

1. BASES DE DATOS RELACIONALES

1.1 INTRODUCCIÓN

No cabe duda que la información es la base de nuestra sociedad, recibimos y manejamos volúmenes enormes de información y el ordenador es la herramienta que nos permite almacenar y tratar esa información.

Para poder guardar y recuperar esa información necesitamos de un sistema de almacenamiento que sea fiable, fácil de manejar, eficiente, y de aplicaciones capaces de llevar a cabo esa tarea y de obtener resultados a partir de la información almacenada.

Este sistema es el denominado Sistema Gestor de Base de Datos (SGBD o DBMS versión inglesa DataBase Management System) y consiste en un conjunto de datos relacionados entre sí (la base de datos) y un conjunto de programas desarrollados para gestionar esos datos y comprobar que el sistema se mantenga libre de errores (en lo posible). Algunos autores excluyen del DBMS la base de datos. Además tenemos aplicaciones de usuario que accedan a la base de datos para utilizar esa información.

Para gestionar el SGBD tenemos la figura del ADMINISTRADOR DE LA BASE DE DATOS, persona (o equipo) encargada de definir y controlar la base de datos.

La inmensa mayoría de bases de datos que existen en el mercado hoy en día son bases de datos relacionales, bases de datos que organizan la información en tablas relacionadas entre sí mediante campos de relación y que cumplen una serie de reglas que procuran evitar redundancia y asegurar la integridad de los datos almacenados en ellas así como la seguridad de los datos.

Por los años setenta, para poder acceder a esos datos, IBM ideó un lenguaje basado en el álgebra relacional que fuese lo más parecido al lenguaje hablado, el SEQUEL que luego derivó en SQL (Structured Query Language), Lenguaje Estructurado de Interrogación/Consulta. A lo largo de su ya larga vida ha ido evolucionando y es hoy en día un lenguaje completo que permite definir la base de datos, controlarla, y manejar los datos almacenados en ella.

1.2 OBJETIVOS DE UN SGBD

1.2.1 INDEPENDENCIA DE LOS DATOS.

La independencia de los datos consiste en hacer que los programas no dependan de la estructura de los datos que debe utilizar, que se pueda cambiar esa estructura sin tener que cambiar los programas relacionados con ella.

Se han definido dos tipos de independencia:

la **independencia física**: consiste en poder modificar parámetros de cómo está almacenada físicamente la información (por ejemplo cambiar el tipo de dato de un campo o cambiar la ubicación de la información de un disco a otro), sin que ello suponga una modificación de los programas existentes.

la **independencia lógica**: consiste en poder cambiar la definición conceptual del sistema de información (por ejemplo añadir un nuevo campo a la información de los clientes) sin que ello suponga una modificación de los programas existentes.

1.2.2 SEGURIDAD E INTEGRIDAD.

Otro objetivo a lograr es el de la **seguridad**, que los usuarios no puedan acceder a datos sin autorización. Si juntamos toda la información de la empresa en un sólo sitio, el SGBD debe tener mecanismos para que cualquier usuario pueda tener acceso únicamente a la información que necesita de cara a la privacidad de esa información, incluso si tiene acceso a una información que se pueda decidir si además de visualizarla puede modificarla. Por ejemplo en un sistema en el que los alumnos pueden consultar sus notas, ¡deberá de existir algún mecanismo para que el alumno pueda ver sus notas pero no cambiarlas!

La **integridad** se refiere a que la información almacenada en la base de datos esté libre de errores. Esto no siempre es posible ya que existen distintos tipos de errores que tienen diferentes soluciones:

* fallos de hardware, estos errores sólo se pueden subsanar mediante copias de seguridad que pueden ser automáticas o manuales.

* fallos del programador, puede que aparezcan datos erróneos en la base de datos como consecuencia de errores en el programa que genera estos datos. Para evitar al máximo este tipo de errores el sistema debe ser capaz de detectar automáticamente la mayor cantidad de errores para descargar los programas de comprobaciones rutinarias, el lenguaje de programación debe ser fácil de utilizar y si el sistema ofrece la posibilidad de utilizar juegos de ensayos bien definidos, será más fácil probar los programas.

* fallos del usuario final, el usuario que introduce datos en la base de datos también puede cometer errores, el sistema debe permitir controlar al máximo la información que se introduce para limitar el número de estos errores, para ello se incluyen cláusulas de validación de los datos, validaciones de diferentes tipos que veremos con más detalle más adelante.

* fallos derivados de la concurrencia, ya que toda la información está centralizada y los distintos usuarios acceden a ella de forma simultánea, pueden ocurrir problemas cuando dos usuarios quieren acceder al mismo dato a la vez. Por ello el SGBD debe tener establecidos mecanismos para evitar este tipo de problema, bloquear registros, abortar automáticamente transacciones etc...

1.2.3 REDUNDANCIA MÍNIMA.

La redundancia consiste en que existan datos idénticos repetidos en varios lugares. Por ejemplo si nos guardamos la dirección del cliente en la factura, en la cta. contable, y en los datos generales del cliente tendremos redundancia, el mismo dato repetido en varios sitios, pues esto nos produce varios problemas:

- * la información repetida ocupa espacio innecesario.
- * la variación de un domicilio supone el variar ese domicilio en todos los lugares donde esté almacenado
=> mayor tiempo de proceso
=> posibilidad de inconsistencia (el mismo cliente con dos domicilios ¿cuál es el bueno?)

Por todo ello hay que intentar eliminar al máximo esa redundancia.

