ÍNDICE

1. Introdución

Esta unidade ten como finalidade aprender a recuperar información de calquera táboa dunha base de datos relacional usando a linguaxe SQL.

Usaremos o SXBD (Sistema Xestor de Base de Datos) SQL Server, polo que a sintaxe que se explicará será a da linguaxe Transact-SQL (T-SQL en adiante), propia do xestor.

A ferramenta gráfica que imos empregar será o SQL Server Management Studio (SSMS en adiante), tanto o editor de consultas como o asistente.

Antes de empezar estableceremos as convencións de escritura para as distintas sintaxes.

Convencións de escritura da sintaxe SQL		
Cláusulas	En maiúsculas	`SELECT, FROM, WHERE, HAVING...`
Predicados	En maiúsculas	`NOT IN, BETWEEN, ANY, SOME...`
Funcións nas consultas	En minúsculas	`isnull, cast...`
Funcións no texto	En cursiva	*`isnull, cast...`*
Palabra reservada para os alias	En minúsculas	`as`
Outras palabras reservadas	En maiúsculas	`ASC, DESC, TOP, DISTINCT...`
Nomes das bases de datos	En maiúsculas	`EMPRESA, SOCIEDADE_CULTURAL...`
Nomes das táboas	En maiúsculas	`ACTIVIDADE, AULA, EMPREGADO...`
Nomes das columnas nas consultas	En minúsculas	`descricion, nome, data_ini,...`
Nomes das columnas no texto	En minúsculas e cursiva	*`descricion, nome, data_ini,...`*
Constantes de texto	En maiúscula	`ADMINISTRATIVO, NON DIRIXE...`
[]	O contido dos corchetes é opcional.	
{}	Entre chaves colócanse listas de valores ou expresións.	
|	Separa distintas opcións a escoller.	

Bases de datos de traballo

Para seleccionar a BD de traballo como por exemplo a BD SOCIEDADE_CULTURAL, bastará con facer unha soa instrución USE:

```
--Seleccionamos a BD SOCIEDADE_CULTURAL
USE SOCIEDADE_CULTURAL;
```

- **Resultado:** Comprobaremos como despois de executar a sentenza, na lista despregable do SSMS aparece o nome da BD SOCIEDADE_CULTURAL.

2. Consultas simples

Explicarase a sintaxe T-SQL para a realización de consultas simples sobre unha única táboa coas seguintes características:

- Consultas sen predicado (sen WHERE).
- Con predicado (con WHERE).
- Con ordenación (ORDER BY) ascendente e descendente.
- Con eliminación de filas repetidas.
- Con operadores aritméticos.
- Operadores de cadeas de carácteres.
- Predicados básicos: NULL, BETWEEN, LIKE, IN e predicado EXISTS.
- Predicados compostos.

2.1. A sublinguaxe DML

A linguaxe SQL, divídese en varias sublinguaxes, en función das operacións que permiten facer coa información das bases de datos.

Para poder recuperar información da BD empregaremos consultas, *queries*, ou tamén chamadas *selects*, pola palabra reservada pola que estas comezan. Para deseñar estas consul-tas utilízase a sublinguaxe DML (*Data Management Language*), ou LMD (*Linguaxe de Manipulación de Datos*).

Estas consultas escribiranse no editor de consultas do SSMS. Este editor permite codifi-car, comprobar a sintaxe, executar e visualizar o resultado das nosas consultas.

Veremos tamén o uso do asistente para realizar consultas.

2.2. Sintaxe das consultas simples

Agora estudarase polo miúdo a sintaxe das consultas que afectan a unha única táboa dunha BD, consultas coñecidas como consultas simples.

Na seguinte táboa amósase a sintaxe das consultas simples.

