre de consulta se puede poner directamente la sentencia **SELECT** correspondiente a esa consulta encerrada entre paréntesis.

Subconsulta en las cláusulas WHERE y HAVING

Se suele utilizar subconsultas en las cláusulas **WHERE** o **HAVING** cuando los datos que queremos visualizar están en una tabla pero para seleccionar las filas de esa tabla necesitamos un dato que está en otra tabla.

Ejemplo:

```
SELECT numemp, nombre
FROM Empleados
WHERE contrato = (SELECT MIN(fechapedido) FROM Pedidos)
```

En este ejemplo listamos el número y nombre de los empleados cuya fecha de contrato sea igual a la primera fecha de todos los pedidos de la empresa.

En una cláusula **WHERE**/**HAVING** tenemos siempre una condición y la subconsulta actúa de operando en esa condición.
En el ejemplo anterior se compara contrato con el resultado de la subconsulta. Hasta ahora las condiciones estudiadas tenían como operandos valores simples (el valor contenido en una columna de una fila de la tabla, el resultado de una operación aritmética...) ahora la subconsulta puede devolver una columna entera por lo que es necesario definir otro tipo de **condiciones especiales** para cuando se utilizan con subconsultas.

Condiciones de selección con subconsultas

Las **condiciones de selección** son las condiciones que pueden aparecer en la cláusula **WHERE** o **HAVING**. La mayoría ya se han visto, pero ahora incluiremos las condiciones que utilizan una subconsulta como operando.
En SQL tenemos cuatro nuevas condiciones: el **test de comparación con subconsulta**, el **test de comparación cuantificada** el **test de pertenencia a un conjunto**, el **test de existencia**
En todos los tests estudiados a continuación *expresion* puede ser cualquier nombre de columna de la consulta principal o una expresión válida.

a) El test de comparación con subconsulta.

Es el **equivalente al test de comparación simple**. Se utiliza para comparar un valor de la fila que se está examinado con un único valor producido por la subconsulta. La subconsulta debe devolver una **única columna**, sino se produce un error.
Si la subconsulta no produce **ninguna fila** o devuelve el valor **nulo**, el test devuelve el **valor nulo**, si la subconsulta produce **varias filas**, SQL devuelve una **condición de error**.

La sintaxis es la siguiente:

expresion1 = | <> | < | <= | > | >= *subconsulta*

```
SELECT oficina, ciudad
FROM Oficinas
WHERE objetivo > (SELECT SUM(ventas)
                  FROM Empleados
                  WHERE Empleados.oficina = Oficinas.oficina)
```

Lista las oficinas cuyo objetivo sea superior a la suma de las ventas de sus empleados. En este caso la subconsulta devuelve una única columna y una única fila (es un consulta de resumen sin **GROUP BY**)

b) El test de comparación cuantificada.

Este test es una extensión del test de comparación y del test de conjunto. **Compara el valor** de la expresión **con cada uno de los valores** producidos por la subconsulta. La subconsulta debe devolver una **única columna** sino se produce un error.
Tenemos el test **ANY** (*algún, alguno* en inglés) y el test **ALL** (*todos* en inglés).
La sintaxis es la siguiente:

expresion1 **=** | **<>** | **<** | **<=** | **>** | **>=** **ANY**|**ALL** *subconsulta*

El **test ANY**.
La subconsulta debe devolver una única columna sino se produce un error.
Se evalúa la comparación con cada valor devuelto por la subconsulta.
Si **alguna de las comparaciones individuales produce el resultado verdadero, el test ANY devuelve el resultado verdadero**.
Si la subconsulta no devuelve **ningún valor**, el test **ANY** devuelve **falso**.
Si el test de comparación es **falso** para **todos los valores** de la columna, **ANY** devuelve **falso**.
Si el test de comparación **no es verdadero** para ningún valor de la columna, **y es nulo** para al menos alguno de los valores, **ANY** devuelve **nulo**.

```
SELECT oficina, ciudad
FROM Oficinas
WHERE objetivo > ANY (SELECT SUM(cuota)
                      FROM Empleados
                      GROUP BY oficina)
```

En este caso la subconsulta devuelve una única columna con las sumas de las cuotas de los empleados de cada oficina.
Lista las oficinas cuyo objetivo sea superior a alguna de las sumas obtenidas.

El **test ALL**.
La subconsulta debe devolver una única columna sino se produce un error.
Se evalúa la comparación con cada valor devuelto por la subconsulta.
Si todas las comparaciones individuales, producen un resultado **verdadero**, el test devuelve el valor **verdadero**.
Si la subconsulta no devuelve **ningún valor** el test **ALL** devuelve el valor **verdadero**. (¡Ojo con esto!)
Si el test de comparación es **falso** para algún valor de la columna, el resultado es **falso**.
Si el test de comparación **no es falso** para ningún valor de la columna, pero es **nulo** para alguno de esos valores, el test **ALL** devuelve valor **nulo**.

```
SELECT oficina, ciudad
FROM Oficinas
WHERE objetivo > ALL (SELECT SUM(cuota)
                      FROM Empleados
                      GROUP BY oficina)
```

En este caso se listan las oficinas cuyo objetivo sea superior a todas las sumas.

c) Test de pertenencia a conjunto (IN).

Examina si el **valor** de la expresión es uno de los valores **incluidos en** la **lista de valores producida por la subconsulta**.
La subconsulta debe generar una única columna y las filas que sean.
Si la subconsulta no produce ninguna fila, el test da falso.
Tiene la siguiente sintaxis:

expresión **IN** *subconsulta*

```
SELECT numemp, nombre, oficina
FROM Empleados
WHERE oficina IN (SELECT oficina
                  FROM Oficinas
                  WHERE region = 'este')
```

Con la subconsulta se obtiene la lista de los números de oficina del *este* y la consulta principal obtiene los empleados cuyo número de oficina sea uno de los números de oficina del *este*.
Por lo tanto lista los empleados de las oficinas del *este*.

d) El test de existencia EXISTS.

Examina si la subconsulta produce alguna fila de resultados.
Si la subconsulta contiene filas, el test adopta el valor **verdadero**, si la subconsulta no contiene ninguna fila, el test toma el valor falso, nunca puede tomar el valor nulo.
Con este test la subconsulta **puede tener varias columnas**, no importa ya que el test se fija no en los valores devueltos sino en si hay o no fila en la tabla resultado de la subconsulta.
Cuando se utiliza el test de existencia en la mayoría de los casos habrá que utilizar una referencia externa.
Si no se utiliza una referencia externa la subconsulta devuelta siempre será la misma para todas las filas de la consulta principal y en este caso se seleccionan todas las filas de la consulta principal (si la subconsulta genera filas) o ninguna (si la subconsulta no devuelve ninguna fila)
La sintaxis es la siguiente:

[**NOT**] **EXISTS** *subconsulta*

```
SELECT numemp, nombre, oficina
FROM Empleados
WHERE EXISTS (SELECT *
                    FROM Oficinas
                    WHERE region = 'este' AND Empleados.oficina = Oficinas.oficina)
```

Este ejemplo obtiene lo mismo que el ejemplo del test **IN**.
Observa que delante de **EXISTS** no va ningún nombre de columna.
En la subconsulta se pueden poner las columnas que queramos en la lista de selección (hemos utilizado el *).
Hemos añadido una condición adicional al **WHERE**, la de la referencia externa para que la oficina que se compare sea la oficina del empleado.

NOTA: Cuando se trabaja con tablas muy voluminosas el test **EXISTS** suele dar mejor rendimiento que el test **IN**.