CONTENIDO

Tema 3. Lenguajes QBE y SQL

Tema 3.1. Consultas QBE

Las consultas son expresiones que, entre otras cosas, permiten obtener datos de las bases de datos. En Access se pueden utilizar dos lenguajes para expresar estas consultas: QBE (Query By Example, consulta mediante ejemplos) y SQL (Structured Query Language, lenguaje estructurado de consulta). En este tema usaremos fundamentalmente QBE.

Query-by-Example (QBE, Consulta mediante ejemplos) se refiere a una familia de lenguajes que implementan las ideas del cálculo relacional de dominios, un lenguaje formal desarrollado para las bases de datos relacionales. Es el nombre tanto de un lenguaje de manipulación de datos como el de un sistema de base de datos que incluyó a este lenguaje. El sistema QBE se desarrolló en el Centro de desarrollo T.J. Watson, de IBM, a principios de los años setenta y el lenguaje de manipulación de datos QBE se usó más tarde en QMF (Query Management Facility, mecanismo de gestión de consultas) como opción de interfaz para DB2. Hay varias implementaciones de este lenguaje, entre las que se incluyen el original de IBM (Sistema QBE), QBE de Microsoft (en Access) y QBE de DB2. Aunque este lenguaje fue originalmente textual, las últimas implementaciones, como la de Access, ofrecen una interfaz gráfica para la expresión de consultas.

Se pueden distinguir varios tipos de consultas:

- Selección básica. Seleccionan registros de una o varias tablas.
- Resumen. Calculan totales para columnas.
- Tabla de referencias cruzadas. Calculan totales para filas y columnas.
- Parámetros. Aceptan valores proporcionados por el usuario para personalizar la consulta.
- Acción. Modifican datos de una tabla (reemplazan datos, eliminan registros o añaden registros).

Para crear una consulta se realizan los siguientes pasos:

- Abrir la consulta (aportando un nombre que la identifique).
- Agregar las tablas implicadas.
- Establecer las relaciones entre las tablas agregadas.
- Establecer los criterios o condiciones de selección de los registros.

En este tema se estudiarán las consultas QBE mediante la creación de diferentes tipos de consultas:

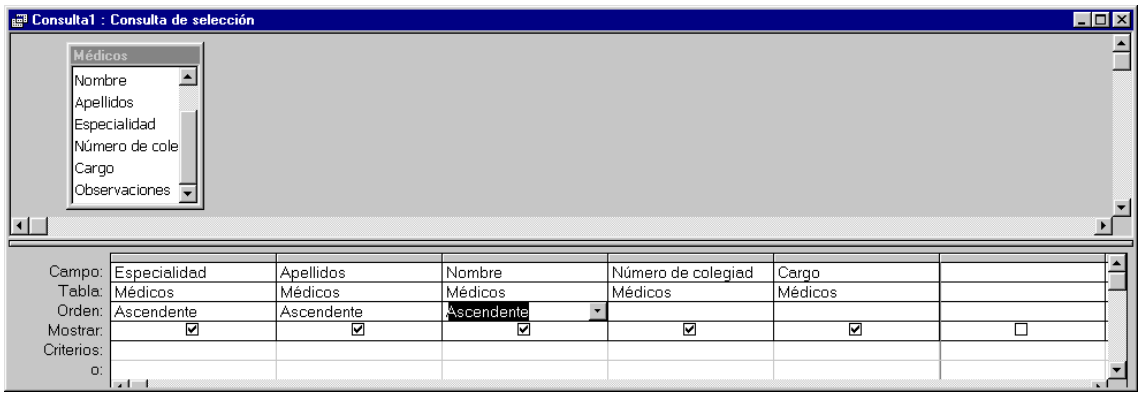
- Dos consultas de selección básica:
  - Listado de médicos por especialidad.
  - Listado de pacientes por diagnóstico.
- Una consulta de resumen:
  - Gasto total diario por tratamiento.
- Tres consultas de tabla de referencias cruzadas:
  - Gasto trimestral por planta.
  - Gasto trimestral por diagnóstico.
  - Resumen anual de gastos por planta.
- Tres consultas de parámetros:
  - Ingresos de pacientes entre dos fechas.
  - Ingresos de pacientes por diagnóstico.
  - Historial clínico de un paciente.
- Cuatro consultas de acción:
  - Anexar ingresos de la planta 2.
  - Eliminar ingresos de la planta 2.
  - Recuperar filas de Ingresos.
  - Reducir coste del tratamiento

3.1.1. Consultas de selección básica

Las consultas más sencillas, las consultas de selección, permiten extraer datos de la base de datos con el criterio que imponga el usuario. En su versión más simple, una consulta de selección mostraría todos los campos de todas las filas de una tabla. Se puede elegir tanto los campos a mostrar como las filas que cumplan una determinada condición.

La ventana QBE que se muestra en la siguiente figura consta de dos paneles: el superior, que contiene todas las tablas de las que se pueden escoger campos para mostrar como resultado de la consulta. También se muestran las relaciones definidas entre campos de diferentes tablas. Estas relaciones pueden mostrar también restricciones de integridad referencial si muestran los símbolos 1 o infinito en sus extremos. Si el 1 etiqueta ambos

extremos, indica una restricción de cardinalidad uno a uno. Si aparece un 1 y un infinito es una relación uno a varios, y si en ambos extremos aparece un símbolo de infinito, indica una relación de varios a varios.



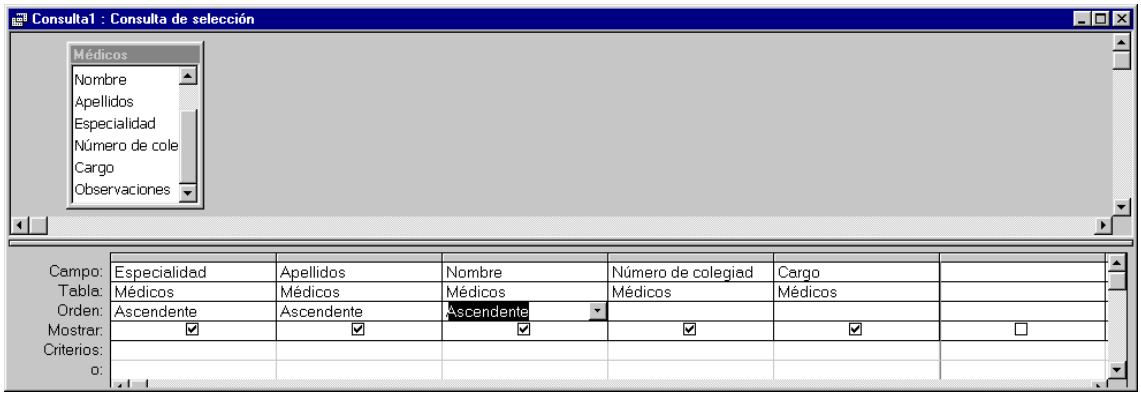
En el panel inferior se muestra la rejilla QBE, que consta de: 1) columnas para indicar los campos o las expresiones que se desean mostrar con los detalles que se indican en las celdas de cada columna, y 2) filas para indicar el nombre del campo (fila Campo), la tabla de origen (fila Tabla), el orden en que se muestran los resultados (fila Orden), si se desea mostrar el campo o expresión (fila Mostrar), el criterio de selección (en fila Criterios y en la fila o:, que indica una disyunción o alternativa).

3.1.1.1. Listado de médicos por especialidad

En esta consulta estamos interesados en obtener los médicos ordenados por especialidad y por apellidos, incluyendo los campos de la tabla Médicos: Especialidad, Apellidos, Nombre, Número de colegiado y Cargo.

Pasos:

1. Seleccionar la ficha Crear y pulsar Diseño de consulta.
2. Pulsar en la tabla Médicos del cuadro de diálogo Mostrar tabla, pulsar Agregar y pulsar Cerrar.
3. Arrastrar el campo Especialidad a la fila Campo de la primera columna de la cuadrícula.
4. Hacer doble clic en los campos Apellidos, Nombre, Número de colegiado y Cargo (se copiarán a la cuadrícula).
5. Pulsar en la fila Orden del campo Especialidad de la cuadrícula y seleccionar Ascendente.
6. Repetir el paso anterior para el campo Apellidos y para el campo Nombre.

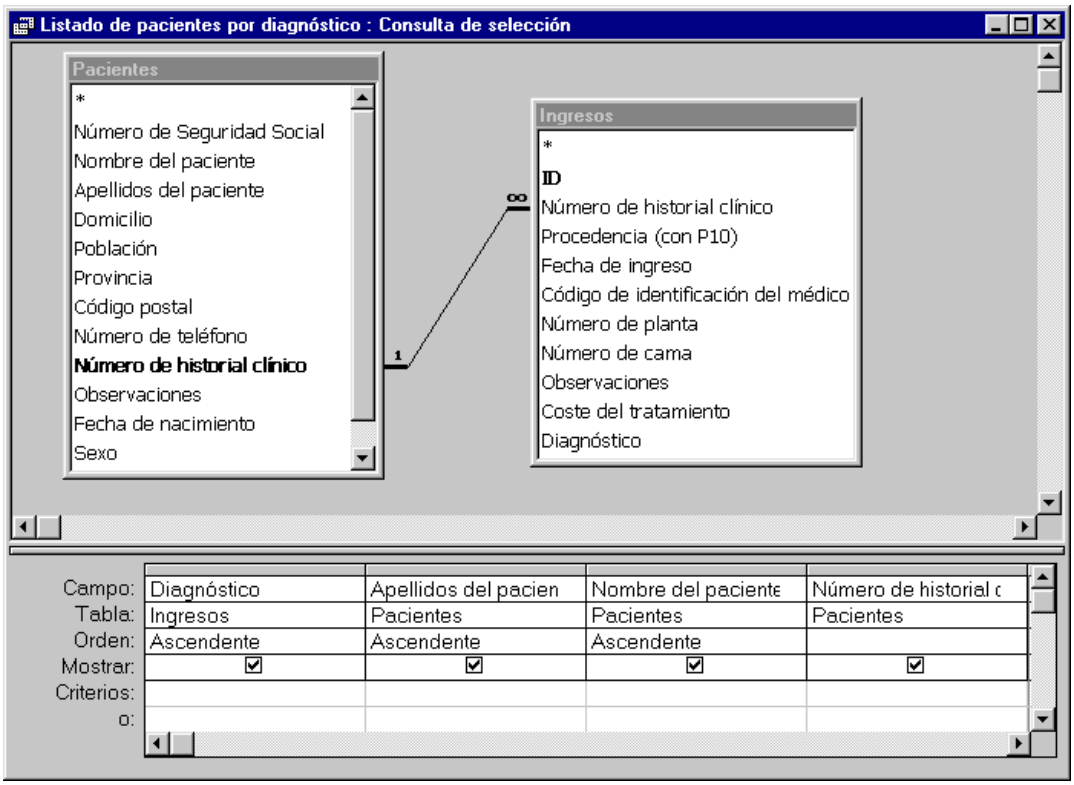


7. Almacenar la consulta con Guardar como del botón Inicio.
8. Escribir Listado de médicos por especialidad y pulsar Aceptar.
9. Cerrar la ventana de creación de consultas y ejecutar la consulta haciendo doble clic en la ficha Consultas de la base de datos Hospital. El resultado debería ser:

3.1.1.2. Listado de pacientes por diagnóstico

Proporciona un listado de pacientes ordenado por diagnóstico y por los apellidos del paciente. Para crear la consulta Listado de pacientes por diagnóstico hay que agregar las tablas Pacientes e Ingresos. Se añadirá a la cuadrícula QBE los siguientes campos de las siguientes tablas:

Campo	**Tabla**
Diagnóstico	Ingresos
Apellidos del paciente	Pacientes
Nombre del paciente	Pacientes
Número de historial clínico	Pacientes



En esta figura se puede observar que se ha impuesto una restricción de integridad referencial sobre el campo Número de historial clínico de la tabla Ingresos: cualquier valor de este campo debe encontrar una correspondencia en el campo de mismo nombre de la tabla Pacientes. Esto quiere decir que si intentamos insertar una tupla en Ingresos con un historial que no aparezca para ningún paciente, el gestor de bases de datos rechazará la operación. Esta restricción de integridad referencial está implementando una restricción de cardinalidad de 1 a varios porque el campo Número de historial clínico es clave primaria en la tabla Pacientes (no se puede repetir), mientras que no lo es en la tabla Ingresos (sí se puede repetir).