Sintaxe T-SQL

`SELECT [TOP X [PERCENT]] col1 [, col2,...., coln] | *`
`FROM tabla`
`[WHERE predicado ]`
`[ORDER BY col1  [ASC | DESC] [, col2 [ASC | DESC], ..., coln [ASC | DESC]]];`

- **SELECT:** Nesta primeira cláusula aparecerán os nomes das columnas da táboa das que queremos amosar o contido no resultado da consulta. Deben separarse por comas.

  Se poñemos * obteremos o contido de todas as columnas da táboa.

  - **TOP:** Cando se pretende obter un número determinado de filas, ou unha porcentaxe delas, usaremos TOP. Por exemplo, SELECT TOP 5 col1, col2... devolvería só as 5 primeiras filas do resultado, e SELECT TOP 50 PERCENT col1, col2... devolvería a metade das filas do resultado, é dicir o 50 por cento.
- **FROM:** Escribirase o nome da táboa da que queremos consultar a información.
- **WHERE:** Cláusula opcional (na sintaxe está indicado cos corchetes) na que se deben poñer as condicións que deben cumprir os datos que amosará a consulta.
- **ORDER BY:** Cláusula opcional na que poremos unha listaxe separada por comas das columnas polas que queremos ordenar os resultados. Indicarase:
  - **ASC:** se queremos ordenar ascendentemente (da A-Z ou números de menor a maior). É a opción por defecto, polo que se non a indicamos, no resultado da consulta a columna ordenarase dese xeito.
  - **DESC:** se queremos que unha ou máis columnas aparezan ordenadas en orde descendente deberemos obrigatoriamente a continuación do nome de cada unha delas, indicar a palabra DESC.
- Remataremos todas as sentenzas SQL nun punto e coma (conforme o estándar ANSI SQL). Nalgúns Sistemas Xestores de Bases de Datos Relacionais (SXBDR en adiante) como é o caso do MS SQL Server, pódense escribir as consultas sen o punto e coma final, pero é unha boa práctica xa que ademais é o xeito habitual en programación de indicar a fin dunha instrución.

2.3. Consultas sen predicado WHERE

As consultas simples sen predicado WHERE son aquelas que amosan datos dunha táboa sen impoñer restricións. Só se compoñen das cláusulas SELECT e FROM, as únicas obrigatorias.

Na cláusula FROM indicaremos o nome da táboa que contén a información a recuperar, e na cláusula SELECT as columnas ou expresións constantes ou variables, que queremos obter desa táboa separadas por comas.

Os nomes das columnas e das táboas pódese evitar escribilos arrastrándoos desde o explorador de obxectos.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 1, 2 e 3, no editor de consultas do SSMS.

- **Consulta de exemplo 1:** Amosaremos o nome de todas as actividades da BD, e para iso indicamos no SELECT o campo que queremos obter (neste caso o nome), e no FROM indicamos a táboa na que está almacenada a información:

```
SELECT nome
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 1
nome
TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ INICIACIÓN Á INFORMÁTICA

- **Consulta de exemplo 2:** Como na consulta anterior, amosaremos o nome de todas as actividades da BD, e ademais o prezo. Para iso repetimos a consulta anterior pero no SELECT engadimos o nome do campo separado por unha coma do campo nome (*nome, prezo*):

```
SELECT nome, prezo
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 2	
nome	**prezo**
TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ INICIACIÓN Á INFORMÁTICA	301.55 50.00 80.00 0.00

- **Consulta de exemplo 3:** Como na consulta anterior, amosaremos o código, nome e prezo de todas as cotas, é dicir, todos os campos da táboa COTA, onde gardamos as cotas da Sociedade cultural e deportiva:

```
--Solución1
SELECT codigo, nome, importe --SINTAXE RECOMENDADA
FROM COTA;
--Solución2
SELECT *
FROM COTA;
```

Neste caso propóñense dúas solucións, unha enumerando todos os campos no SELECT, e outra empregando o símbolo * (asterisco). Cando nunha consulta no SELECT temos que amosar todos os campos dunha táboa pódese empregar *, pero o recomendable é indicar a listaxe completa dos campos, xa que así sabemos exactamente o que imos amosar no resultado. E nunca se debe empregar cando as consultas están embebidas no código dun programa, xa que, se por exemplo engadimos unha columna na táboa, cando fagamos SELECT * FROM COTA, o código do programa espera 3 columnas cando en realidade recibe 4, co conseguinte erro na aplicación.