1.2.4 FACILIDAD DE RECUPERACIÓN DE LA INFORMACIÓN.

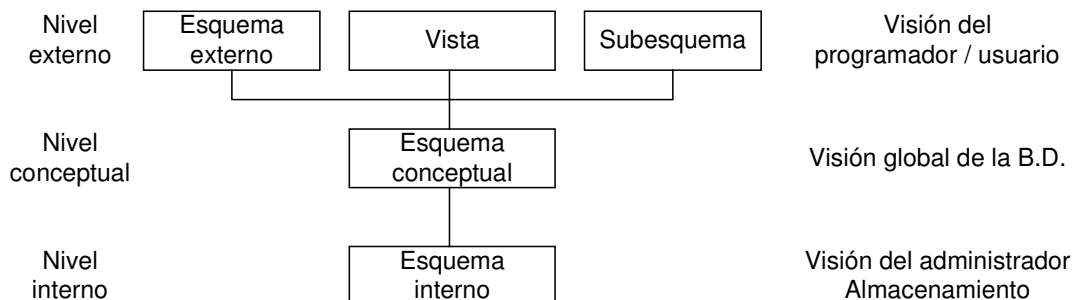
Otro objetivo muy importante de un SGBD es el proporcionar al usuario (y al programador) unas herramientas potentes de manejo de datos para que pueda de manera sencilla y rápida, obtener toda la información que desea.

1.3 ARQUITECTURA DE UN SGBD.

Para lograr los objetivos anteriores las bases de datos tienen una estructura en tres niveles, y el sistema gestor es el encargado de relacionar los niveles entre sí.

Los niveles no son más que diferentes formas de ver la información almacenada en la base de datos, permiten ver esa información desde varios puntos de vista.

Esta arquitectura es la ANSI PSARC y define tres niveles o tres “visiones distintas de la base de datos, el nivel interno, conceptual y externo.



1.3.1 EL NIVEL INTERNO.

El nivel interno es el más cercano al almacenamiento de los datos, este nivel es el de la visión del administrador de la BD, ya que es donde tenemos la definición física de los datos, su organización interna, la definición de las reglas de validación, controles de seguridad, etc...

1.3.2 EL NIVEL CONCEPTUAL.

En este nivel se tiene una visión global de todos los datos que intervienen en la base de datos, pero a nivel de concepto olvidándose de la estructura interna. Podemos decir que en este nivel es donde tenemos la definición de qué datos se guardan en la base de datos y las relaciones que unen los datos entre sí.

1.3.3 EL NIVEL EXTERNO.

Este nivel representa la percepción de los datos por cada uno de los usuarios o programadores, digamos que el usuario no ve todos los datos que aparecen en el nivel conceptual, sino sólo los que le hacen falta. Cada esquema externo es un subconjunto del esquema conceptual, proporciona una visión parcial del mismo. De esta forma se puede proteger la información reservada a unos pocos usuarios. Para el usuario no habrá más datos en la bd. que los definidos en su esquema externo.

El SGBD es el encargado de realizar el enlace entre los tres niveles de manera que cuando un usuario pida un datos definido en su esquema externo, el SGBD lo busca en el fichero adecuado y de acuerdo a la definición interna, pero esa transformación será totalmente transparente para el usuario, el usuario sólo pide el nombre de un cliente y el SGBD se encarga de buscar ese datos en la tabla correspondiente almacenada en tal dispositivo.

Esta estructura permite lograr los objetivos nombrados anteriormente, tenemos independencia de los datos, integridad gracias a las reglas que se pueden definir a nivel interno; seguridad que se consigue por medio de los esquemas externos, ya que el usuario sólo tiene acceso a su esquema externo que le proporciona los datos que el administrador ha considerado incluir en el esquema, para el usuario no hay más datos que éstos.

Además los SGBD tienen mecanismos para limitar el acceso a cada usuario según el tipo de operación que quiera realizar. Esto lo consigue mediante la definición de autorizaciones que pueden ser de distinto tipo: autorización de lectura, de inserción, de actualización, autorizaciones especiales para poder variar el esquema conceptual etc...

Gracias al nivel conceptual tenemos una definición "resumida" de la información almacenada por lo que es más fácil detectar redundancia en los datos.

1.4 EL ADMINISTRADOR DE LA BASE DE DATOS.

El administrador es el encargado de gestionar y controlar todo el sistema con la ayuda que le proporciona el SGBD. Entre sus responsabilidades se incluye:

- * Diseñar la base de datos:
 - definir el esquema conceptual
 - definir el esquema interno, supone definir soportes, organizaciones, métodos de acceso, factores de bloqueo, índices, tamaño y tipo de los campos etc... y definir los procedimientos de validación a efectuar por el sistema sobre los datos.
 - definir los esquemas externos, tantos como visiones distintas tengan que tener todos los usuarios de la empresa.
- * Definir los usuarios.
- * Conceder autorizaciones a los distintos usuarios.
- * Mantener los esquemas actualizados.
- * Definir estrategias de recuperación adecuada de la información en caso de pérdida o daños sufridos por algún fallo.
- * Realizar estadísticas para evaluar el rendimiento del sistema

1.5 EL DICCIONARIO DE DATOS.

Dentro del SGBD, hay una parte que son datos sobre los datos, es una base de datos en la que se almacena toda la información necesaria para que el sistema funcione. Esta base de datos es el diccionario de datos y contiene:

- * las tablas que almacenan las definiciones de los esquemas,
- * las tablas que contienen los códigos de autorizaciones,
- * la definición de los alias,
- * la relación entre los distintos usuarios y los esquemas externos asociados a los mismos.

1.6 Importancia de definir bien la base de datos.

Un sistema gestor de base de datos nos permite almacenar la información de forma muy eficiente y permite realizar controles importantes sobre los datos. Pero todas estas cualidades no tendrán efecto si no hemos realizado un buen diseño de la base de datos.

Es fundamental realizar un buen diseño de la base de datos.

No podemos aquí enseñaros a diseñar bases de datos, esto está desarrollado en otro módulo del ciclo, pero sí recordaremos los puntos a tener en cuenta para que nuestra base de datos sea la mejor:

- obtener un esquema lógico de la base de datos que sea normalizado, es completamente necesario distribuir de forma correcta los diferentes datos en las diferentes tablas, no incluir redundancia, definir la claves primarias adecuadas y las claves ajenas (o relaciones entre las diferentes tablas).
- Definir reglas de validación de los campos
- Definir reglas de comportamientos.