3.1.2. Consultas de resumen

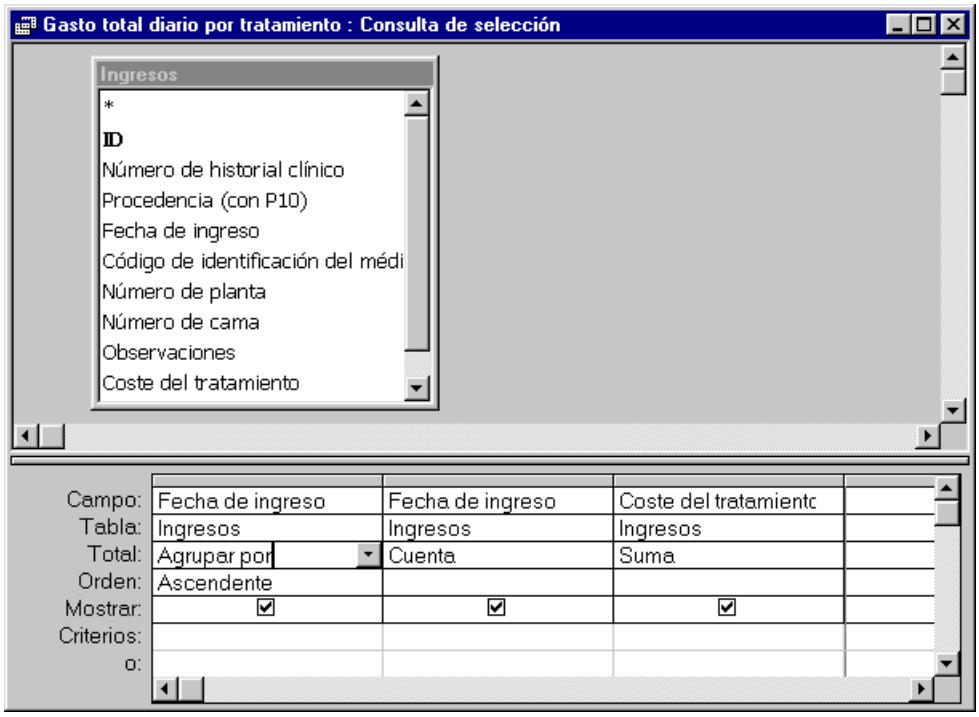
Este tipo de consultas calculan información resumida como totales y medias de un campo en concreto (el cálculo es por columnas). Generalmente se aplican a campos numéricos y con fines estadísticos. Además de los totales y las medias, también es posible calcular el recuento de filas, y los valores máximos y mínimos de un campo. En estos casos se puede aplicar también a otros tipos de datos, como campos textuales.

3.1.2.1. Gasto total diario por tratamiento

Debe mostrar un listado con el número de ingresos diario y el coste total del tratamiento. En esta consulta se introduce el cálculo de totales. Para ello hay que:

1. Agregar la tabla Ingresos.

2. Agregar el campo Fecha de ingreso dos veces y Coste del tratamiento.
3. Pulsar el botón Totales de la barra de herramientas (Σ) o activar la opción Totales del menú Ver.
4. Seleccionar Agrupar por de la casilla Totales de la cuadrícula (con esto se definen el campo o conjunto de campos para los que se calcula un total) para el campo Fecha de ingreso.
5. Seleccionar Cuenta en la casilla Totales del segundo campo Fecha de ingreso.
6. Seleccionar Suma en la casilla Totales del campo Coste del tratamiento.



Se puede cambiar el nombre de los encabezados del resultado de la consulta anteponiendo al nombre del campo el nombre del encabezado separado por dos puntos. Por ejemplo Fecha:Fecha de ingreso.

El resultado de la consulta con los nombres de los encabezados cambiados sería:

Resumen de opciones de la fila Total	
Opción	**Acción**
Agrupar por	Define los grupos para los que desea realizar los cálculos.
Suma	Calcula la suma del valor numérico contenido en el campo de todos los registros.
Promedio	Calcula la media aritmética del valor numérico contenido en el campo de todos los registros.
Mín	Halla el mínimo valor numérico contenido en ese campo.
Máx	Halla el máximo valor numérico contenido en ese campo.
Cuenta	Cuenta el número de registros de la tabla que no están en blanco.

3.1.3. Consultas de tabla de referencias cruzadas

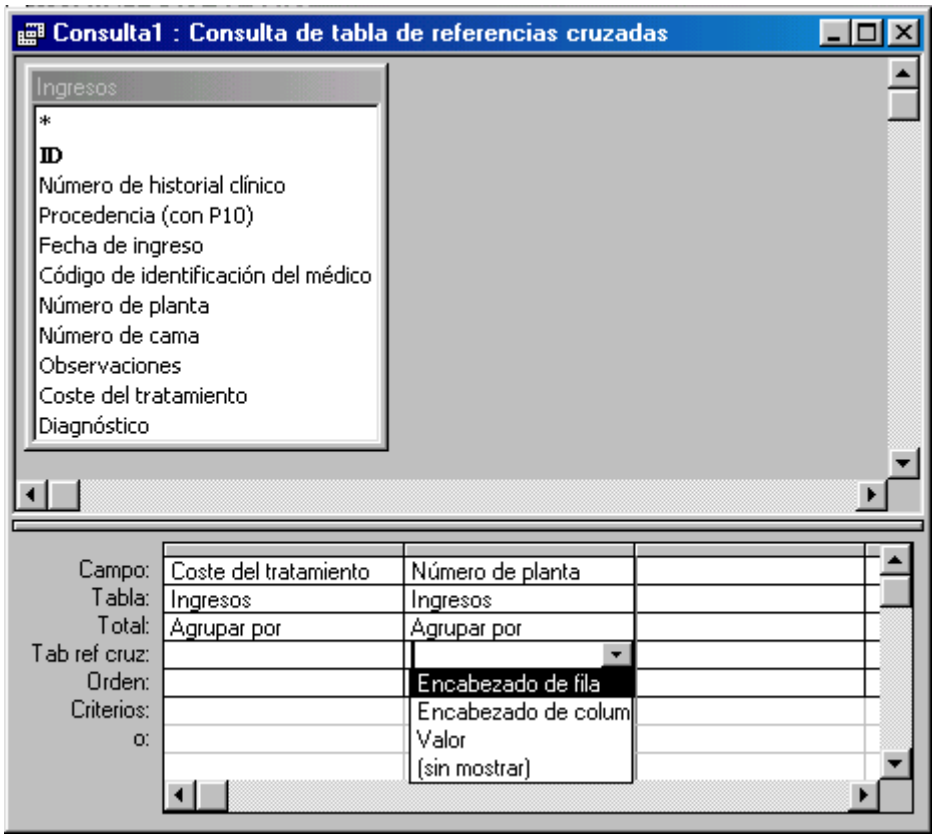
Este tipo de consultas permite generar columnas que no existen en una determinada tabla a partir de los datos que aparecen en las filas. Son útiles cuando, por ejemplo, deseamos generar columnas con fechas o intervalos de tiempo (como un año) que contengan totales (como total de gastos). Estas consultas son difíciles de expresar en el lenguaje SQL estándar (que se verá en el siguiente tema), pero Access contiene una instrucción especial para ellas

3.1.3.1. Gasto trimestral por planta

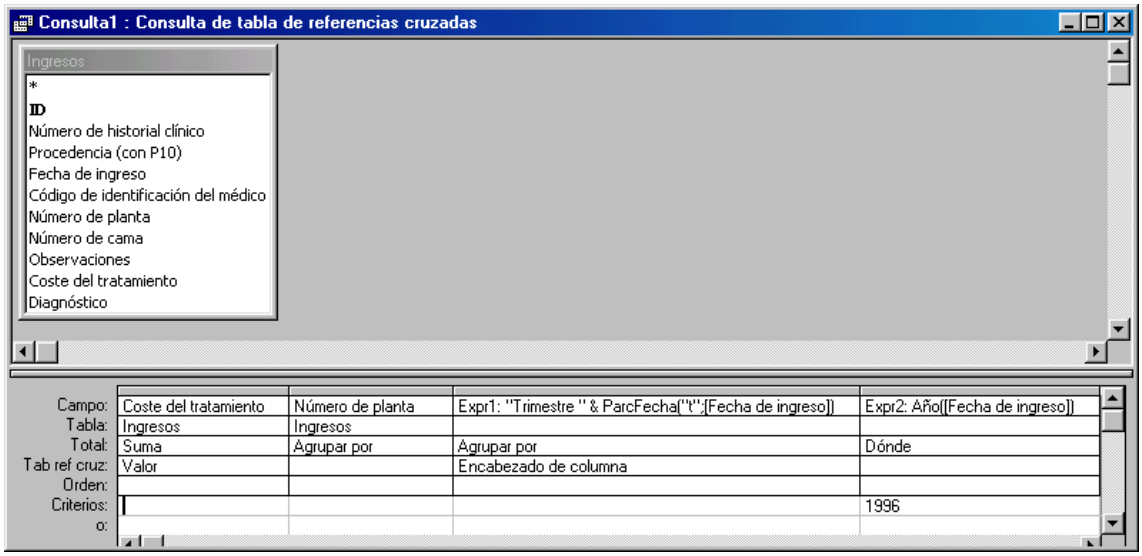
Debe proporcionar el gasto total de cada una de las plantas en el año 2010 agrupado por trimestre. Hay que seguir los pasos:

1. Seleccionar la ficha Crear y pulsar Diseño de consulta .

2. Agregar la tabla Ingresos y pulsar Cerrar.
3. En la ficha Diseño, pulsar General en Tipo de consulta.
4. Hacer doble clic en los campos Coste del tratamiento y Número de planta.
5. En la cuadrícula, pulsar en la fila Tab ref cruz del campo Número de planta y seleccionar Encabezado de fila de la lista desplegable.



6. Ahora se definen los encabezados del resto de columnas. Para la tercera hay que pulsar en la fila Campo y escribir "Trimestre " & ParcFecha("t";[Fecha de ingreso]). Seleccionar Tab ref cruz de esta columna y escoger Encabezado de columna. En la fila Total dejar Agrupar por.
7. Pulsar en la fila Tab ref cruz del campo Coste del tratamiento y seleccionar Valor. En la fila Total, seleccionar Suma.
8. Para que los cálculos se apliquen a 2010 hay que pulsar en la fila Campo de la cuarta columna y escribir Año([Fecha de ingreso]). Seleccionar Dónde en la fila Total y escribir 2010 en la fila Criterios. La fila Tab ref cruz debe quedar en blanco.



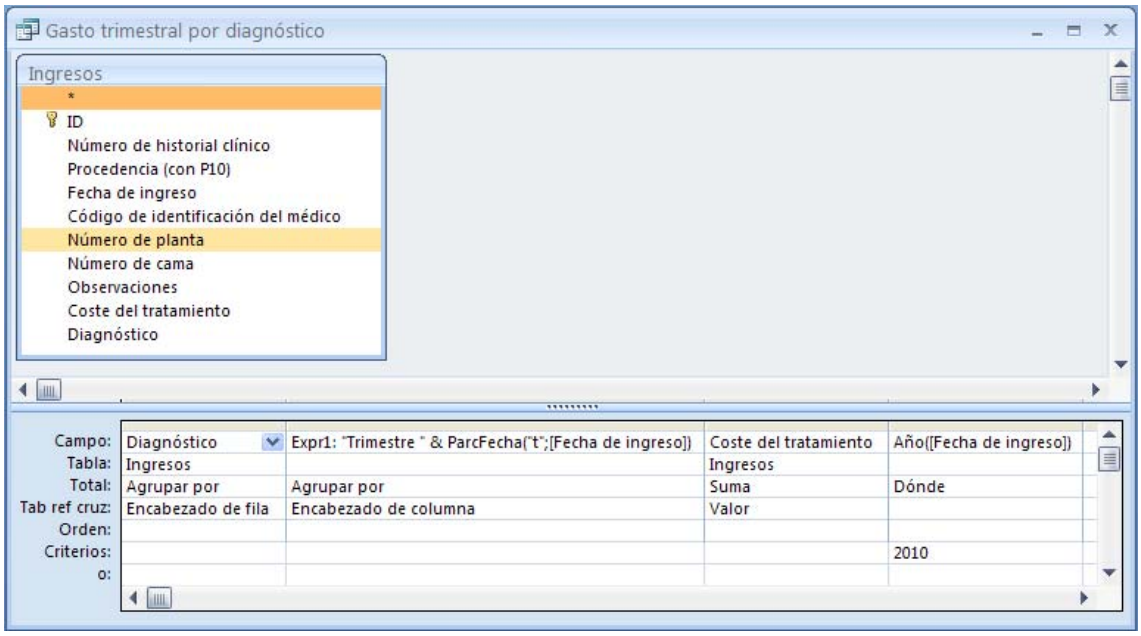
9. Pulsar el botón Vista de la barra de herramientas para ver el resultado.

10. Almacenar la consulta con el nombre Gasto trimestral por planta

3.1.3.2. Gasto trimestral por diagnóstico

Esta consulta debe producir para cada diagnóstico el gasto producido para cada uno de los trimestres del año.

1. Abrir la consulta Gasto trimestral por planta en Vista Diseño.
2. Cambiar el campo Número de planta por el campo Diagnóstico y guardar la consulta como Gasto trimestral por diagnóstico. La consulta debe quedar:



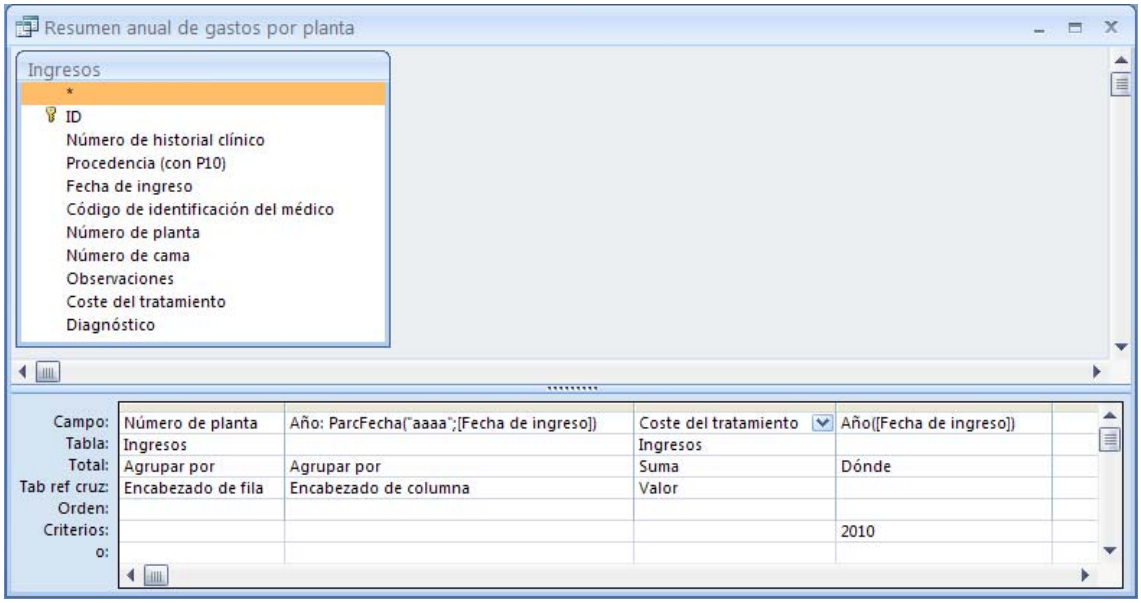
El resultado de su ejecución debe ser:

3.1.3.3. Resumen anual de gastos por planta

Esta consulta mostrará para cada planta el gasto realizado durante el año 2010.