Resultado da consulta do exemplo 3		
codigo	**nome**	**importe**
11	TENIS PARA PRINCIPIANTES	301.55
12	REPOSTARÍA	50.00
13	XADREZ	80.00
99	INICIACIÓN Á INFORMÁTICA	0.00

2.4. Consultas con predicado (con WHERE)

Esta cláusula opcional permítenos restrinxir aos resultados da nosa consulta, xa que só devolverá os datos da táboa que cumpran a condición (ou condicións) do WHERE.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 4, no editor de consultas do SSMS.

- **Consulta de exemplo 4:** Nesta consulta engadiremos a cláusula WHERE. A consulta de exemplo amosará nome, prezo e prazas das actividades organizadas pola Sociedade cultural e deportiva que teñan un prezo inferior a 100€:

```
SELECT nome, prezo, num_prazas
FROM ACTIVIDADE
WHERE prezo < 100;
```

Resultado da consulta do exemplo 4		
nome	**prezo**	**num_prazas**
REPOSTARÍA	50.00	20
XADREZ	80.00	10
INICIACIÓN Á INFORMÁTICA	0.00	20

Aínda que os operadores de comparación non se ven ata o apartado de Predicados básicos, para entender esta consulta chegará con entender que o símbolo < da condición prezo < 100, indica que o prezo debe ser *menor que* 100.

2.5. Consultas con ordenación (con ORDER BY)

Con esta cláusula opcional conséguese que o resultado apareza ordenado pola columna ou columnas que indiquemos na mesma.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 5 e 6, no editor de consultas do SSMS.

- **Consulta de exemplo 5:** Agora repetirase a consulta anterior, exemplo 4, pero engadindo a cláusula ORDER BY. Neste caso ao poñer ao lado ca columna DESC

estamos indicando que queremos ordenar o resultado polo prezo en orde DESCendente, de xeito que aparecerán primeiro as actividades con maior prezo.

```
SELECT nome, prezo, num_prazas
FROM ACTIVIDADE
WHERE prezo < 100
ORDER BY prezo DESC;
```

Resultado da consulta do exemplo 5		
nome	**prezo**	**num_prazas**
XADREZ REPOSTARÍA INICIACIÓN Á INFORMÁTICA	80.00 50.00 0.00	10 20 20

A continuación imos ver como afectaría ao resultado a inclusión de TOP no SELECT.

```
--Modificación con TOP 1
SELECT TOP 1 nome, prezo, num_prazas
FROM ACTIVIDADE
WHERE prezo < 100
ORDER BY prezo DESC;
```

Resultado da consulta do exemplo 5		
nome	**prezo**	**num_prazas**
XADREZ	80.00	10

Obsérvase como agora no resultado aparece só a primeira fila do resultado, por ter indicado TOP 1.

A modificación seguinte consiste en devolver só o 50% das filas do resultado.

```
--Modificación con TOP 50 PERCENT
SELECT TOP 50 PERCENT nome, prezo, num_prazas
FROM ACTIVIDADE
WHERE prezo < 100
ORDER BY prezo DESC;
```

Resultado da consulta do exemplo 5		
nome	**prezo**	**num_prazas**
XADREZ REPOSTARÍA	80.00 50.00	10 20

Como se pode observar o xestor aproxima por exceso a devolución de filas. Concretando para este exemplo que devolvía 3 filas sen TOP, se calculamos o 50% de 3 devolvería 1'5 filas, pero como non é posible devolver unha fila e media, aproxima por exceso e devolve 2.