Cuanto más reglas definamos, más cosas comprobará de forma automática el SGBD reduciendo así la probabilidad de errores en los datos.

1.7 Base de datos local versus Cliente/Servidor

Existen diferentes sistemas gestores de bases de datos que pueden funcionar en modo local o en modo Cliente/Servidor.

En modo local tenemos la base de datos y el usuario ubicados en el mismo ordenador. Un ejemplo de base de datos que funciona en modo local es el Microsoft Access, MS Access es una base de datos fácil de manejar por usuarios poco expertos que funciona bien en modo local y mientras no tenga que albergar grandes cantidades de información.

En modo Cliente/Servidor, la base de datos se encuentra en un ordenador (el Servidor) y los usuarios acceden simultáneamente a esa base de datos a través de la red (sea una red local o Internet) desde sus ordenadores a través de un programa Cliente.

Aunque Access permite el acceso compartido y a través de una red, no está diseñado con ese objetivo por lo que si desea utilizar un sistema Cliente/Servidor se recomienda utilizar otro sistema como por ejemplo SQL Server, Oracle, DBII,...

1.8 Conceptos básicos sobre bases de datos relacionales.

Para trabajar con bases de datos relacionales es muy importante tener claros una serie de conceptos que veremos en este punto.

Toda la información almacenada en una base de datos relacional está organizada en tablas.

La tabla es el primer objeto de una base de datos y se organiza en **filas** y **columnas**, una fila equivale a un **registro** y las columnas definen los **campos** del registro.

Cada columna se define sobre un **tipo de datos**, existen tipos de datos estándares como en cualquier lenguaje de programación, y tipos propios del SGBD.

Se pueden definir reglas de validación de los campos. Una regla de validación no es más que una condición que deberán cumplir todas las filas de la tabla y que el sistema se encarga automáticamente de verificar e impedir que exista en la tabla un registro que no cumpla esa condición.

En una tabla debe de existir una **clave principal/primaria** formada por una columna o combinación de varias columnas que permite identificar a cada uno de los registros de la tabla. Para poder cumplir con este objetivo la clave primaria **no puede contener valores nulos ni valores duplicados**. De eso se encargará el sistema, si intentamos crear un registro con el valor nulo en su campo clave o si el valor de clave ya existe en otro registro de la tabla, el sistema no nos dejará. En esto consiste la integridad de claves.

También nos podemos encontrar con otros campos identificadores, se conocen como **claves secundarias**, estos campos también permiten identificar los registros de la tabla por lo que tienen las mismas restricciones que las claves principales (**no nulo y no duplicado**).

Para terminar con las claves, tenemos otro tipo de claves, las **claves ajenas (foráneas/externas)**. Una clave ajena es un campo (o combinación de campos) que señala a un registro de otra tabla, contiene un valor que identifica un registro de la otra tabla. Estos campos sirven para relacionar las tablas entre sí.

Por ejemplo tenemos una tabla de facturas con los siguientes campos:

Código de factura, Fecha, importe total, cliente.

El campo cliente indica de qué cliente es la factura, contiene pues un código de cliente. Es una clave ajena, señala a un registro de la tabla de clientes. Es evidente que para que no haya errores en la tabla de facturas, los códigos de cliente introducidos deben ser de clientes registrados en la tabla de Clientes. En esto consiste la **integridad referencial**.

Para que no existan errores de integridad en la base de datos, el sistema comprueba automáticamente que los valores introducidos en las claves ajenas existan en el campo de referencia en la otra tabla, si no existe, no nos dejará insertar el registro.

Si intentamos grabar una factura con un código de cliente que no existe en la tabla de clientes, el sistema no nos dejará grabar la factura.

A diferencia de las claves primarias y secundarias, una clave ajena en principio sí puede contener el valor nulo (es decir ningún valor) como cualquier otro campo.

El valor nulo (NULL) es importante porque representa la ausencia de valor en el campo y no es lo mismo que el valor cero "0". De hecho es un valor tan especial que no funciona como los demás valores, por ejemplo no podemos comparar (con el operador de comparación =) un campo con el valor nulo, tenemos que utilizar una fórmula especial (IS NULL).

Estos son los conceptos mínimos, luego cada SGBD utiliza sus propios objetos que en algunos casos se repiten en varios sistemas como las VIEWS, TRIGGERS, INDEX etc... pero esto se verá cuando se estudie un SGBD concreto como haremos con SQL Server.

2. DML – Consultas Simples

2.1 Introducción.

La sentencia SELECT es, con diferencia, la más compleja y potente de las sentencias SQL, con ella podemos recuperar datos de una o más tablas, seleccionar ciertos registros e incluso obtener resúmenes de los datos almacenados en la base de datos, por algo está considerada como la “reina” del SQL.

La sintaxis completa es la siguiente:

```
SELECT sentencia ::= [WITH <expresion_tabla_comun> [,...n]]
    <expresion_consulta>
    [ORDER BY {expresion_columna|posicion_columna [ASC|DESC] }
    [,...n ]]
    [COMPUTE
    [{AVG|COUNT|MAX|MIN|SUM} (expresion)][ ,...n ] [BY expresion[ ,...n ]]
    ]
    [<FOR clausula_for>]
    [OPTION (<query_hint>[ ,...n ])]

<expresion_consulta> ::=
    {<especificacion_consulta> | ( < expresion_consulta > ) }
    [ {UNION [ALL]|EXCEPT|INTERSECT}
    <especificacion_consulta> | (<expresion_consulta>) [...n ]
    ]

<especificacion_consulta> ::=
    SELECT [ALL|DISTINCT]
    [TOP expresion [PERCENT] [WITH TIES] ]
    <lista_seleccion>
    [INTO nueva_tabla]
    [FROM { <origen> } [ ,...n ] ]
    [WHERE <condicion_busqueda> ]
    [GROUP BY [ ALL ] expresion_agrupacion [ ,...n ]
    [WITH { CUBE | ROLLUP } ]
    ]
    [HAVING < condicion_busqueda > ]
```

Debido a la complejidad de la sentencia (en la sintaxis anterior no hemos detallado algunos elementos), la iremos viendo poco a poco, empezaremos por ver consultas básicas para luego ir añadiendo más cláusulas.