1. Abrir la consulta Gasto trimestral por planta en Vista Diseño.
2. Sustituir el campo "Trimestre " & ParcFecha("t";[Fecha de ingreso]) por Año: ParcFecha("aaaa";[Fecha de ingreso]) y guardar la consulta como Resumen anual de gastos por planta.

   La consulta debe quedar:



El resultado de su ejecución debe ser:

3.1.4. Consultas de parámetros

Las consultas de parámetros permiten que el usuario proporcione datos que determinen el comportamiento de estas consultas. Estos datos particularizan las consultas, de forma que se adaptan a las necesidades concretas del usuario.

3.1.4.1. Ingresos de pacientes entre dos fechas

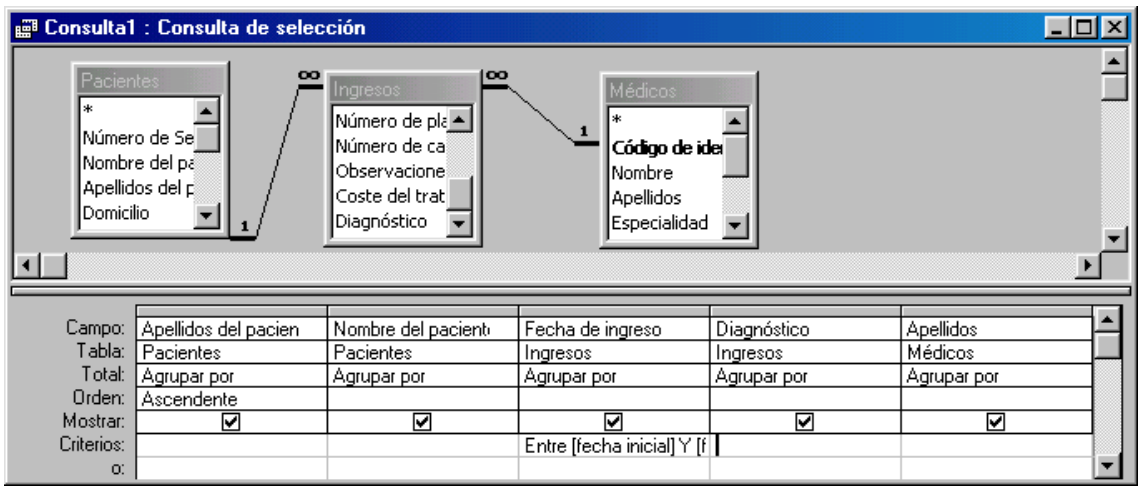
Dadas dos fechas que debe proporcionar el usuario al ejecutar la consulta, su resultado debe ser el nombre y apellidos de todos los pacientes que han ingresado entre las fechas proporcionadas, la fecha en que ingresaron al paciente y el médico que lo atendió.

Los datos se encuentran en las tablas Pacientes (nombre y apellidos del paciente), Ingresos (fecha de ingreso) y Médicos (nombre y apellidos del médico).

Pasos:

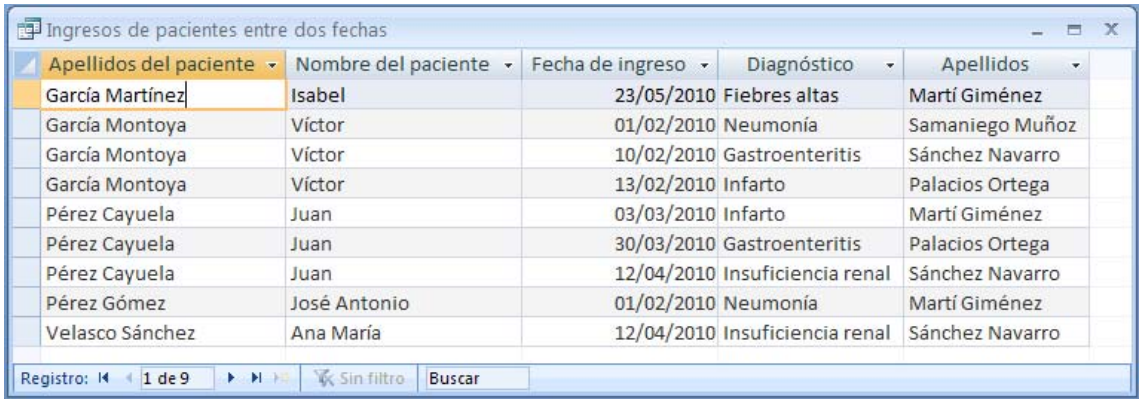
1. Seleccionar la opción Diseño de consulta en la ficha Crear.

2. Pulsar dos veces en el nombre de las tablas Pacientes, Ingresos y Médicos en el cuadro de diálogo Mostrar tabla para agregar estas tablas al diseño de la consulta.
3. Pulsar dos veces en los campos Apellidos del paciente y Nombre del paciente de la tabla Pacientes; en los campos Fecha de ingreso y Diagnóstico de la tabla Ingresos y en el campo Apellidos de la tabla Médicos ,en este orden, para llevarlos a la cuadrícula QBE.
4. Seleccionar Totales en Mostrar u ocultar de la ficha Diseño.
5. Pulsar en la fila Orden del campo Apellidos del paciente, desplegar la lista y elegir Ascendente.
6. En la fila Criterios del campo Fecha de ingreso escribir Entre [fecha inicial] Y [fecha final].



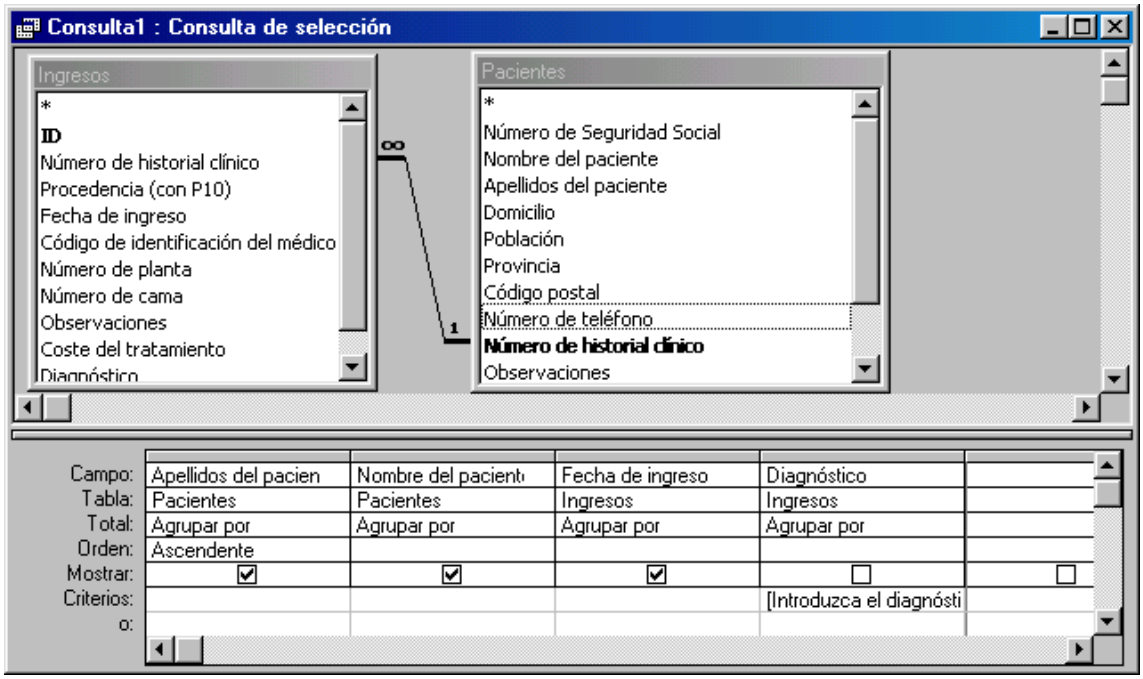
7. Almacenar la consulta con la orden Guardar como del botón Inicio, con nombre Ingresos de pacientes entre dos fechas.

El resultado de su ejecución para el intervalo de fechas del 1/1/2010 al 1/6/2010 debe ser:

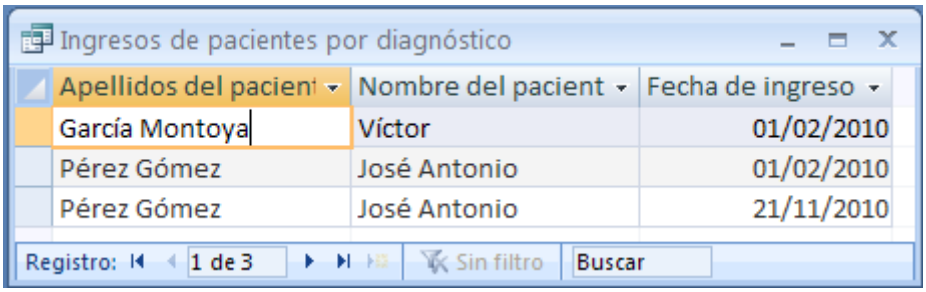


3.1.4.2. Ingresos de pacientes por diagnóstico

Esta consulta debe presentar el nombre, apellidos y fecha de ingreso de todos los pacientes para un diagnóstico determinado que será introducido en la consulta como parámetro.



El resultado para el diagnóstico Neumonía debe ser:

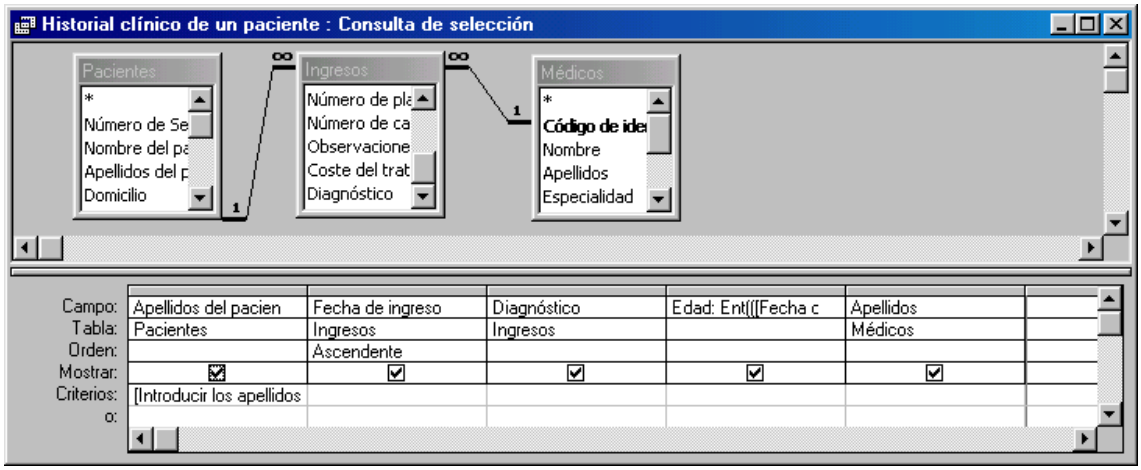


3.1.4.3. Historial clínico de un paciente

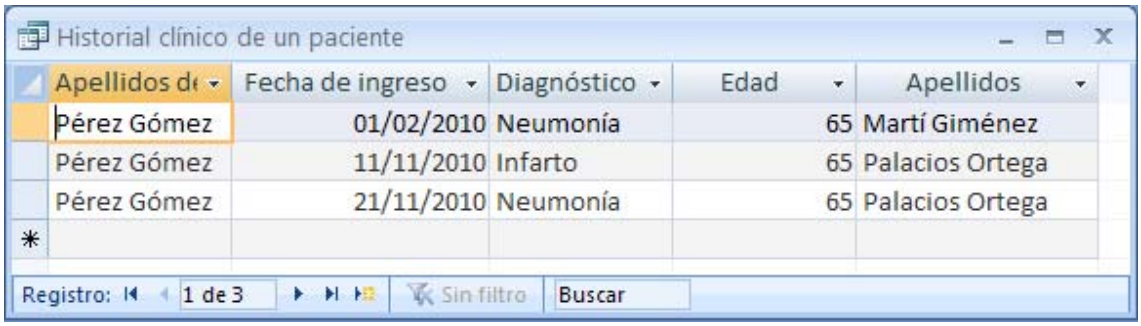
Esta consulta debe solicitar los apellidos del paciente y mostrar la fecha de ingreso, el diagnóstico, la edad del paciente y los apellidos del médico que atendió al paciente.