- **Consulta de exemplo 6:** Nesta consulta comprobaremos como se pode ordenar un resultado por varias columnas. Na consulta obteranse o nome, prezo e número de prazas das actividades con prezo inferior a 100€. O resultado deberá aparecer

ordenado primeiro de maior a menor número de prazas, e no caso de que coincida ese número, ordenarase en orde alfabética do nome da actividade.

```
--Solución1
SELECT nome, prezo, num_prazas
FROM ACTIVIDADE
WHERE prezo < 100
ORDER BY num_prazas DESC, nome; --SINTAXE RECOMENDADA
--Solución2
SELECT nome, prezo, num_prazas
FROM ACTIVIDADE
WHERE prezo < 100
ORDER BY 3 DESC, 1;
--Solución3
SELECT nome, prezo, num_prazas
FROM ACTIVIDADE
WHERE prezo < 100
ORDER BY num_prazas DESC, nome ASC;
```

Propoñemos tres solucións para que se observe o seguinte:

- na cláusula ORDER BY na segunda solución, no canto do nome dos campos indícase a posición que ocupan os mesmos na cláusula SELECT, a terceira posición no caso do número de prazas e a primeira no caso do nome da actividade. Aconséllase empregar o nome do campo e non a súa posición, xa que en consultas de moitas columnas faise moito máis complexo saber por que campos se está ordenando o resultado.
- Na terceira solución o campo que queremos que apareza ordenado alfabeticamente, o nome, vai acompañado da abreviatura ASC de ASCendente. Se a comparamos coa primeira solución vemos que ao ser a *opción por defecto* non é necesario empregar esa abreviatura.

Resultado da consulta do exemplo 6		
nome	**prezo**	**num_prazas**
INICIACIÓN Á INFORMÁTICA REPOSTARÍA XADREZ	0.00 50.00 80.00	20 20 10

2.6. Consultas con eliminación de filas repetidas (con DISTINCT)

Cando nas nosas consultas queremos eliminar os resultados repetidos, debemos engadir a palabra reservada DISTINCT xusto despois da palabra SELECT (deixando un espazo en branco entre ás dúas).

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 7, no editor de consultas do SSMS.

- **Consulta de exemplo 7:** Nesta consulta queremos buscar os tipos de socios que temos na nosa BD. Para iso temos que consultar a columna tipo da táboa SOCIO. Imos facer a consulta con e sen DISTINCT para comprobar como os resultados son distintos.

```
--Solución1
SELECT tipo
FROM SOCIO;
--Solución2
SELECT DISTINCT tipo
FROM SOCIO;
```

Resultado da consulta do exemplo 7. Solución 1
tipo
HONORÍFICO FAMILIAR COMÚN HONORÍFICO
Resultado da consulta do exemplo 7. Solución 2
tipo
COMÚN FAMILIAR HONORÍFICO

Coa primeira solución vemos como ao existir dous socios de tipo honorífico, aparece dúas veces. Na segunda solución ao engadir DISTINCT eliminouse do resultado o tipo de socio repetido.

2.7. Consultas con operadores aritméticos

Nas consultas de T-SQL empregaranse os seguintes símbolos correspondentes as seguintes operacións aritméticas:

- Suma: operador +
- Resta: operador –
- Multiplicación: operador *
- División: operador /
- Módulo: operador %. Este operador devolve o resto de dividir dous números.

Os operadores aritméticos poderán usarse tanto na cláusula SELECT como na cláusula WHERE.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 8 e 9, no editor de consultas do SSMS.