Empezaremos por ver las consultas más simples, basadas en una sola tabla y nos limitaremos a la siguiente sintaxis:

```
SELECT [ALL|DISTINCT]
    [TOP expresion [PERCENT] [WITH TIES]]
    <lista_seleccion>
    FROM <origen>
    [WHERE <condicion_busqueda> ]
    [ORDER BY {expresion_columna|posicion_columna [ASC|DESC]} [ ,...n ]]
```

2.2 Origen de datos FROM

De la sintaxis anterior, el elemento <origen> indica de dónde se va a extraer la información y se indica en la cláusula FROM, es la única cláusula obligatoria.

Para empezar veremos un origen de datos basado en un sola tabla.

La sintaxis será la siguiente:

```
<origen>::=
nb_tabla | nb_vista [[ AS ] alias_tabla ]
```

nb_tabla representa un nombre de tabla

nb_vista un nombre de vista

Opcionalmente podemos definir un **nombre de alias**.

Un nombre de alias (*alias_tabla*) es un nombre alternativo que se le da a la tabla dentro de la consulta.

Si se define un nombre de alias, dentro de la consulta, será el nombre a utilizar para referirnos a la tabla, el nombre original de la tabla ya no tendrá validez.

Se utilizan los nombres de alias para simplificar los nombres de tablas a veces largos y también cuando queremos combinar una tabla consigo misma; ya volveremos sobre los alias de tabla cuando veamos consultas multitabla.

La palabra AS no añade ninguna operatividad, está más por estética.

Podemos escribir:

```
SELECT ...
FROM tabla1          Sacamos los datos de la tabla tabla1
```

```
SELECT ...
FROM tabla1 t1      Sacamos los datos de la tabla tabla1 y le asignamos un
alias de tabla: t1
```

```
SELECT ...
FROM tabla1 AS t1  Es equivalente a la sentencia anterior.
```

2.3 La lista de selección

En la lista de selección <lista_seleccion> indicamos las columnas que se tienen que visualizar en el resultado de la consulta.

```
<lista_seleccion> ::=
{
  [{nombre_tabla|nombre_vista|alias_tabla}.]
  { *
  | {nb_columna |<expresion>|$IDENTITY|$ROWGUID} [[AS] alias_columna]
  }
  | alias_columna = <expresion>
} [ ,...n ]
```

Separamos la definición de cada columna por una coma y las columnas del resultado aparecerán en el mismo orden que en la lista de selección.

El tipo de datos, tamaño, precisión y escala de la columna del resultado de la consulta son los mismos que los de la expresión que define la columna.

Podemos definir las columnas del resultado de varias formas, mediante:

- una expresión simple: una referencia a una función, una variable local, una constante o una columna del origen de datos,

- una subconsulta escalar, que es otra instrucción SELECT que se evalúa como un valor individual para cada fila del origen de datos (esto no lo veremos de momento),

- una expresión compleja generada al usar operadores en una o más expresiones simples.

- la palabra clave *.

- la asignación de variables con el formato @variable_local = expresión.

- la palabra clave \$IDENTITY.

- la palabra clave \$ROWGUID.

2.3.1 Columnas del origen de datos

Cuando queremos indicar en la lista de selección una columna del origen de datos, la especificamos mediante su nombre simple o nombre cualificado.

El nombre cualificado consiste en el nombre de la columna precedido del nombre de la tabla donde se encuentra la columna. Si en el origen de datos hemos utilizado una vista o un nombre de alias, para la tabla deberemos utilizar ese nombre. Es obligatorio utilizar el nombre cualificado cuando el nombre de la columna aparece en más de una tabla del origen de datos.

Ejemplo de consulta simple:

Listar nombres, oficinas, y fechas de contrato de todos los empleados:

```
SELECT nombre, oficina, contrato
FROM empleados;
```

El resultado sería:

<u>nombre</u>	<u>oficina</u>	<u>contrato</u>
Antonio Viguer	12	1986-10-20
Alvaro Jaumes	21	1986-12-10
Juan Rovira	12	1987-03-01
José González	12	1987-05-19
Vicente Pantalla	13	1988-02-12
Luis Antonio	11	1988-06-14
Jorge Gutiérrez	22	1988-11-14
Ana Bustamante	21	1989-10-12
María Sunta	11	1999-10-12
Juan Víctor	NULL	1990-01-13

Listar una tarifa de productos:

```
SELECT idfab, idproducto, descripcion, productos.precio
FROM productos;
```

Hemos cualificado la columna precio aunque no es necesario en este caso.

El resultado sería:

<u>Idfab</u>	<u>idproducto</u>	<u>descripcion</u>	<u>precio</u>
aci	41001	arandela	0,58
aci	41002	bisagra	0,80
aci	41003	art t3	1,12
aci	41004	art t4	1,23
aci	4100x	junta	0,26
aci	4100y	extractor	28,88
aci	4100z	mont	26,25
bic	41003	manivela	6,52
bic	41089	rodamiento	2,25

2.3.2 Alias de columna

Por defecto, en el encabezado de cada columna del resultado, aparece el nombre de la columna origen, pero esto se puede cambiar definiendo un alias de columna, el alias de columna es un nombre alternativo que se le da a esa columna.

El alias de columna se indica mediante la cláusula AS. Se escribe el nuevo texto tal cual sin comillas.