La edad puede calcularse con la expresión: Edad: Ent(([Fecha de ingreso]-[Fecha de nacimiento])/365).

La cuadrícula QBE quedará:



El resultado para el paciente Pérez Gómez será:

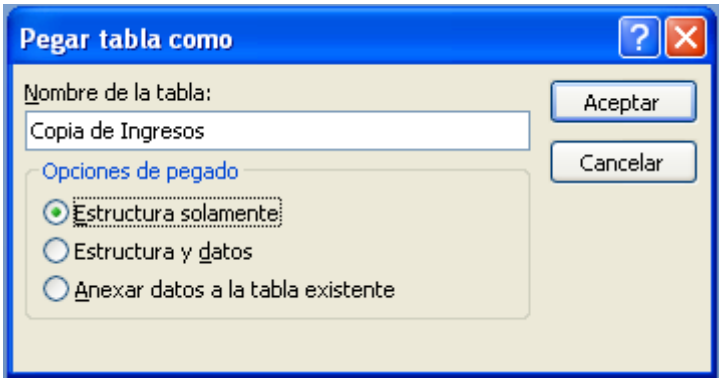


3.1.5. Consultas de acción

Las consultas de acción permiten eliminar, añadir y modificar filas de tablas.

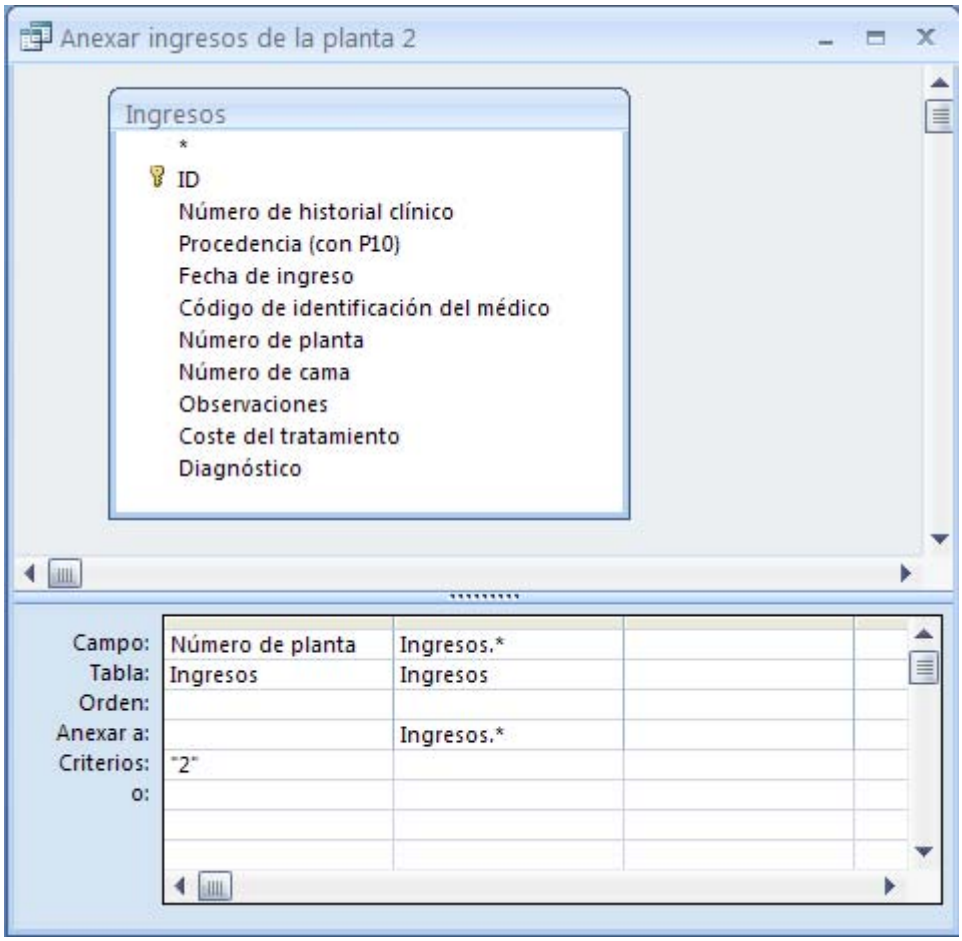
3.1.5.1. Anexar ingresos de la planta 2

El objetivo es insertar (anexar) las filas de la tabla Ingresos correspondientes a la planta 2 en una nueva tabla denominada Copia de Ingresos. Para ello se debe copiar en primer lugar la tabla Ingresos en la nueva tabla. Al pulsar Ctrl-C y Ctrl-V en Ingresos en el panel de la izquierda se crea esta tabla. Se debe seleccionar el botón de radio Estructura solamente:



A continuación se debe. crear una nueva consulta denominada "Anexar ingresos de la planta 2". Para especificar que se trata de una consulta de inserción de filas, hay que pulsar el botón Anexar:

Los pasos que se deben seguir son similares a las consultas anteriores. La diferencia se ilustra en la siguiente figura, en la que se observa que las filas de la rejilla QBE han cambiado como resultado de seleccionar esta consulta como de inserción, y así aparece la fila Anexar a. El criterio en este caso es la coincidencia del valor 2 para el campo Número de planta:

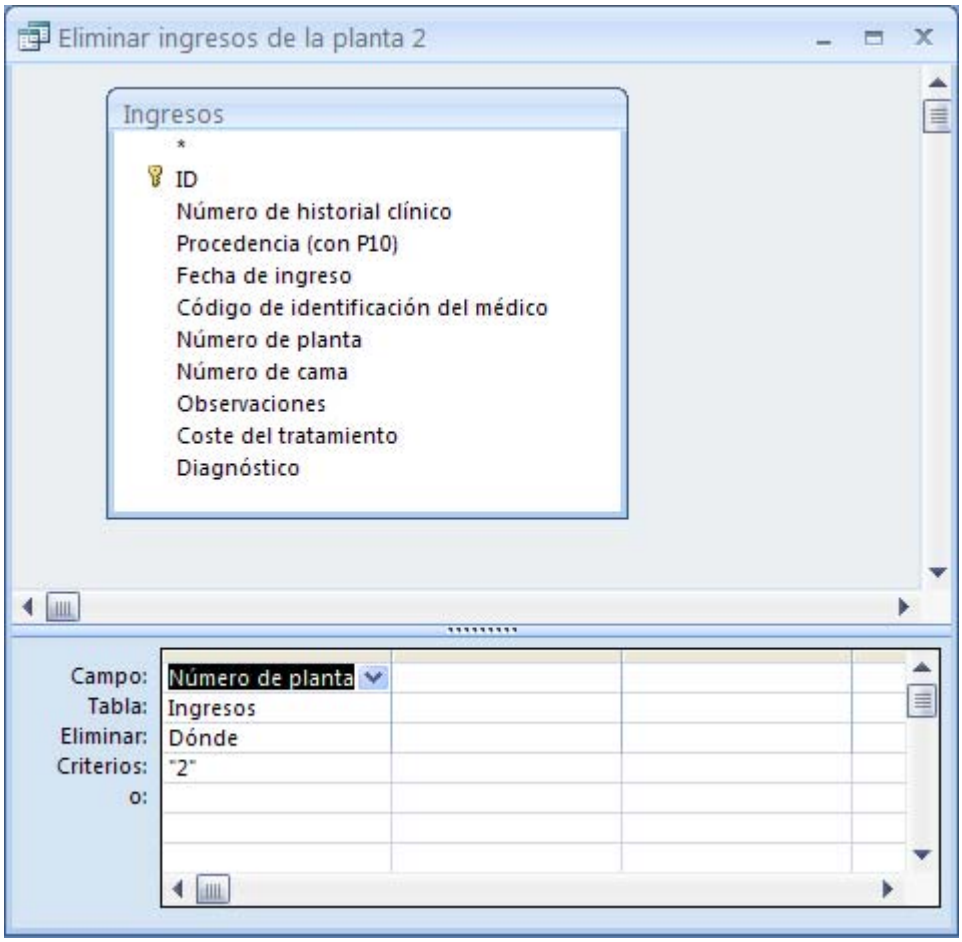


Para ejecutar esta consulta se debe pulsar el botón Ejecutar que se ilustra en la siguiente figura:

3.1.5.2. Eliminar ingresos de la planta 2

El objetivo es eliminar las filas de la tabla Ingresos correspondientes a la planta 2. Para ello se debe crear una nueva consulta denominada "Eliminar ingresos de la planta 2". Para especificar que se trata de una consulta de eliminación de filas, hay que pulsar el botón Eliminar:

Los pasos que se deben seguir son similares a las consultas anteriores. La diferencia se ilustra en la siguiente figura, en la que se observa que las filas de la rejilla QBE han cambiado como resultado de seleccionar esta consulta como de eliminación, y así aparece la fila Eliminar. El criterio en este caso es la coincidencia del valor 2 para el campo Número de planta:



Para ejecutar esta consulta se debe pulsar el botón Ejecutar como en el caso anterior.

3.1.5.3. Recuperar filas de Ingresos

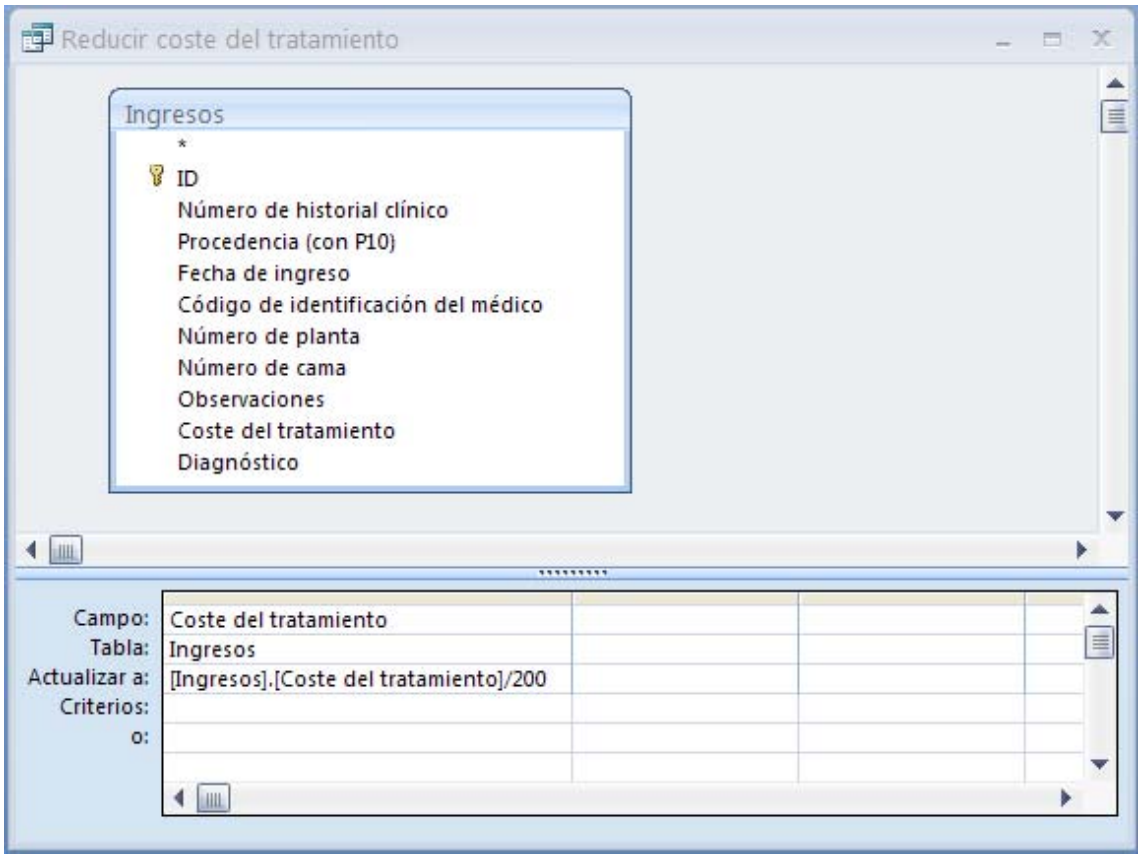
Para recuperar los datos eliminados de la tabla Ingresos hay que crear una nueva consulta de inserción como en el primer apartado de estas consultas de acción, que traslade las filas de la tabla Copia de Ingresos a la tabla Ingresos. En este caso no es necesario indicar ningún criterio puesto

que recuperaremos todas las filas de la tabla copiada. Esta consulta se debe denominar Recuperar filas de Ingresos.

3.1.5.4. Reducir coste del tratamiento

El objetivo es modificar la columna Coste del tratamiento de la tabla Ingresos. Para ello se debe crear una nueva consulta denominada "Reducir coste del tratamiento". Para especificar que se trata de una consulta de modificación (actualización) de filas, hay que pulsar el botón Actualizar:

Al igual que en los casos anteriores, el proceso de creación de la consulta es similar y se debe llegar a la siguiente configuración de la consulta en la vista Diseño:



Obsérvese la línea Actualizar a:. Contiene una referencia al mismo campo que se desea modificar (Coste del tratamiento) y está calificada con la notación [*Tabla*].[*Campo*]. Se puede escribir simplemente el nombre del campo y, si como es el caso, contiene espacios en blanco, se debe encerrar entre corchetes. El propio Access rellenará el resto de información. Si Access

cambia algo de lo escrito de forma que aparezca encerrado entre dobles comillas, significará que lo ha interpretado como un texto (una cadena constante) y no el nombre del campo. Si nos queremos referir a un campo como es el caso que nos ocupa, habría que corregirlo escribiendo los corchetes adecuadamente. Si Access nos lo impide, podemos escribirlo en cualquier editor de texto (Bloc de notas, por ejemplo), y después copiarlo y pegarlo en Access.