- **Consulta de exemplo 8:** Supoñendo que a cada empregado da Sociedade Cultural e Deportiva se lle retén do seu soldo un 21%, na seguinte consulta devólvese o nome,

primeiro apelido, segundo apelido e o salario neto de cada un dos empregados. Neste caso empregamos os operadores aritméticos no SELECT.

```
--Solución1
SELECT nome, ape1, ape2, salario_mes-(salario_mes*0.21)
FROM EMPREGADO;
--Solución2
SELECT nome, ape1, ape2, salario_mes*0.79
FROM EMPREGADO;
--Solución3
SELECT nome, ape1 as "primeiro apelido",
       ape2 as "segundo apelido",
       salario_mes*0.79  as salario_neto
FROM EMPREGADO;
```

Resultado da consulta do exemplo 8. Solución 1

nome	ape1	ape2	(Sin nombre de columna)
MARÍA	GARCÍA	PÉREZ	711.0000
CARLOS	REGO	PENA	632.0000
JUANA	POSE	VARELA	1185.7110
JOSÉ	GONZÁLEZ	ÍNSUA	474.0000

Resultado da consulta do exemplo 8. Solución 2

nome	ape1	ape2	(Sin nombre de columna)
MARÍA	GARCÍA	PÉREZ	711.0000
CARLOS	REGO	PENA	632.0000
JUANA	POSE	VARELA	1185.7110
JOSÉ	GONZÁLEZ	ÍNSUA	474.0000

Resultado da consulta do exemplo 8. Solución 3

nome	primeiro apelido	segundo apelido	salario_neto
MARÍA	GARCÍA	PÉREZ	711.0000
CARLOS	REGO	PENA	632.0000
JUANA	POSE	VARELA	1185.7110
JOSÉ	GONZÁLEZ	ÍNSUA	474.0000

Na segunda solución a única diferenza é que para obter o salario neto sacamos factor común.

Nas dúas primeiras consultas como nome da cuarta columna na que se fai un cálculo, o xestor indica *(Sin nome de columna)*, xa que o dato que se obtén non é o valor dunha columna, senón o resultado dunha operación. Para evitar isto e darlle un nome á columna que se corresponda coa información da mesma, na terceira solución empregamos un *alias*. Para facelo, despois da columna á que lle queremos cambiar o nome engadimos a palabra *as* e o nome novo.

Por compatibilidade con outros xestores, empregaremos sempre a palabra *as* entre a expresión e o nome novo, aínda que en SQL Server tamén sexa correcta a sintaxe sen

especificar *as*.

Tamén se poden usar alias para darlle un nome diferente a calquera columna, como se pode observar no caso das columnas dos apelidos. Cando no nome do alias se inclúan espazos en branco, este nome deberá poñerse entre comiñas.

- **Consulta de exemplo 9:** A continuación verase un exemplo de como se poden usar os operadores aritméticos no WHERE. Nesta consulta devólvese o nome e importe das cotas que cumpran que o dobre do seu importe é maior a 90€.

```
SELECT nome, importe
FROM COTA
WHERE 2*importe >90;
```

Resultado da consulta do exemplo 9	
nome	**importe**
DE HONRA HABITUAL	100.00 50.00

Neste exemplo comprobouse como se poden empregar operadores matemáticos tamén na cláusula WHERE.

2.8. Consultas con operadores de cadeas de caracteres

Concatenación. Símbolo +

Para a concatenación de cadeas de caracteres, é dicir de datos alfanuméricos, en SQL Server empregarase o mesmo símbolo que para a operación matemática da suma, o símbolo +. No ANSI SQL ou SQL estándar utilízase o operador dobre barra **||.**

As cadeas de caracteres constantes deben indicarse entre comiñas simples.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 10, no editor de consultas do SSMS.

- **Consulta de exemplo 10:** Esta consulta de exemplo amosa nunha soa columna de nome *nome_completo_empregados,* o nome completo de cada empregado co formato:

  apelido1 apelido2, nome

```
SELECT ape1+' '+ape2+', '+nome as nome_completo_empregados
FROM EMPREGADO;
```

Resultado da consulta do exemplo 10
nome_completo_empregados
GARCÍA PÉREZ, MARÍA REGO PENA, CARLOS POSE VARELA, JUANA GONZÁLEZ ÍNSUA, JOSÉ