Ejemplo:

```
SELECT numclie, nombre AS nombrecliente
FROM clientes;
```

El resultado será :

<u>Numclie</u>	<u>nombrecliente</u>
2101	Luis García Antón
2102	Alvaro Rodríguez
2103	Jaime Llorens

en vez de:

<u>numclie</u>	<u>nombre</u>
2101	Luis García Antón
2102	Alvaro Rodríguez
2103	Jaime Llorens

La palabra AS es opcional.

```
SELECT numclie,nombre nombrecliente
FROM clientes;
```

Sería equivalente a la consulta anterior

Si queremos incluir espacios en blanco en el nombre lo debemos encerrar entre corchetes.

```
SELECT numclie,nombre AS [nombre cliente]
FROM clientes;
```

Nota importante: Este nombre de alias se podrá utilizar en la lista de selección y en la cláusula ORDER BY pero no en la cláusula WHERE.

2.3.3 Columnas calculadas

Además de las columnas que provienen directamente de la tabla, una consulta SQL puede incluir columnas calculadas cuyos valores se evalúan a partir de una expresión.

La expresión puede contener cualquier operador válido (+, -, *, /, &...), cualquier función válida, nombres de columnas del origen de datos, nombres de parámetros o constantes y para combinar varias operaciones se pueden utilizar los paréntesis.

Ejemplos:

Listar la ciudad, región y el superávit de cada oficina. Consideraremos el superávit como el volumen de ventas que se encuentran por encima o por debajo del objetivo de la oficina.

```
SELECT ciudad, region, (ventas-objetivo) AS superavit
FROM oficinas;
```

El resultado será:

<u>ciudad</u>	<u>region</u>	<u>superavit</u>
Valencia	este	11800,00
Alicante	este	-6500,00
Castellon	este	1800,00
Badajoz	oeste	11100,00
A Coruña	oeste	-11400,00
Madrid	centro	NULL
Madrid	centro	-10000,00
Pamplona	norte	NULL
Valencia	este	-90000,00

De cada producto queremos saber el id-fabricante, id-producto, su descripción y el valor de sus existencias.

```
SELECT idfab, idproducto, descripcion, (existencias*precio) AS valoracion
FROM productos;
```

El resultado sería:

idfab	idproducto	descripción	valoración
aci	41001	arandela	160,66
aci	41002	bisagra	133,60
aci	41003	art t3	231,84
aci	41004	art t4	170,97
aci	4100x	junta	9,62
aci	4100y	extractor	722,00
aci	4100z	mont	735,00
bic	41003	manivela	19,56
bic	41089	rodamiento	175,50

Listar el nombre, mes y año del contrato de cada vendedor.

```
SELECT nombre, MONTH(contrato) AS [Mes de contrato], YEAR(contrato) AS [Año
de contrato]
FROM empleados;
```

El resultado será:

Nombre	Mes de contrato	Año de contrato
Antonio Viguer	10	1986
Alvaro Jaumes	12	1986
Juan Rovira	3	1987

Listar las ventas en cada oficina con el formato: 22 tiene ventas de 186,042.00 €

```
SELECT oficina, 'tiene ventas de ' AS [ ], ventas
FROM oficinas;
```

El resultado sería:

oficina		ventas
11	tiene ventas de	69300,00
12	tiene ventas de	73500,00
13	tiene ventas de	36800,00
21	tiene ventas de	83600,00
22	tiene ventas de	18600,00
23	tiene ventas de	NULL
24	tiene ventas de	15000,00
26	tiene ventas de	NULL
28	tiene ventas de	0,00

También podemos utilizar la sintaxis:

```
alias_columna = <expresion>
```

Ejemplo:

```
SELECT oficina, superavit = ventas-objetivo
```

Esto tiene el mismo efecto que

```
SELECT oficina, ventas-objetivo AS superavit
```

2.3.4 Utilización del *

Si queremos visualizar todas las columnas del origen de datos, en lugar de indicar todas las columnas una a una se puede utilizar el carácter de sustitución *.

Mostrar todos los datos de la tabla oficinas.

```
SELECT *
FROM oficinas
```

Obtener todos los datos y el superávit de cada oficina.

```
SELECT *, (ventas-objetivo) AS superavit
FROM oficinas
```

También podemos cualificar el * con un nombre de tabla, de vista o un alias de tabla:

```
SELECT oficinas.*
FROM oficinas
```

*oficinas.** se interpreta como: *todas las columnas de la tabla oficinas.*

Esta forma se utiliza normalmente cuando el origen está basado en varias tablas y queremos indicar todas las columnas no del origen completo sino de una tabla concreta.

2.3.5 La palabra clave \$IDENTITY

La palabra clave \$IDENTITY se interpreta como la columna de la tabla que tiene la propiedad IDENTITY. Por ejemplo, si en la columna oficina de la tabla oficinas se ha definido la propiedad IDENTITY,

```
SELECT $IDENTITY, ciudad
FROM oficina;
```

Es equivalente a:

```
SELECT oficina, ciudad
FROM oficinas;
```

2.3.6 La palabra clave \$ROWGUID.

La palabra clave \$ROWGUID se interpreta como la columna de la tabla que tiene la propiedad ROWGUIDCOL y se puede utilizar de la misma forma que \$IDENTITY.

2.4 Ordenación de las filas del resultado ORDER BY

Si queremos ordenar las filas del resultado de la consulta, lo podemos hacer mediante la cláusula ORDER BY

```
ORDER BY {expression_columna|posicion_columna [ASC|DESC]} [ , ...n ]
```

Podemos indicar una columna o varias separadas por una coma, la columna de ordenación se especifica mediante el nombre de columna en el origen de datos o su posición dentro de la lista de selección. Si utilizamos el nombre de columna, no hace falta que la columna aparezca en la lista de selección. Si utilizamos la posición es la posición de la columna dentro de la lista de selección empezando en 1.

Por defecto la filas se ordenarán en modo ascendente (ASC), de menor a mayor; si queremos invertir ese orden añadimos detrás de la columna la palabra DESC.

Si la columna de ordenación es alfanumérica, las filas se ordenarán por orden alfabético.