Tema 3.2. SQL: el lenguaje de programación de bases de datos relacionales

SQL es un lenguaje de consulta que utilizan la práctica totalidad de sistemas de gestión de bases de datos relacionales. Su última versión es SQL: 1999. A diferencia de la implementación gráfica de QBE, es un lenguaje textual como la mayoría de lenguajes de programación.

Aunque el lenguaje SQL se considere un lenguaje de consulta, contiene muchas otras capacidades además de la consulta en bases de datos. Incluye características para definir la estructura de los datos, para la modificación de los datos en la base de datos y para la especificación de restricciones de integridad y de acceso (seguridad).

IBM desarrolló la versión original en su Laboratorio de investigación de San José (*San José Research Center*, actualmente Centro de investigación de Almadén, *Almadén Research Center*). IBM implementó el lenguaje, originalmente denominado Sequel, como parte del proyecto System R, a principios de 1970. El lenguaje Sequel ha evolucionado desde entonces y su nombre ha pasado a ser SQL (*Structured Query Language*, Lenguaje estructurado de consultas). Actualmente, numerosos productos son compatibles con el lenguaje SQL. SQL se ha establecido como *el* lenguaje estándar de bases de datos relacionales.

En 1986, ANSI (*American National Standards Institute*, Instituto nacional americano de normalización) e ISO (*International Standards Organization*, Organización internacional de normalización), publicaron una norma SQL, denominada SQL-86. En 1987, IBM publicó su propia norma de SQL corporativo: Interfaz de bases de datos para arquitecturas de aplicación a sistemas (*Systems Application Architecture Database Interface*, SAA-SQL). En 1989 se publicó una norma extendida para SQL denominada SQL-89 y actualmente los sistemas de bases de datos son normalmente compatibles al menos con las características de SQL-89. La siguiente versión de la norma fue SQL-92 y más reciente es SQL:1999.

El lenguaje SQL tiene tres partes diferenciadas:

- Lenguaje de manipulación de datos. Es la parte del lenguaje que permite extraer datos de la base de datos y modificarlos. Es el equivalente a las consultas de selección y modificación que se vieron en el tema anterior.
- Lenguaje de definición de datos. Es la parte del lenguaje que permite crear el esquema de la base de datos. Es el equivalente a la posibilidad de crear tablas que se vio en el módulo anterior.
- Lenguaje de acceso a datos. Es la parte del lenguaje que permite definir la seguridad de acceso a los datos, determinando qué usuarios tienen acceso a qué objetos del sistema de bases de datos (tablas, campos, ...). Access no incorpora esta parte del lenguaje, aunque Oracle sí, y se estudiará en el siguiente módulo.

En los dos siguientes apartados se estudiarán el lenguaje de manipulación de datos y el lenguaje de definición de datos.

3.2.1. Lenguaje de manipulación de datos

En este apartado se estudiarán en primer lugar las sentencias de extracción de datos y, en segundo, las que permiten modificar los datos (añadiendo, borrando o alterando datos).

3.2.1.1. Sentencia SELECT

La sentencia SELECT es la más utilizada en el acceso a bases de datos. Su objetivo es seleccionar filas y columnas de una o varias tablas con respecto a los criterios que se especifiquen. Se conoce como sentencia (o consulta) de selección porque permite seleccionar los campos y filas a extraer de la base de datos.

3.2.1.1.1. Forma básica de SELECT

La forma básica de la instrucción SELECT es como se muestra a continuación:

```
SELECT  ListaDeAtributos
FROM   ListaDeTablas
WHERE  Condición;
```

donde ListaDeAtributos es la lista de los atributos (campos) que interesa recuperar de las tablas ListaDeTablas, cumpliendo la condición especificada en Condición. La cláusula WHERE es opcional, pero la parte SELECT y FROM son obligatorias. Si no se proporciona esta cláusula, se extraerán todas las filas sin filtrar ninguna (equivale a usar WHERE TRUE, es decir, la condición de selección es siempre cierta, así que no se descarta ninguna fila).

Por ejemplo, una consulta básica de selección que extrae la lista de médicos adjuntos sería:

```
SELECT *
FROM Médicos
WHERE Cargo='Adjunto';
```

Nótese que se ha usado el asterisco (*) para indicar que se desean extraer todas las columnas.

Su resultado es:

Código de identificación del médico	Nombre	Apellidos	Especialidad	Número de colegiado	Cargo
AMG1	Alejandro	Martí Giménez	Medicina General	2354	Adjunto
MRSN	María Rosa	Sánchez Navarro	Análisis Clínicos	1214	Adjunto

En esta consulta se ha usado una condición lógica que es cierta para todas las filas cuyo valor del campo Cargo sea 'Adjunto'.

Las condiciones lógicas pueden ser más elaboradas, conteniendo los siguientes tipos de operadores:

- Operadores lógicos
  - AND. Conjunción lógica. Evalúa dos condiciones y devuelve el valor cierto sólo si sus argumentos son ciertos.
  - OR. Disyunción lógica. Evalúa dos condiciones y devuelve el valor cierto si alguno de sus argumentos es cierto.
  - NOT. Negación lógica. Devuelve el complemento lógico de su argumento.
- Operadores relacionales:
  - <. Menor que.
  - >. Mayor que.
  - <>. Distinto de.
  - <=. Menor o igual que.
  - >=. Mayor o igual que.
  - = Igual a.
- Otros operadores:
  - BETWEEN. Se usa para especificar un intervalo de valores.
  - LIKE. Se usa la comparación mediante patrones (con comodines).
  - IN. Se usa para determinar la pertenencia a un conjunto.

Por ejemplo, la siguiente consulta extrae las filas de la tabla Ingresos cuyo coste se encuentra comprendido entre 50 y 100, seleccionando sólo los campos Número de historial clínico y Coste del tratamiento. Obsérvese que estos campos se encierran entre corchetes porque contienen el carácter espacio.

```
SELECT  [Número de historial clínico], [Coste del tratamiento]
FROM Ingresos
WHERE [Coste del tratamiento]>=50 AND [Coste del tratamiento] <=100;
```

Esta primera versión usa el operador lógico AND y los relacionales <= y >= para definir el intervalo de valores de interés. La siguiente versión es más reducida al usar el operador BETWEEN, pero ambas son equivalentes:

```
SELECT  [Número de historial clínico], [Coste del tratamiento]
FROM Ingresos
WHERE [Coste del tratamiento] BETWEEN 50 AND 100;
```

Su resultado es:

Número de historial clínico	Coste del tratamiento
3	75,00
2	100,00
1	75,00
5	100,00
4	100,00
1	75,00

Las consultas SQL pueden devolver en general valores repetidos. En ocasiones esto es útil, por ejemplo cuando se desean usar funciones de agrupación como el promedio, pero en otras es conveniente descartar los valores repetidos. Si se desea el descarte de valores repetidos hay que usar la cláusula DISTINCT después de SELECT, como se muestra en el siguiente ejemplo:

```
SELECT DISTINCT [Número de historial clínico] , [Coste del
tratamiento]
FROM Ingresos
WHERE [Coste del tratamiento] BETWEEN 50 AND 100;
```

Su resultado es:

Número de historial clínico	Coste del tratamiento
1	75,00
2	100,00
3	75,00

4	100,00
5	100,00

En ocasiones también resulta útil extraer las filas distintas, independientemente de cuáles sean los campos de la lista de selección. Para ello hay que usar la cláusula DISTINCTROW. Nótese que esto sólo es útil cuando la tabla contenga filas duplicadas, es decir, una tabla sin clave principal definida. De otro modo, no habría posibilidad de encontrar filas duplicadas a descartar en el resultado.

Con el siguiente ejemplo se usa el operador LIKE para obtener los médicos cuyo nombre comienza por M.

```
SELECT Nombre FROM Médicos
WHERE Nombre LIKE 'M*' ;
```

Su resultado es:

Nombre
María Rosa
Manuel

Si es necesario hacer una comparación con una fecha hay que escribirla entre almohadillas, primero el mes, luego el día y finalmente el año, como se muestra en el siguiente ejemplo:

Ej. Ingresos en la fecha 01/02/2010

```
SELECT *
FROM Ingresos
WHERE [Fecha de ingreso]=#02/01/2010#;
```

3.2.1.1.2. Consultas de varias tablas

La sentencia SELECT permite extraer datos de más de una tabla de forma que se obtengan los datos relacionados entre varias tablas. Estas relaciones son generalmente las que se definen entre las tablas y en las que se imponen restricciones de integridad referencial.

Por ejemplo, como se vio en la definición de la base de datos Hospital, la tabla Ingresos está relacionada con la tabla Pacientes, indicando que a cada ingreso le corresponde un paciente en concreto según el campo Número de historial clínico. Si deseamos conocer las fechas de ingreso de todos los pacientes se debería emitir una consulta como la siguiente:

Ej. Ingresos de pacientes

```
SELECT [Nombre del paciente], [Apellidos del paciente], [Fecha de
ingreso]
```

```
FROM Pacientes, Ingresos
WHERE Pacientes.[Número de historial clínico]=Ingresos.[Número de
historial clínico];
```

Nótese que para obtener la respuesta adecuada se indican en la cláusula FROM las dos tablas (Pacientes e Ingresos) y se impone como condición de la cláusula WHERE que se extraigan las filas de ambas tablas que coincidan en el atributo Número de historial clínico. Para ello se identifica cada atributo indicando a qué tabla nos referimos, anteponiendo el nombre de la tabla al nombre del atributo y separándolos con un punto.

El resultado de la consulta se muestra a continuación:

Nombre del paciente	Apellidos del paciente	Fecha de ingreso
Víctor	García Montoya	01/02/2010
Víctor	García Montoya	10/02/2010
Víctor	García Montoya	30/08/2010
Víctor	García Montoya	13/02/2010
José Antonio	Pérez Gómez	01/02/2010
José Antonio	Pérez Gómez	11/11/2010
José Antonio	Pérez Gómez	21/11/2010
Juan	Pérez Cayuela	12/04/2010
Juan	Pérez Cayuela	03/03/2010
Juan	Pérez Cayuela	30/03/2010
Ana María	Velasco Sánchez	12/04/2010
Ana María	Velasco Sánchez	08/06/2010
Ana María	Velasco Sánchez	11/11/2010

Hay que ser cuidadoso con el empleo de listas de tablas porque el proceso que el motor de bases de datos realiza es costoso, y este coste aumenta exponencialmente con el número de tablas implicadas. El motivo es que resulta necesario aplicar la operación producto cartesiano entre las filas de las tablas implicadas. Esta operación genera todas las posibles combinaciones entre las filas. De ellas se filtran las que cumplen la condición de selección.

Por ejemplo, dadas las tablas:

A

CA
1
2

B

CB
3
4

C

CC
5
6

La instrucción:

```
SELECT *
FROM A,B;
```

Da como resultado:

CA	CB
1	3
1	4
2	3
2	4

Y la instrucción:

```
SELECT *
FROM A,B,C;
```

CA	CB	CC
1	3	5
1	3	6
1	4	5
1	4	6
2	3	5
2	3	6
2	4	5
2	4	6

En general, para las tablas T1, ..., TN, el resultado de su producto cartesiano T1 × ... × TN tiene un tamaño |T1| * ... * |TN|, donde las barras verticales denotan la cardinalidad de la tabla (número de elementos).

3.2.1.1.3. Cláusula ORDER BY

La cláusula ORDER BY se usa para especificar el orden en que se muestran las filas resultado de una consulta. El orden puede ser ascendente (el predeterminado) o descendente. Hay que especificar la cláusula DESC después del atributo por el que se quiere ordenar para indicar una ordenación descendente. Se puede especificar una lista de atributos según los cuales se ordene.

Ej. Ingresos de pacientes ordenados por apellido

```
SELECT [Nombre del paciente], [Apellidos del paciente], [Fecha de
ingreso]
FROM Pacientes, Ingresos
WHERE Pacientes.[Número de historial clínico]=Ingresos.[Número de
historial clínico]
ORDER BY [Apellidos del paciente];
```

Nombre del paciente	Apellidos del paciente	Fecha de ingreso
Isabel	García Martínez	08/06/2010
Isabel	García Martínez	23/05/2010
Víctor	García Montoya	13/02/2010
Víctor	García Montoya	30/08/2010
Víctor	García Montoya	10/02/2010
Víctor	García Montoya	01/02/2010
Juan	Pérez Cayuela	30/03/2010
Juan	Pérez Cayuela	03/03/2010
Juan	Pérez Cayuela	12/04/2010
José Antonio	Pérez Gómez	21/11/2010
José Antonio	Pérez Gómez	11/11/2010
José Antonio	Pérez Gómez	01/02/2010
Ana María	Velasco Sánchez	11/11/2010

3.2.1.1.4. Cláusula GROUP BY

Esta cláusula se usa cuando es necesario realizar cálculos sobre grupos de datos. La idea es agrupar el conjunto de resultados de una instrucción SELECT en subconjuntos formados por las tuplas de este conjunto tales que coincidan sus valores de una lista de campos especificada después de la cláusula GROUP BY. A cada subconjunto se le aplica una

función de agregación para calcular operaciones como la suma (SUM), el promedio (AVG), el máximo (MAX), el mínimo (MIN) y el recuento de elementos (COUNT). Las cuatro operaciones primeras (SUM, AVG, MAX y MIN) requieren como parámetro el nombre de un campo por el que realizar la operación correspondiente. La operación COUNT se usa habitualmente como COUNT(*), que realiza el recuento de todas las tuplas de una tabla. También se puede indicar como parámetro un campo en lugar del asterisco. En este caso realiza la cuenta de todas las tuplas con un valor en ese campo distinto de NULL.