Para consultar a concatenación a través das funcións *concat* e *concat_ws* pódese acceder aos libros de axuda online de SQL Server na seguinte ligazón https://learn.microsoft.com/es-es/sql/t-sql/functions/concat-transact-sql?view=sql-server-ver16

Funcións de conversión cast e convert

No caso de que interese concatenar cadeas de caracteres con datos que non son cadeas, como datas ou números, deberemos facer que SQL Server transforme os números en cadeas, porque senón o sistema ao detectar o símbolo + ao carón de números tentará facer unha suma. Para facer esta conversión de tipos deberase utilizar a función *cast* ou a función *convert*

Tanto a función *cast* como a función *convert* empréganse para a conversión de tipos de datos. A continuación explícase a súa sintaxe e exemplos do seu uso.

- **Función cast**:

Sintaxe función cast
`cast(expresión AS tipo_dato)`

A función *cast* transforma a *expresión* indicada entre parénteses no *tipo_dato* indicado despois do AS.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 11 e 12, no editor de consultas do SSMS.

- **Consulta de exemplo 11:** Esta consulta de exemplo amosa nunha soa columna de nome *datos_empregados,* o número identificador e o nome completo de cada empregado co formato:

  número identificador - apelido1 apelido2, nome

```
--Solución CON ERROS
SELECT numero+' - '+ape1+' '+ape2+', '+nome as datos_empregados
FROM EMPREGADO;
--Solución CORRECTA
SELECT cast(numero as varchar(5))+' - '+ape1+' '+ape2+', '+nome
       as datos_empregados
FROM EMPREGADO;
```

  Na primeira solución, sen empregar a función *cast*, o xestor intenta facer unha suma e transformar o valor de ape1 en *varchar*, por iso devolve o seguinte erro:

```
Error de conversión al convertir el valor varchar ', ' al tipo de datos int.
```

  Na segunda solución ao utilizar a función *cast* para transformar o número nunha cadea de caracteres, o xestor fai correctamente a concatenación. A solución correcta

devolverá o seguinte resultado.

Resultado da consulta do exemplo 11
datos_empregados
100 - GARCÍA PÉREZ, MARÍA 200 - REGO PENA, CARLOS 300 - POSE VARELA, JUANA 400 - GONZÁLEZ ÍNSUA, JOSÉ

- **Función convert**:

Sintaxe función convert
`convert(tipo_dato, expresión [,estilo])`

A función *convert* transforma a *expresión* no *tipo_dato* indicado como primeiro parámetro xusto despois dos parénteses.

O valor do terceiro parámetro *estilo* dependerá do tipo de dato que indiquemos. Utilizarase principalmente nos tipos de dato de datas. Permítenos escoller o formato co que representar unha data como dd/mm/aaaa, dd-mm-aaaa, dd mes aa. Verémolo no seguinte exemplo.

- **Consulta de exemplo 12:** Esta consulta de exemplo amosa nunha soa columna o nome de todas as actividades e o día que comezan co formato:

  nome_actividade comeza o *data_inicio_da_actividade*

```
--Solución1
SELECT nome+' comeza o '+convert(char(20),data_ini)
       as acti_sen_estilo_data
FROM ACTIVIDADE;
--Solución2
SELECT nome+' comeza o '+convert(char(20),data_ini,103)
       as acti_estilo_data_103
FROM ACTIVIDADE;
--Solución3
SELECT nome+' comeza o '+convert(char(20),data_ini,113)
       as acti_estilo_data_113
FROM ACTIVIDADE;
--Solución4
SELECT nome+' comeza ás '+convert(char(20),data_ini,108)
       as acti_estilo_data_108
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 12. Solución 1
acti_sen_estilo_data
TENIS PARA PRINCIPIANTES comeza o Feb 10 2014 4:00PM REPOSTARÍA comeza o Feb 15 2015 5:00PM XADREZ comeza o Mar 20 2014 4:30PM INICIACIÓN Á INFORMÁTICA comeza o Mar 1 2015 4:30PM
Resultado da consulta do exemplo 12. Solución 2
acti_sen_estilo_data_103
TENIS PARA PRINCIPIANTES comeza o 10/02/2014 REPOSTARÍA comeza o 15/02/2015 XADREZ comeza o 20/03/2014 INICIACIÓN Á INFORMÁTICA comeza o 01/03/2015
Resultado da consulta do exemplo 12. Solución 3
acti_sen_estilo_data_113
TENIS PARA PRINCIPIANTES comeza o 10 Feb 2014 16:00:00 REPOSTARÍA comeza o 15 Feb 2015 17:00:00 XADREZ comeza o 20 Mar 2014 16:30:00 INICIACIÓN Á INFORMÁTICA comeza o 01 Mar 2015 16:30:00
Resultado da consulta do exemplo 12. Solución 4
acti_sen_estilo_data_108
TENIS PARA PRINCIPIANTES comeza ás 16:00:00 REPOSTARÍA comeza ás 17:00:00 XADREZ comeza ás 16:30:00 INICIACIÓN Á INFORMÁTICA comeza ás 16:30:00

Na primeira solución, non indicamos ningún estilo, polo que amosa o formato predeterminado que se corresponde co número de *estilo* 0 ou 100.

Nas seguintes solucións, en función do *estilo* indicado no *convert*, así se amosa a data cun formato ou outro, con hora ou sen ela, ou a hora unicamente.

Para consultar o resto dos estilos posibles da función *convert* pódese acceder aos libros de axuda online de SQL Server na seguinte ligazón https://msdn.microsoft.com/es-es/library/ms187928

2.9. Predicados

Un predicado expresa unha condición entre valores, e segundo sexan estes, pode resultar *verdadeiro*, *falso* ou *descoñecido*.

Os predicados especifícanse na cláusula WHERE, e noutras cláusulas que veremos máis adiante noutras actividades da unidade 5.

2.9.1. Predicados básicos

Son predicados simples que expresan condicións de comparación entre dous valores. Especifícanse cos signos =, >, < ou as combinacións deles que vemos a continuación:

PREDICADO	É verdadeiro ⇔ (si e só si)
X = Y	X é igual a Y
X <> Y (ou !=)	X es distinto de Y (tamén é válido o símbolo != en SQL Server)
X < Y	X é menor que Y
X > Y	X é maior que Y
X <= Y	X é menor ou igual que Y
X >= Y	X é maior ou igual que Y

Se algún dos dous comparandos, X ou Y, ou ambos, son NULOS, o resultado do predicado será *descoñecido*.

Os comparandos poden ser expresión, non teñen porque ser nomes de campos simplemente.

O segundo comparando pode ser o resultado dunha consulta SELECT, que deberá ir sempre entre parénteses e producir un resultado único (unha soa fila cunha soa columna). Este tipo de consultas denomínanse *consultas subordinadas* ou *subconsultas*.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 13, no editor de consultas do SSMS.

- **Consulta de exemplo 13:** Queremos obter o nome das cotas non gratuítas. Neste caso farase a consulta buscando as cotas con importe diferente a 0 e como xa se indicou na táboa anterior, pódese usar o símbolo < > ou o símbolo !=