Ejemplos:

Mostrar las ventas de cada oficina, ordenadas por orden alfabético de región y dentro de cada región por ciudad.

```
SELECT oficina, region, ciudad, ventas
FROM oficinas
ORDER BY region, ciudad;
```

Da como resultado:

Oficina	region	ciudad	ventas
24	centro	Aranjuez	15000,00
23	centro	Madrid	NULL
12	este	Alicante	73500,00
13	este	Castellón	36800,00
11	este	Valencia	69300,00
28	este	Valencia	0,00
26	norte	Pamplona	NULL
22	oeste	A Coruña	18600,00
21	oeste	Badajoz	83600,00

Listar las oficinas de manera que las oficinas de mayores ventas aparezcan en primer lugar.

```
SELECT ciudad, region, ventas
FROM oficinas
ORDER BY ventas DESC;
```

Ciudad	region	ventas
Badajoz	oeste	83600,00
Alicante	este	73500,00
Valencia	este	69300,00
Castellón	este	36800,00
A Coruña	oeste	18600,00
Aranjuez	centro	15000,00
Valencia	este	0,00
Pamplona	norte	NULL
Madrid	centro	NULL

Listar las oficinas clasificadas en orden descendente de rendimiento de ventas, de modo que las de mayor rendimiento aparezcan las primeras.

```
SELECT ciudad, region, ventas-objetivo
FROM oficinas
ORDER BY 3 DESC;
```

Lo mismo que el anterior pero agrupadas por región.

```
SELECT region, ciudad, (ventas-objetivo) AS superavit
FROM oficinas
ORDER BY region, superavit DESC;
```

Resultado:

Region	ciudad	superavit
centro	Aranjuez	-10000,00
centro	Madrid	NULL
este	Valencia	11800,00
este	Castellón	1800,00
este	Alicante	-6500,00
este	Valencia	-90000,00
norte	Pamplona	NULL
oeste	Badajoz	11100,00
oeste	A Coruña	-11400,00

En este caso hemos utilizado el alias de columna para hacer referencia a la columna calculada y también se puede observar que las filas aparecen ordenadas por región ascendente (no hemos incluido nada después del nombre de la columna) y dentro de cada región por superávit y descendente.

2.5 Eliminar filas duplicadas DISTINCT/ALL

SQL no elimina las filas duplicadas en el resultado de la consulta, si nosotros no queremos que se repitan las filas, tenemos la cláusula DISTINCT.

Al incluir la cláusula DISTINCT en la SELECT, se eliminará del resultado las repeticiones de filas de resultado. Si por el contrario queremos que aparezcan todas las filas seleccionadas podemos incluir la cláusula ALL o nada, ya que ALL es el valor por defecto.

Listar los nº de empleado de los directores de las oficinas.

```
SELECT dir
FROM oficinas;
```

```
dir
106
104
105
108
108
108
108
108
NULL
NULL
```

Si un mismo empleado dirige varias oficinas (por ejemplo el 108), su código aparece repetido en el resultado. Para evitarlo modificamos la consulta:

```
SELECT DISTINCT dir
FROM oficinas;
```

```
dir
NULL
104
105
106
108
```

Han desaparecido los valores duplicados. Los que se eliminan son valores duplicados de filas del resultado:

<u>Dir</u>	<u>region</u>
NULL	este
NULL	norte
104	este
105	este
106	este
108	centro
108	oeste

NOTA: La cláusula DISTINCT hace que la consulta tarde algo más en ejecutarse debido al proceso adicional de buscar y eliminar las repeticiones, por lo que se aconseja utilizarla únicamente cuando sea imprescindible.

2.6 La cláusula TOP.

```
[TOP <expresión> [PERCENT] [WITH TIES]]
```

La cláusula TOP indica que en el resultado no deben aparecer todas las filas resultantes sino un cierto número de registros, las n primeras. Si la consulta incluye la cláusula ORDER BY, se realiza la ordenación antes de extraer los n primeros registros.

La expresión representa ese número n y debe devolver un número entero sin signo.

Por ejemplo tenemos la siguiente tabla:

	cod	ventas
1	1	100,00
2	2	200,00
3	3	200,00
4	4	10000,00
5	5	10000,00
6	6	10,00
7	7	250,00

```
select * from productos:
```

Si ordenamos por ventas:

```
select * from productos
order by ventas;
```

Obtenemos el siguiente resultado:

	cod	ventas
1	6	10,00
2	1	100,00
3	2	200,00
4	3	200,00
5	7	250,00
6	4	10000,00
7	5	10000,00

Si añadimos la cláusula TOP:

```
select TOP 3 * from productos
order by ventas
```

	cod	ventas
1	6	10,00
2	1	100,00
3	3	200,00

Obtenemos los 3 primeros registros:

Si existen más registros con las mismas ventas que el último valor de la lista, éstos no saldrán en el resultado de la consulta.

En el ejemplo el registro con cod = 2 no sale en el resultado y tiene las mismas ventas que cod = 3.

Si queremos que salgan añadimos la cláusula WITH TIES. La cláusula WITH TIES sólo se puede emplear si la SELECT incluye un ORDER BY, de lo contrario dará error.

Si añadimos la cláusula WITH TIES:

```
SELECT TOP 3 WITH TIES *  
FROM productos  
ORDER BY ventas
```

Obtenemos:

	cod	ver
1	6	10,
2	1	100
3	2	200
4	3	200

Se incluyen en el resultado todos los registros que tienen ventas iguales al último registro.

Otro ejemplo:

```
SELECT TOP 10 oficina, ciudad, ventas  
FROM oficinas  
ORDER BY ventas;
```

Devuelve las 10 peores oficinas en cuanto a ventas: ordenamos las oficinas por ventas de menor a mayor y sacamos las 10 primeras.

Si incluimos la palabra PERCENT, entonces n no indica el número de registros a recuperar sino el porcentaje de registros a recuperar del total de filas recuperadas después de ejecutar la cláusula WHERE.

```
SELECT TOP 50 PERCENT *  
FROM productos  
ORDER BY ventas
```

Devuelve:

	cod	ventas
1	6	10,00
2	1	100,00
3	2	200,00
4	3	200,00

Si el porcentaje no da exacto, siempre redondea al alza.