Ej. Coste total de tratamientos

```
SELECT SUM([Coste del tratamiento]) AS [Suma del coste]
FROM Ingresos;
```

Suma del coste
295.000 pta

Ej. Coste total de tratamientos por médico

```
SELECT SUM([Coste del tratamiento]) AS [Suma del coste]
FROM Ingresos
GROUP BY [Código de identificación del médico];
```

Código de identificación del médico	Suma del coste
AMG1	77.000 pta
FPO1	129.000 pta
MRSN	67.000 pta
MSM1	22.000 pta

Ej. Recuento de médicos

```
SELECT COUNT(*) AS [Número de médicos]
FROM Médicos;
```

El resultado es:

Número de médicos
5

Ej. Recuento de médicos con fotografía

```
SELECT COUNT(Fotografía) AS [Número de médicos con fotografía]
FROM Médicos;
```

Número de médicos con fotografía
0

Otros SGBDR permiten el uso de DISTINCT como parámetro de COUNT, lo que permite determinar el número tuplas con valores distintos de un campo en concreto. Por ejemplo:

```
SELECT COUNT(DISTINCT [Número de historial clínico])
FROM Ingresos;
```

Sin embargo, Access 2000 no permite este uso. Para solucionarlo se emitiría una consulta como la siguiente:

Ej. Recuento de pacientes con ingresos

```
SELECT COUNT(*)
FROM
(SELECT DISTINCT [Número de historial clínico]
 FROM Ingresos);
```

Número de pacientes con ingresos
5

Cláusula HAVING

La cláusula HAVING indica que en el resultado de la consulta se incluyan sólo ciertos grupos producidos por la cláusula GROUP BY. Al igual que la cláusula WHERE, utiliza una condición de búsqueda para especificar los grupos deseados. En otras palabras, especifica la condición que deben cumplir los grupos. La condición incluye generalmente funciones de agregación. La sintaxis de la cláusula HAVING es:

HAVING Expresión1 Operador Expresión2

Donde Expresión1 y Expresión2 pueden ser nombres de campos, valores constantes o expresiones y no deben coincidir con una expresión de columna en la cláusula SELECT. Operador es un operador relacional que compara ambas expresiones.

Ej. Sumas de costes por tratamiento mayores que 200

```
SELECT Diagnóstico,
                SUM([Coste del tratamiento]) AS [Suma del coste]
FROM Ingresos
GROUP BY Diagnóstico
HAVING SUM([Coste del tratamiento])>200;
```

Su resultado es:

Diagnóstico	Suma del coste
Infarto	750,00
Insuficiencia renal	300,00
Neumonía	225,00

3.2.1.1.5. Cláusula JOIN (combinación de relaciones)

La cláusula JOIN se usa para extraer datos de dos tablas de forma parecida a como hace la instrucción SELECT con una lista de dos tablas. De hecho, en su versión más simple, realiza la misma función. En otros sistemas, como Oracle, la consulta resulta de expresión más concisa.

Microsoft denomina a la operación JOIN como combinación, aunque no es una traducción consensuada (se pueden encontrar las traducciones reunión, unión, ...).

Hay dos tipos de combinaciones: la interna y la externa.

Combinación interna (INNER JOIN)

La combinación interna funciona de forma análoga a la instrucción SELECT con listas de tablas y condición.

Ej. Médicos que atienden ingresos

```
SELECT Médicos.[Código de identificación del médico], Diagnóstico,
[Fecha de ingreso]
FROM Médicos INNER JOIN Ingresos
ON Médicos.[Código de identificación del médico] = Ingresos.[Código de
identificación del médico];
```

Que es equivalente a la siguiente:

```
SELECT Médicos.[Código de identificación del médico], Diagnóstico,
[Fecha de ingreso]
FROM Médicos, Ingresos
WHERE Médicos.[Código de identificación del médico] = Ingresos.[Código
de identificación del médico];
```

Y su resultado es:

Código de identificación del médico	Diagnóstico	Fecha de ingreso
AMG1	Fiebres altas	23/05/2010
AMG1	Neumonía	01/02/2010
AMG1	Gastroenteritis	11/11/2010
AMG1	Infarto	03/03/2010
FPO1	Gastroenteritis	08/06/2010
FPO1	Infarto	11/11/2010
FPO1	Gastroenteritis	30/03/2010
FPO1	Infarto	13/02/2010
FPO1	Neumonía	21/11/2010
MRSN	Insuficiencia renal	12/04/2010
MRSN	Gastroenteritis	10/02/2010

Combinación externa por la izquierda (LEFT OUTER JOIN)

El segundo tipo de combinación es la combinación externa de la que, a su vez, podemos encontrar varios tipos: por la izquierda, por la derecha y completa. La combinación externa por la izquierda toma dos tablas y cada tupla del resultado está formada por una tupla de la tabla de la izquierda a la que añaden las columnas de la tupla de la tabla de la derecha que verifica la condición de combinación (ON). Si no hay ninguna tupla de la tabla de la izquierda que la verifique, entonces los campos en el resultado correspondientes a la tabla de la derecha se dejan con nulos.

En el siguiente ejemplo se extraen los diagnósticos emitidos por los médicos y sus fechas.

Ej. Diagnósticos de médicos

```
SELECT Médicos.[Código de identificación del médico], Diagnóstico,
[Fecha de ingreso]
FROM Médicos LEFT OUTER JOIN Ingresos
ON Médicos.[Código de identificación del médico] = Ingresos.[Código de
identificación del médico];
```

Código de identificación del médico	Diagnóstico	Fecha de ingreso
AMG1	Fiebres altas	23/05/2010
AMG1	Neumonía	01/02/2010
...	...	...
MSM1	Neumonía	01/02/2010
MSM1	Gastroenteritis	30/08/2010
MSM2		

Como se puede observar, hay un médico que no ha emitido ningún diagnóstico y, por tanto, los valores correspondientes de la tabla Ingresos aparecen vacíos (valores NULL).

Combinación externa por la derecha (RIGHT OUTER JOIN)

La combinación externa por la derecha es análoga a la combinación externa por la izquierda, pero permutando el orden de las tablas. Con una de las versiones de la combinación externa se consigue la misma funcionalidad de la otra.

Combinación externa completa

Access no incluye la combinación externa completa, aunque Oracle sí. No obstante, se puede simular con los otros tipos de combinaciones, aunque a costa de una expresión más complicada.

3.2.1.1.6. Subconsultas

Las subconsultas son consultas anidadas que pueden formar parte de otras consultas. Las subconsultas se pueden encontrar como origen de datos en una cláusula FROM o en las cláusulas IN y EXISTS.

Subconsultas en la cláusula FROM

Con el siguiente ejemplo se extraen los médicos que atienden los ingresos del paciente con número de historial clínico 1.

Ej. Médicos que atienden al paciente 1

```
SELECT *
FROM Médicos, (SELECT * FROM Ingresos
                              WHERE [Número de historial clínico]='1')
AS I
WHERE Médicos.[Código de identificación del médico] =
               I.[Código de identificación del médico];
```

Subconsultas en la cláusula IN

La cláusula IN puede formar parte de la condición WHERE para especificar la pertenencia a conjuntos. La sintaxis de la sentencia SELECT con la cláusula IN es la siguiente:

```
SELECT  ListaDeAtributos
FROM   ListaDeTablas
WHERE  Atributo IN ListaValores;
```

Por ejemplo, la siguiente consulta extrae todos los médicos cuyos cargos son Adjunto o Jefe de planta.

Ej. Médicos adjuntos o jefes de planta

```
SELECT  Nombre
FROM   Médicos
WHERE  Cargo IN ('Adjunto', 'Jefe de planta');
```

Esta consulta es equivalente a la siguiente:

```
SELECT  Nombre
FROM   Médicos
WHERE  Cargo = 'Adjunto' OR Cargo = 'Jefe de planta';
```

pero es más concisa (considérese que podría haber muchos valores por los que estuviésemos interesados en filtrar).

Otro posible uso de la cláusula IN es proporcionarle el conjunto de valores posibles a través de una consulta SELECT, según la siguiente sintaxis:

```
SELECT  ListaDeAtributos
FROM   ListaDeTablas
WHERE  Atributo IN ConsultaDeSelección;
```

En el siguiente ejemplo se muestran los médicos que finalmente se borrarán con una consulta de borrado.

Ej. Médicos a borrar según Especialidades a borrar

```
SELECT *
FROM Médicos
WHERE Médicos.[Especialidad] IN
(SELECT [Especialidad]
 FROM [Especialidades a borrar]);
```

Subconsultas en la cláusula EXISTS

La cláusula EXISTS puede formar parte de la condición en una instrucción SELECT. Es una condición lógica que es cierta si la consulta sobre la que se aplica devuelve alguna fila. Su sintaxis es como se indica a continuación:

```
SELECT  ListaDeAtributos
FROM   ListaDeTablas
WHERE  EXISTS ConsultaDeSelección;
```

También puede formar parte de una expresión lógica, como NOT EXISTS.

En el siguiente ejemplo se piden todos los médicos que no tengan ingresos asignados.

Ej. Médicos sin asignación

```
SELECT *
FROM Médicos
WHERE NOT EXISTS
(SELECT * FROM Ingresos WHERE Ingresos.[Código de identificación del
médico] = Médicos.[Código de identificación del médico]);
```

Subconsultas en expresiones

Las expresiones lógicas de la cláusula WHERE pueden contener relaciones resultados de una subconsulta. En el siguiente ejemplo se muestra cómo obtener un dato mediante una subconsulta que se compara con otro procedente de una fila.

Ej. Determinar si el paciente 1 ha ingresado después del *último* ingreso del paciente 2

```
SELECT [Fecha de ingreso]
FROM Ingresos
WHERE [Número de historial clínico]='1' AND
       [Fecha de ingreso] > (SELECT MAX([Fecha de ingreso])
FROM Ingresos
WHERE [Número de historial clínico]='2');
```

Una cláusula útil que se puede usar en estas subconsultas es ALL. Por ejemplo, la consulta anterior se puede reescribir como:

```
SELECT [Fecha de ingreso]
FROM Ingresos
WHERE [Número de historial clínico]='1' AND
   [Fecha de ingreso] > ALL(SELECT [Fecha de ingreso]
FROM Ingresos
WHERE [Número de historial clínico]='2');
```

Finalmente también se puede usar ANY (o su alias SOME), como en el siguiente ejemplo:

Ej. Determinar si el paciente 1 ha ingresado después de *algún* ingreso del paciente 2

```
SELECT [Fecha de ingreso]
FROM Ingresos
WHERE [Número de historial clínico]='1' AND
   [Fecha de ingreso] > ANY (SELECT [Fecha de ingreso]
                FROM Ingresos
          WHERE [Número de historial clínico]='2');
```

3.2.1.1.7. Consultas de conjuntos: UNION

La sentencia UNION se usa para realizar la unión de conjuntos (y multiconjuntos) de dos fuentes de datos. Sus argumentos son dos fuentes de datos que deben tener el mismo esquema y el resultado es la unión de las filas de ambas fuentes. Si se especifica UNION ALL en el resultado se encuentran todas las filas de ambas fuentes, incluso si hay duplicados.

Si necesitamos un listado de todas las personas relacionadas con el hospital (ya sean pacientes o médicos) se puede hacer una unión de dos consultas: una que extraiga los datos de los médicos y otra la de los pacientes.

Ej: Personas relacionadas con el hospital.

```
(SELECT Nombre, Apellidos
 FROM Médicos)
UNION
(SELECT [Nombre del paciente], [Apellidos del paciente]
 FROM Pacientes);
```

3.2.1.1.8. Consultas de tablas de referencias cruzadas: la cláusula TRANSFORM

La sentencia TRANSFORM permite especificar consultas de tablas de referencias cruzadas, como se vieron en el tema de creación de consultas en QBE.

La sintaxis para este tipo de consulta es la siguiente:

```
TRANSFORM FunciónAgregada InstrucciónSelect PIVOT CampoPivote
    [IN (Cabecera1[,Cabecera2[, ...]])]
```

Donde FunciónAgregada es una función SQL agregada que opera sobre los datos seleccionados. InstrucciónSelect es una instrucción SELECT que se ocupa de la selección de los datos. CampoPivote es un campo o expresión que se desea utilizar para crear las cabeceras de la columna en el resultado de la consulta. Finalmente, Cabecera*i* son valores fijos utilizados para crear las cabeceras de la columna.

Para resumir datos utilizando una consulta de referencia cruzada se seleccionan los valores de los campos o expresiones especificadas como

cabeceras de columnas de tal forma que pueden verse los datos en un formato más compacto que con una consulta de selección.

TRANSFORM es opcional pero si se incluye es la primera instrucción de una cadena SQL. Precede a la instrucción SELECT que especifica los campos utilizados como encabezados de fila y una cláusula GROUP BY que especifica el agrupamiento de las filas. Opcionalmente puede incluir otras cláusulas como por ejemplo WHERE, que especifica una selección adicional o un criterio de ordenación .

Los valores devueltos en el campo pivote se utilizan como cabeceras de columna en el resultado de la consulta. Por ejemplo, al utilizar las cifras de ventas en el mes de la venta como pivote en una consulta de referencia cruzada se crearían 12 columnas. Se puede restringir el campo pivote para crear cabeceras a partir de los valores fijos (valor1, valor2) listados en la cláusula opcional IN.

También puede incluir valores fijos, para los que no existen datos, con el objetivo de crear columnas adicionales.

Ejemplos

```
TRANSFORM Sum(Cantidad) AS Ventas SELECT Producto, Cantidad FROM
Pedidos WHERE Fecha Between #01-01-2010# And  #12-31-2010# GROUP BY
Producto
ORDER BY Producto PIVOT DatePart("m", Fecha);
```

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por mes para un año específico. Los meses aparecen de izquierda a derecha como columnas y los nombres de los productos aparecen de arriba hacia abajo como filas.

```
TRANSFORM Sum(Cantidad) AS Ventas SELECT Compania FROM Pedidos
WHERE Fecha Between #01-01-2010# And  #12-31-2010# GROUP BY Compania
ORDER BY Compania PIVOT "Trimestre " & DatePart("q", Fecha) In
('Trimestre1', 'Trimestre2', 'Trimestre 3', 'Trimestre 4');
```

Crea una consulta de tabla de referencias cruzadas que muestra las ventas de productos por trimestre de cada proveedor en el año indicado. Los trimestres aparecen de izquierda a derecha como columnas y los nombres de los proveedores aparecen de arriba hacia abajo como filas.

3.2.1.1.9. Acceso a bases de datos externas

Para el acceso a bases de datos externas se utiliza la cláusula IN. Se puede acceder a una base de datos Access distinta de la que esté abierta en Access, e incluso a bases e datos de dBase, Paradox o Btrieve. Esta cláusula sólo permite la conexión de una base de datos externa a la vez. No obstante, para mejorar el rendimientos es mejor adjuntar las tablas de las bases de datos externas a la base de datos actual y trabajar con ellas.

Para especificar una base de datos que no pertenece a Access se agrega un punto y coma (;) al nombre y se encierra entre comillas simples. También puede utilizar la palabra reservada DATABASE para especificar la

base de datos externa. Por ejemplo, las líneas siguientes especifican la misma tabla:

```
FROM Tabla IN '[dBASE IV; DATABASE=C:\DBASE\DATOS\VENTAS;]';
FROM Tabla IN 'C:\DBASE\DATOS\VENTAS' 'dBASE IV;'
```

Ej. Acceso a una base de datos externa de Microsoft Access.

```
SELECT IDCliente FROM Clientes IN MISDATOS.MDB WHERE IDCliente Like
'A*';
```

Donde MISDATOS.MDB es el nombre de una base de datos de Microsoft Access que contiene la tabla Clientes.

Ej. Acceso a una base de datos externa de dBASE III o IV.

```
SELECT IDCliente FROM Clientes IN 'C:\DBASE\DATOS\VENTAS' 'dBASE IV';
    WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de dBASE III+ hay que utilizar 'dBASE III+;' en lugar de 'dBASE IV;'.

Ej. Acceso a una base de datos de Paradox 3.x o 4.x:

```
SELECT IDCliente FROM Clientes IN 'C:\PARADOX\DATOS\VENTAS'
    'Paradox 4.x;' WHERE IDCliente Like 'A*';
```

Para recuperar datos de una tabla de Paradox versión 3.x, hay que sustituir 'Paradox 4.x;' por 'Paradox 3.x;'.

Ej. Acceso a una base de datos de Btrieve:

```
SELECT IDCliente FROM Clientes IN 'C:\BTRIEVE\DATOS\VENTAS\FILE.DDF'
    'Btrieve;' WHERE IDCliente Like 'A*';
```

Donde C:\BTRIEVE\DATOS\VENTAS\FILE.DDF es el nombre completo (ruta y nombre de archivo) del archivo de definición de datos de Btrieve.

3.2.1.2. Expresiones en las consultas

Las consultas pueden incluir expresiones tanto en la lista de selección como en la condición. Las expresiones pueden ser aritméticas, de texto, de fechas, ... según del tipo de datos que se trate.

Un ejemplo de expresión aritmética es el siguiente, en el que se traducen a euros los importes especificados en pesetas:

```
SELECT  [Número de historial clínico], [Coste del tratamiento]/166.386
AS Euros
FROM Ingresos
WHERE [Coste del tratamiento]>=50 AND [Coste del tratamiento] <=100;
```

Número de historial clínico	Euros
3	0,450759078287837

2	0,601012104383782
1	0,450759078287837
5	0,601012104383782
4	0,601012104383782
1	0,450759078287837

En general se pueden usar todas las funciones disponibles en Visual Basic para construir expresiones. A continuación se muestran varios tipos de funciones que se pueden usar en expresiones según el tipo de datos que devuelven.

3.2.1.2.1. Función de selección IIF

IIF Devuelve uno de dos posibles valores. Los parámetros son tres: el primero es una expresión lógica, el segundo el valor a devolver si la expresión es cierta, y el último, el valor a devolver si es falsa. **IIF(SEXO='V','Masculino','Femenino')** devolvería la palabra 'Masculino' si el campo sexo contiene 'V' y caso contrario devolvería la palabra 'Femenino'.

Esta función se puede aplicar a los formularios, como se ve en el tema dedicado a ellos. Veamos los siguientes ejemplos:

Supongamos que en un formulario tenemos una casilla de texto llamada tbApellido. Si cuando ejecutamos esta consulta la casilla contiene algún valor se devuelven todos los empleados cuyo apellido coincida con el texto de la casilla, en caso contrario se devuelven todos los empleados.

```
SELECT * Total
FROM Empleados
WHERE Apellido = IFF(tbApellido.Text < '', tbApellido.Text, *) ;
```

La siguiente consulta devuelve los campos Fecha, Nombre del Producto y Cantidad de la tabla Pedidos, añadiendo un campo al final con el valor Madrid si el código postal está dentro del intervalo, en caso contrario devuelve Nacional.

```
SELECT Fecha, Producto, Cantidad,
(IIF(CodigoPostal=28000 AND CodigoPostal <=28999,'Madrid','Nacional'))
AS Destino
FROM Pedidos;
```

3.2.1.2.2. Funciones de caracteres

- **LEN** Devuelve la longitud de una cadena.

  LEN('Cadena') devuelve 6

- **&** Operador de concatenación de cadenas de caracteres.

  Nombre & " " & Apellidos construye el nombre y los apellidos a partir de los campos Nombre y Apellidos, separándolos con un espacio en blanco.

3.2.1.2.3. Funciones de fechas y horas

Relativas a fechas

- **NOW** Devuelve la fecha y hora actual del sistema.
- **DAY** Devuelve el día de una fecha.

  DAY(#01/30/89#) devuelve 30
- **MONTH** Devuelve el mes en cifras de un fecha.

  MONTH(#01/30/89#) devuelve 1
- **YEAR** Devuelve el año, con todas sus cifras, de una fecha.

  YEAR(#01/30/89#) devuelve 1989.
- **WEEKDAY** Devuelve un número entero que representa el día de la semana del parámetro fecha.

  WEEKDAY(#25/06/03#) devuelve 3
- **DATEPART** devuelve una parte en concreto de una fecha concreta (el mes, el año, ...) en el formato especificado en una cadena de caracteres. Su sintaxis es:

  DatePart(Parte, Fecha[, ComienzoSemana, ComienzoAño])

  Parte indica la parte y el formato de la fecha que se desea devolver y puede tomar los siguientes valores:

Valor	**Descripción**
yyyy	Año
q	Trimestre
m	Mes
y	Día del año
d	Día del mes
w	Día de la semana
ww	Semana del año
h	Hora
m	Minutos
s	Segundos

  ComienzoSemana indica el primer día de la semana. Los posibles valores son:

Valor	**Descripción**
0	Utiliza el valor predeterminado del sistema
1	Domingo (es el valor predeterminado)
2	Lunes
3	Martes

4	Miércoles
5	Jueves
6	Viernes
7	Sábado

ComienzoAño indica cual es la primera semana del año; los posibles valores son:

Valor	**Descripción**
0	Valor del sistema
1	Comienza el año el 1 de enero (valor predeterminado).
2	Empieza con la semana que tenga al menos cuatro días en el nuevo año.
3	Empieza con la semana que esté contenida completamente en el nuevo año.

3.2.1.2.4. Funciones aritméticas

- **MOD** Divide dos números y devuelve el resto de la división.

  10 MOD 3 devuelve 1.
- **^** Eleva un número a una potencia.

  2 ^ 3 devuelve 8
- **INT** Devuelve la parte entera de un número.

  INT(6.4321) devuelve 6
- **ROUND** Redondea un número.

  ROUND(123.456,0) devuelve 123

  ROUND(123.456,2) devuelve 123.46

  ROUND(123.456,-2) devuelve 100

3.2.1.2.5. Resumen de la sintaxis de la instrucción SELECT

En este apartado se resume la sintaxis de la instrucción SELECT en notación BNF.

Una instrucción SELECT básica tiene la siguiente sintaxis:

```
SELECT ListaDeAtributosOExpresiones
FROM ListaDeOrígenesDeDatos
WHERE Condición;
```

Donde:

```
ListaDeAtributosOExpresiones = AtributoOExpresión{,
AtributoOExpresión}*
AtributoOExpresión = { Atributo | Expresión }
```

'Atributo' es el identificador de un campo. En general se escribe sólo el atributo, como por ejemplo en:

```
SELECT Apellidos, Nombre
FROM Médicos;
```

Si el mismo atributo aparece en más de una tabla hay que calificarlo con el nombre de la tabla. Por ejemplo:

```
SELECT Médicos.Apellidos, Pacientes.Apellidos
FROM Médicos, Pacientes;
```

Si el atributo contiene espacios hay que encerrarlo entre corchetes, como en:

```
SELECT [Código de identificación del médico]
FROM Médicos;
```

Se pueden dar las dos situaciones anteriores como en el siguiente ejemplo:

```
SELECT Médicos.[Código de identificación del médico]
FROM Médicos;
```

'Expresión" es una expresión construida con constantes (como 'Sí', 1000 o #31/12/2010#), atributos, funciones (como IIF) y operadores (como AND).

Por ejemplo, una expresión que determina si una persona es mayor de edad en el momento de su ingreso en el hospital es:

```
SELECT DISTINCT [Nombre del paciente], [Apellidos del paciente],
IIF(([Fecha de ingreso]-[Fecha de nacimiento])/365>=18,"Mayor de
edad", "Menor de edad") AS Situación
FROM Ingresos, Pacientes
WHERE Pacientes.[Número de historial clínico]=Ingresos.[Número de
historial clínico];
```

```
ListaDeOrígenesDeDatos = OrigenDeDatos{, OrigenDeDatos}*
OrigenDeDatos = { Tabla | ConsultaDeSelección |
ConsultaDeSelecciónAlmacenada}
```

Donde:

'Tabla' es el identificador de una tabla.

'ConsultaDeSelección' es una consulta SELECT.

'ConsultaDeSelecciónAlmacenada' es el identificador de una consulta almacenada previamente en la base de datos.

'Condición' es una expresión lógica que se construye con constantes, operadores lógicos y funciones.

Por ejemplo, la siguiente cláusula WHERE contiene una condición lógica. Devuelve los ingresos posteriores al 1/6/2010 que hayan tenido un diagnóstico de neumonía o fiebres altas.

```
SELECT [Fecha de ingreso], Diagnóstico
FROM  Ingresos
WHERE [Fecha de ingreso]>#06/01/2010# AND
      Diagnóstico IN ('Neumonía', 'Fiebres altas')
```

3.2.1.3. Sentencias INSERT, DELETE y UPDATE

Estas sentencias permiten la modificación de datos. Con INSERT se agregan tuplas a las tablas, con DELETE se borran y con UPDATE se cambian los valores de tuplas existentes.

3.2.1.3.1. Sentencia INSERT

La sintaxis básica de la sentencia INSERT permite insertar una tupla de valores en una tabla. La tupla de valores se escribe como una lista de valores (que deben coincidir con el tipo de los campos de la tabla en el orden que están definidos) separada por comas y encerrada entre paréntesis. La sintaxis básica es:

```
INSERT INTO NombreTabla
VALUES NuevaTupla
```

Ej. Insertar médico

```
INSERT INTO Médicos
VALUES ('MSM2','Roberto', 'Ríos', Pediatría', '2002', 'Adjunto',
                  NULL, NULL);
```

También es posible obtener los datos mediante una consulta SELECT que actúe como proveedor de datos. Por ejemplo, en el caso más simple:

```
INSERT INTO Médicos
SELECT 'MSM2', 'Roberto', 'Ríos', 'Pediatría', '2002', 'Adjunto',
NULL, NULL;
```

Otro ejemplo más general obtendría datos de otra tabla:

```
INSERT INTO Médicos
SELECT * FROM [Médicos nuevos];
```

3.2.1.3.2. Sentencia DELETE

A continuación se muestra la sintaxis de la cláusula DELETE:

```
DELETE FROM NombreTabla
WHERE Condición;
```

Con esta sentencia se borran las tuplas de la tabla NombreTabla que cumplan la condición Condición.

La parte WHERE es opcional y, si no se especifica, se borran todas las tuplas de la tabla (lo cual es equivalente a WHERE TRUE).

Ej. Borrar médico

```
DELETE *
FROM Médicos
WHERE [Código de identificación del médico]='MSM2';
```

Si se emite la siguiente consulta se borrarían todos los médicos de la tabla Médicos (al menos, todos aquellos cuya eliminación no viole las restricciones de integridad referencial).