2.7 Selección de filas WHERE

La cláusula WHERE se emplea para especificar las filas que se desean recuperar del origen de datos.

```
WHERE <condicion_búsqueda>

<condicion_búsqueda> ::=
  { [NOT]<predicado>
    |(<condicion_búsqueda>)
  }
  [{AND|OR} [NOT] {<predicado>|(<condicion_búsqueda>)}]
[ , ...n ]
```

En el resultado de la consulta sólo aparecerán las filas que cumplan que la condición de búsqueda sea TRUE, los valores NULL no se incluyen, por lo tanto, en las filas del resultado. La condición de búsqueda puede ser una condición simple o una condición compuesta por varias condiciones (predicados) unidas por operadores AND y OR, no hay límite en cuanto al número de predicados que se pueden incluir. En las condiciones compuestas se pueden utilizar paréntesis para delimitar predicados y se aconseja su uso cuando se incluyen operadores AND y OR en la misma condición de búsqueda.

2.7.1 Predicados

En SQL tenemos 7 tipos de predicados, condiciones básicas de búsqueda:

- . la comparación estándar
- . coincidencia con patrón (LIKE)
- . pertenencia a un intervalo (BETWEEN)
- . si contiene (CONTAINS)
- . pertenencia a un conjunto (IN)
- . el test de valor nulo (IS NULL).
- . FREETEXT

2.7.1.1 Comparación estándar.

Compara el valor de una expresión con el valor de otra. Para la comparación se pueden emplear = , <> , !=, < , <= , !< , > , >= , !>

Sintaxis:

```
<expresion> {=|<>|!=|>|>=|!>|<|<=|!<} <expresion>
```

<expresion> Es un nombre de columna, una constante, una función, una variable, una subconsulta escalar o cualquier combinación de nombres de columna, constantes y funciones conectados mediante uno o varios operadores o una subconsulta. La expresión también puede contener la función CASE.

Hallar los vendedores cuyas ventas superan sus cuotas

```
SELECT numemp, nombre, ventas, cuota
FROM empleados
WHERE ventas > cuota
```

<u>Numemp</u>	<u>nombre</u>	<u>ventas</u>	<u>cuota</u>
101	Antonio Viguer	30500,00	30000,00
102	Alvaro Jaumes	47400,00	35000,00
103	Juan Rovira	28600,00	27500,00
105	Vicente Pantalla	36800,00	35000,00
106	Luis Antonio	29900,00	27500,00
108	Ana Bustamante	36100,00	35000,00
109	María Sunta	39200,00	3000,00

Las columnas que aparecen en el WHERE no tienen por qué aparecer en la lista de selección, esta instrucción es igual de válida:

```
SELECT numemp, nombre
FROM empleados
WHERE ventas > cuota;
```

Hallar vendedores contratados antes de 1988.

```
SELECT numemp, nombre, contrato
FROM empleados
WHERE contrato < '01/01/1988';
```

<u>Numemp</u>	<u>nombre</u>	<u>contrato</u>
101	Antonio Viguer	1986-10-20
102	Alvaro Jaumes	1986-12-10
103	Juan Rovira	1987-03-01
104	José González	1987-05-19

También podemos utilizar funciones, ésta es equivalente a la anterior:

```
SELECT numemp, nombre
FROM empleados
WHERE YEAR(contrato) < 1988;
```

La función YEAR() devuelve el año de una fecha.

Hallar oficinas cuyas ventas estén por debajo del 80% de su objetivo:

```
SELECT oficina
FROM oficinas
WHERE ventas < (.8 * objetivo);
```

Hallar las oficinas dirigidas por el empleado 108:

```
SELECT oficina
FROM oficinas
WHERE dir = 108;
```

2.7.1.2 Pertenencia a un intervalo BETWEEN

```
<expresion> [NOT] BETWEEN <expresion2> AND <expresion3>
```

Examina si el valor de la expresión de test está en el rango delimitado por los valores resultantes de expresion1 y expresion2, estos valores no tienen porqué estar ordenados en ANSI/ISO; expresion1 debe ser menor o igual a expresion2.

Hallar vendedores cuyas ventas estén entre 100,000 y 500,000

```
SELECT numemp, nombre, ventas
FROM empleados
WHERE ventas BETWEEN 20000 AND 100000;
```

Numemp	nombre	ventas
101	Antonio Viguer	30500,00
102	Alvaro Jaumes	47400,00
103	Juan Rovira	28600,00
105	Vicente Pantalla	36800,00
106	Luis Antonio	29900,00
108	Ana Bustamante	36100,00
109	María Sunta	39200,00

2.7.1.3 Test de pertenencia a conjunto IN

```
<expresion> IN ( <exp_valor> [ , ...n ] )
```

Examina si el valor de la expresión es uno de los valores incluidos en la lista de valores indicados entre paréntesis. Para expresar los valores mediante cualquier expresión, la única condición es que todas las exp_valor devuelvan el mismo tipo de datos.

Obtener los empleados que trabajan en las oficinas 11, 20 y 22:

```
SELECT oficina, numemp, nombre
FROM empleados
WHERE oficina IN (11,20,22);
```

Oficina	numemp	nombre
11	106	Luis Antonio
22	107	Jorge Gutiérrez
11	109	María Sunta

2.7.1.4 Test de valor nulo IS NULL

```
<expression> IS [NOT] NULL
```

Una condición de búsqueda puede ser TRUE, FALSE o NULL, este último caso se produce cuando algún campo que interviene en la condición tiene valor NULL.

A veces es útil comprobar explícitamente los valores NULL en una condición de búsqueda ya que estas filas puede que queramos darles un tratamiento especial, para ello tenemos la cláusula IS NULL.

Este test produce un valor TRUE o FALSE, por lo que se podrá combinar con otras condiciones. El valor NULL no es en sí un valor por eso no lo podemos utilizar en una igualdad.

```
SELECT numemp, nombre
FROM empleados
WHERE oficina = NULL;
```

Esta instrucción no da error pero no obtiene lo que en principio parece que quiere obtener. No obtenemos los empleados cuya oficina sea un valor nulo, no obtenemos nada, en cambio los obtendremos utilizando el test de valor nulo:

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE oficina IS NULL;
```

Resultado:

Numemp	nombre	oficina
110	Juan Victor	NULL

Juan Victor es el único empleado que no tiene oficina asignada.

Listar los vendedores asignados a alguna oficina.

```
SELECT numemp, nombre, oficina
FROM empleados
WHERE oficina IS NOT NULL;
```

Numemp	nombre	oficina
101	Antonio Viguer	12
102	Alvaro Jaumes	21
103	Juan Rovira	12
104	José González	12
105	Vicente Pantalla	13
106	Luis Antonio	11
107	Jorge Gutiérrez	22
108	Ana Bustamante	21
109	María Sunta	11

2.7.1.5 Test de correspondencia con patrón LIKE

Se utiliza cuando queremos comparar el valor de una columna con un patrón en el que se utilice caracteres comodines.

```
<expression> [NOT] LIKE <patron>
[ESCAPE 'car_escape']
```

Con expresión indicamos el valor a comparar (normalmente será el nombre de una columna) y patrón es la cadena que se busca.

El patrón es de tipo texto y tiene que escribirse entre comillas simples.

Dentro del patrón podemos utilizar los siguientes comodines:

% representa cualquier cadena de cero o más caracteres.

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE 'An%';
```

Numemp	nombre
101	Antonio Viguer
108	Ana Bustamante

Obtiene todos los nombres que empiecen por An

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '%z';
```

Numemp	nombre
104	José González
107	Jorge Gutiérrez

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '%on%';
```

<u>Numemp</u>	<u>nombre</u>
101	Antonio Viguer
104	José González
106	Luis Antonio

Obtiene los nombres que contienen on.

 representa cualquier carácter (sólo uno).

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '__a%';
```

<u>Numemp</u>	<u>nombre</u>
103	Juan Rovira
108	Ana Bustamante
110	Juan Victor

Obtiene los nombres cuya tercera letra sea una a (en el patrón tenemos dos caracteres subrayado).

[] sirve para indicar un carácter cualquiera perteneciente al conjunto indicando. El conjunto se indica enumerando los caracteres o indicando un intervalo.

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '[a-d]%' ;
```

Obtiene los nombres que empiezan por cualquier letra de la a á la d

Es equivalente a escribir:

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '[abcd]%' ;
```

[^] significa cualquier carácter individual que no se encuentre en el conjunto.

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '[^abcd]%' ;
```

Y

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '[^a-d]%' ;
```

Obtienes los nombres que no empiecen por a, b, c ni d.

Es importante tener en cuenta que dentro del patrón el espacio en blanco es considerado como un carácter más, si colocamos dos espacios en el patrón, se buscarán dos espacios en el campo.

Si queremos incluir en el patrón uno de los caracteres comodines y que no sea interpretado como un comodín, sino como un carácter normal, lo tenemos que encerrar entre corchetes o utilizar un carácter de escape.

[ESCAPE 'car_escape']

La cláusula ESCAPE es opcional y permite definir un carácter de escape.

Un carácter de escape es un carácter que se coloca delante de un carácter comodín para indicar que el comodín no debe interpretarse como tal, sino como un carácter normal.

Por ejemplo queremos buscar los nombres compuestos que incluyen un subrayado. En este caso tenemos que poner el carácter _ como un carácter normal no como un comodín, así que lo escribiremos así:

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '%[_]%' ;
```

O bien,

```
SELECT numemp, nombre
FROM empleados
WHERE nombre LIKE '%!_%' ESCAPE '!';
```

2.7.2 Condiciones de búsqueda compuestas.

En una cláusula WHERE podemos incluir una condición de búsqueda simple (formada por un solo predicado) o compuesta (formada por la combinación de predicados unidos por los operadores lógicos NOT, AND, OR).

Cuando la condición incluye varios operadores lógicos, el orden de prioridad de estos operadores es: NOT (el más alto), seguido de AND y OR (al mismo nivel). Como siempre, se pueden utilizar paréntesis para alterar esta prioridad en una condición de búsqueda.

El orden de evaluación de los operadores lógicos puede variar dependiendo de las opciones elegidas por el optimizador de consultas.

Los operadores lógicos pueden devolver tres valores distintos: TRUE, FALSE, NULL (UNKNOWN).

Tablas de verdad de los operadores:

AND Combina dos condiciones y se evalúa como TRUE cuando ambas condiciones son TRUE.

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR Combina dos condiciones y se evalúa como TRUE cuando alguna de las condiciones es TRUE.

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT Niega la expresión booleana que especifica el predicado

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

Hallar los vendedores que están por debajo de su cuota y tienen ventas inferiores a 30,000.

```
SELECT nombre
FROM empleados
WHERE ventas < cuota AND ventas < 300000;
```

Hallar los vendedores que están debajo de su cuota, pero cuyas ventas no son inferiores a 150,000.

```
SELECT nombre
FROM empleados
WHERE ventas < cuota AND ventas < 300000;
```

Hallar las oficinas no dirigidas por el empleado 108

```
SELECT oficina
FROM oficinas
WHERE NOT dir = 108;
```

O

```
SELECT oficina
FROM oficinas
WHERE dir <> 108;
```

Devuelven:

oficina
11
12
13

Las oficinas sin director no aparecen, para que aparezcan deben añadir otro predicado:

```
SELECT oficina, dir
FROM oficinas
WHERE NOT dir = 108 or dir is null;
```

<u>Oficina</u>	<u>dir</u>
11	106
12	104
13	105
26	NULL
28	NULL