```
DELETE *
FROM Médicos;
```

La cláusula WHERE admite consultas anidadas, como en el siguiente ejemplo, que borra todos los médicos cuya especialidad se encuentre en la tabla Especialidades a borrar.

Ej. Borrar médicos según Especialidades a borrar

```
DELETE *
FROM Médicos
WHERE Médicos.[Especialidad] IN (SELECT [Especialidad] FROM
[Especialidades a borrar]);
```

Si es necesario borrar tuplas según los valores simultáneos de dos campos, cabría escribir una consulta basada en la idea del ejemplo anterior como la siguiente:

```
DELETE *
FROM Médicos
WHERE (Médicos.[Especialidad], Médicos.Cargo) IN (SELECT
[Especialidad], Cargo FROM [Especialidades y cargos a borrar]);
```

Sin embargo, Access no permite tuplas en la condición WHERE (la norma SQL:1999 sí lo especifica). Hay que reescribir la consulta haciendo uso de la cláusula EXISTS.

Ej. Borrar médicos según Especialidades y cargos a borrar

```
DELETE *
FROM Médicos
WHERE EXISTS
(SELECT *
 FROM [Especialidades y cargos a borrar]
 WHERE Médicos.[Especialidad] = [Especialidades y cargos a
borrar].[Especialidad] AND Médicos.Cargo = [Especialidades y cargos a
borrar].Cargo);
```

3.2.1.3.3. Sentencia UPDATE

La sentencia UPDATE permite modificar los valores de los atributos de las tuplas que cumplan una determinada condición.

```
UPDATE tabla
SET atributo1 = valor1, ...., atributon = valorn
WHERE condición;
```

La parte WHERE es opcional y, si no se especifica, se modifican todas las tuplas de la tabla (lo cual es equivalente a WHERE TRUE).

En el siguiente ejemplo se modifica el cargo del médico cuyo código de identificación es MSM1.

Ej. Modificar médico

```
UPDATE Médicos SET Cargo = 'Jefe de sección'
WHERE [Código de identificación del médico] = 'MSM1';
```

Al igual que la sentencia DELETE, en la cláusula WHERE se admiten consultas anidadas, como en el siguiente ejemplo, que actualiza todos los médicos cuya especialidad se encuentre en la tabla Especialidades y cargos a modificar.

Ej. Modificar médicos según Especialidades y cargos a modificar

```
UPDATE Médicos SET Cargo = 'Jefe de sección'
WHERE EXISTS
(SELECT *
 FROM [Especialidades y cargos a modificar]
 WHERE Médicos.[Especialidad] = [Especialidades y cargos a
modificar].[Especialidad] AND Médicos.Cargo = [Especialidades y cargos
a modificar].Cargo);
```

3.2.2. Lenguaje de definición de datos

Con las consultas que se pueden escribir en el lenguaje de definición de datos es posible crear tablas, eliminarlas, modificarlas, añadirles restricciones de integridad y crear índices sobre sus campos.

En la creación de las tablas es necesario especificar los tipos de datos de los campos. En el siguiente apartado se indican los tipos de datos que se pueden manejar en SQL.

3.2.2.1. Tipos de datos

Hay trece tipos de datos primarios y varios sinónimos que pueden usarse en lugar de los primarios.

Los tipos de datos primarios se recogen en la siguiente tabla:

Tipo de datos	Longitud	Descripción
BINARY	1 byte	Tipo de datos binario.
BIT	1 byte	Valores Si/No ó True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automáticamente (de tipo Long)
CURRENCY	8 bytes	Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.

DATETIME	8 bytes	Un valor de fecha y hora entre los años 100 y 9999.
SINGLE	4 bytes	Un valor en coma flotante de precisión simple con un rango de -3.402823*10^{38} a -1.401298*10^{-45} para valores negativos, 1.401298*10^{-45} a 3.402823*10^{38} para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de -1.79769313486232*10^{308} a -4.94065645841247*10^{-324} para valores negativos, 4.94065645841247*10^{-324} a 1.79769313486232*10^{308} para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 2,147,483,647.
LONGTEXT	1 byte por carácter	De cero a un máximo de 1.2 gigabytes.
LONGBINARY	Según se necesite	De cero 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por caracter	De cero a 255 caracteres.

La siguiente tabla recoge los sinónimos de los tipos de datos anteriores:

Tipo de datos	**Sinónimos**
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT
CURRENCY	MONEY
DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEESINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEEDOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT
LONG	INT

	INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR CHARACTER STRING VARCHAR
VARIANT (No Admitido)	VALUE

3.2.2.2. Creación de tablas

La sintaxis de la consulta de creación de tablas es:

```
CREATE TABLE NombreTabla
  (Campo1 TipoCampo1 [(Tamaño)] [RestCampo1],
   Campo2 TipoCampo2 [(Tamaño)] [RestCampo2], ...,
   [Restricción1, Restricción2, ...] )
```

Donde:

- NombreTabla. Es el nombre de la tabla que se va a crear.
- Campo*i*. Es el nombre del campo o de los campos que se van a crear en la nueva tabla. La nueva tabla debe contener, al menos, un campo.
- Tipo. Es el tipo de datos de campo en la nueva tabla. Los tipos posibles son los indicados en el apartado anterior y pueden llevar asociado un tamaño expresado entre paréntesis.
- Tamaño. Es el tamaño del campo. Sólo se aplica para campos de tipo texto.
- RestCampo*i*. Es una cláusula CONSTRAINT (que se verá más adelante) que define una restricción sobre un único campo.
- Restricción*i*. Es una cláusula CONSTRAINT que define que define una restricción sobre varios campos.

Como ejemplo, considérese la siguiente consulta que crea una nueva tabla llamada Empleados con dos campos, uno llamado Nombre de tipo texto y longutid 25 y otro llamado apellidos con longitud 50.

```
CREATE TABLE Empleados (Nombre TEXT (25), Apellidos TEXT (50));
```

3.2.2.2.1. La cláusula CONSTRAINT

La cláusula CONSTRAINT se utiliza para definir restricciones de integridad de clave primaria, de superclave, de existencia y de integridad referencial.

La primera versión de esta cláusula se aplica a campos únicos y su sintaxis es:

```
[CONSTRAINT NombreRestricción] {PRIMARY KEY | UNIQUE | NOT NULL |
REFERENCES TablaExterna [(CampoClaveTE)]}
```

Donde NombreRestricción es un nombre opcional que se puede dar a la restricción. PRIMARY KEY especifica que el campo forma clave primaria. UNIQUE especifica que no puede haber dos o más registros en la tabla con el mismo valor de campo. NOT NULL especifica que el campo no puede contener un valor NULL (debe existir un valor para el campo). REFERENCES indica que el campo es una clave externa que debe coincidir con el campo clave de la tabla externa CampoClaveTE de algún registro de la tabla TablaExterna.

El siguiente ejemplo crea la tabla Empleados indicando que el campo Nombre no debe contener nulos y que el campo Apellidos forma clave primaria.

```
CREATE TABLE Empleados (Nombre TEXT (25)  NOT NULL, Apellidos TEXT
(50) PRIMARY KEY);
```

La segunda versión de la cláusula CONSTRAINT se aplica a varios campos. Su sintaxis es:

```
[CONSTRAINT NombreRestricción]
{PRIMARY KEY (CampoClave1[, CampoClave2 [, ...]]) |
 UNIQUE (CampoÚnico1[,CampoÚnico2[, ...]]) |
 FOREIGN KEY (CampoClaveExterna1[,CampoClaveExterna2 [, ...]])
   REFERENCES TablaExterna [(CampoClaveTE1[,CampoClaveTE2 [, ...]])]}
```

Por ejemplo, la siguiente consulta crea una tabla como en el anterior pero especificando los campos Nombre y Apellidos como clave primaria:

```
CREATE TABLE Empleados (Nombre TEXT (25), Apellidos TEXT (50), PRIMARY
KEY(Nombre, Apellidos));
```

En la siguiente consulta se crea la tabla Médicos especificando que sus campos Nombre y Apellidos deben guardar restricción de clave externa con los campos homónimos de la tabla Empleados.

```
CREATE TABLE Médicos (Nombre TEXT (25), Apellidos TEXT (50), FOREIGN
KEY (Nombre, Apellidos) REFERENCES Empleados (Nombre, Apellidos));
```

3.2.2.3. Creación de índices

Una vez creada una tabla se pueden crear índices sobre ella. Los índices mejoran el rendimiento del acceso en lectura a las tablas, aunque dismuyen el rendimiento de las escrituras. Para crearlos se usa la siguiente sintaxis:

```
CREATE [ UNIQUE ] INDEX NombreÍndice
ON NombreTabla (NombreCampo1 [ASC|DESC][, NombreCampo2 [ASC|DESC],
...])
 [WITH { PRIMARY | DISALLOW NULL | IGNORE NULL }]
```

Donde NombreÍndice es el nombre que se le asigna al índice, NombreTabla es la tabla donde se aplica el índice, NombreCampo*i* es el nombre del campo que forma parte del índice. ASC denota una ordenación creciente y DESC decreciente. PRIMARY indica que se trata de clave primaria (por lo tanto, no puede contener duplicados ni valores nulos). DISALLOW NULL impide que aparezcan valores nulos, mientras que IGNORE NULL indica que pueden aparecer, pero excluye del índice los registros que contengan estos valores nulos.

Se puede utilizar CREATE INDEX para crear índices sobre tablas adjuntas (por ejemplo, de una fuente de datos ODBC tal como SQL Server u Oracle). La creación del índice es local y no afecta a la fuente de datos externa.

En el siguiente ejemplo se crea un índice llamado MiÍndice en la tabla Empleados formado por los campos Prefijo y Telefono.

```
CREATE INDEX MiÍndice ON Empleados (Prefijo, Teléfono);
```

Con la siguiente consulta se crea un índice en la tabla Empleados utilizando el campo ID, obligando que que el campo ID no contenga valores nulos ni repetidos.

```
CREATE UNIQUE INDEX MiÍndice ON Empleados (ID) WITH DISALLOW NULL;
```

3.2.2.4. Modificación de tablas

La instrucción ALTER TABLE permite modificar el diseño de una tabla. Su sintaxis es:

```
ALTER TABLE NombreTabla
{ADD  {COLUMN NombreCampo Tipo [(Tamaño)] [RestricciónCampoÚnico] |
 RestricciónMulticampo} |
 DROP {COLUMN NombreCampo | CONSTRAINT NombreRestricción} }
```

Donde NombreTabla es la tabla a modificar. Con la cláusula ADD se puede agregar un campo o una restricción multicampo. Con la cláusula DROP se puede eliminar un campo, en cuyo caso hay que especificar COLUMN seguido del nombre del campo a eliminar, o una restricción, ya sea de campo único o multicampo, proporcionando su nombre (NombreRestricción) después de CONSTRAINT.

Con la siguiente consulta se agrega un campo Salario de tipo Moneda a la tabla Empleados.

```
ALTER TABLE Empleados ADD COLUMN Salario CURRENCY;
```

Con la siguiente consulta se elimina el campo Salario de la tabla Empleados.

```
ALTER TABLE Empleados DROP COLUMN Salario;
```

Con la siguiente consulta se agrega una restricción de integridad referencial a la tabla Pedidos. La clave externa es el campo ID_Empleado y la tabla externa es Empleados, cuya clave de referencia es ID_Empleado. En este ejemplo no sería necesario indicar el campo junto al nombre de la tabla en la cláusula REFERENCES, pues ID_Empleado es la clave principal de la tabla Empleados.

```
ALTER TABLE Pedidos ADD CONSTRAINT RelaciónPedidos FOREIGN KEY
    (ID_Empleado) REFERENCES Empleados (ID_Empleado);
```

Finalmente, la siguiente consulta elimina la restricción que se acaba de agregar a la tabla Pedidos.

```
ALTER TABLE Pedidos DROP CONSTRAINT RelaciónPedidos;
```

3.2.2.5. Eliminación de tablas e índices

La eliminación de tablas e índices se lleva a cabo mediante la sentencia DROP. Si se aplica a una tabla con datos, se borra tanto los datos como el esquema, por lo que hay que usarla con cuidado. Cuando se aplica a un índice, simplemente se elimina el índice del esquema de la base de datos. La sintaxis de la consulta de eliminación de tablas es:

```
DROP TABLE NombreTabla;
```

Donde NombreTabla es el nombre de la tabla a borrar.

La sintaxis de la consulta de eliminación de índices es:

```
DROP INDEX NombreÍndice ON NombreTabla;
```

Donde NombreÍndice es el nombre del índice a borrar que está definido sobre la tabla de nombre NombreTabla.

La siguiente consulta elimina la tabla Médicos:

```
DROP TABLE Médicos;
```

La siguiente consulta elimina el índice Número de colegiado de la tabla Médicos:

```
DROP INDEX [Número de colegiado] ON Médicos;
```