```
--Solución1
SELECT nome
FROM COTA
WHERE importe<>0;
--Solución2
SELECT nome
FROM COTA
WHERE importe!=0;
```

Nos dous casos devolve o mesmo.

Resultado da consulta do exemplo 13
nome
DE HONRA FAMILIAR HABITUAL

2.9.2. Uso de NULL

Nunha BD existirán campos baleiros ou sen información coñecida. Por exemplo, cando descoñecemos a profesión dun socio non debemos gardar a frase *Sen profesión*, senón que non gardaremos ningún dato e o sistema asignaralle o valor NULL. Para que isto sexa posible o campo ten que estar definido na táboa como opcional (NULL).

Cando queremos buscar datos nulos non podemos empregar nin o operador de igualdade =. O mesmo ocorrerá cando busquemos datos que non sexan nulos, para o que tampouco poderemos usar os operadores de desigualdade < > ou !=.

É moi importante lembrarse disto, xa que se usamos estes operadores o xestor SQL Server non da erro e podemos pensar que a consulta está ben realizada, cando non é así.

A sintaxe correcta será a seguinte:

Sintaxe NULL
`nome_columna IS [NOT] NULL`

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 14 e 15, no editor de consultas do SSMS.

- **Consulta de exemplo 14:** Buscamos os nomes e apelidos dos socios dos que descoñecemos a súa profesión, ordenados alfabeticamente por apelidos e nome.

```
--Solución INCORRECTA
SELECT ape1, ape2, nome
FROM SOCIO
WHERE profesion=NULL
ORDER BY ape1, ape2, nome;
--Solución CORRECTA
SELECT ape1, ape2, nome
FROM SOCIO
WHERE profesion IS NULL
ORDER BY ape1, ape2, nome;
```

Resultado da consulta do exemplo 14. Solución 1		
ape1	**ape2**	**nome**

Resultado da consulta do exemplo 14. Solución 2		
ape1	**ape2**	**nome**
GRAÑA	UMIA	MARÍA

Podemos observar como a primeira solución non devolve ningún erro, pero tampouco

dato algún co que poderíase pensar que a consulta é correcta e que non hai socios sen profesión coñecida, cando en realidade hai unha socia sen profesión, como comprobamos ao executar a consulta da segunda solución.

- **Consulta de exemplo 15:** Buscaremos agora xusto o contrario, os nomes e apelidos dos socios dos que si coñecemos a súa profesión, ordenados alfabeticamente por apelidos e nome.

```
--Solución INCORRECTA
SELECT ape1, ape2, nome
FROM SOCIO
WHERE profesion!=NULL
ORDER BY ape1, ape2, nome;
--Solución CORRECTA
SELECT ape1, ape2, nome
FROM SOCIO
WHERE profesion IS NOT NULL
ORDER BY ape1, ape2, nome;
```

Resultado da consulta do exemplo 15. Solución 1		
ape1	**ape2**	**nome**
Resultado da consulta do exemplo 15. Solución 2		
ape1	**ape2**	**nome**
DEL CARMEN SIEIRO VIEITO	LÉREZ CAMPOS GIL	JORGE MANUEL CARLA

Igual que na consulta de exemplo 14, a primeira solución non devolve ningún erro, pero tampouco dato algún co que poderíase pensar que a consulta é correcta e que non coñecemos a profesión de ningún dos nosos socios. Ao executar a segunda consulta comprobamos como de 3 dos nosos socios gardamos na BD a súa profesión.

2.9.3. Predicado BETWEEN

Este predicado usarase para saber se un valor está comprendido, ou non, entre outros dous, ambos incluídos. Eses valores poden ser tamén datas, sempre que expresión1 sexa tamén unha data.

Sintaxe BETWEEN
`expresión1 [NOT] BETWEEN expresión2 AND expresión3`

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 16 e 17, no editor de consultas do SSMS.

- **Consulta de exemplo 16:** Búscanse os nomes e importes das cotas con importe entre 50 e 200, ambos inclusive.

```
SELECT nome, importe
FROM COTA
WHERE importe BETWEEN 50 AND 200;
```

Resultado da consulta do exemplo 16	
nome	**importe**
DE HONRA HABITUAL	100.00 50.00

Esta consulta devolve as cotas con importe no rango [50-200]. Os corchetes nos extremos indican que o número está incluído no rango.

- **Consulta de exemplo 17:** Búscanse os nomes e importes das cotas con importe menor a 50 ou maior a 200.

```
SELECT nome, importe
FROM COTA
WHERE importe NOT BETWEEN 50 AND 200;
```

Resultado da consulta do exemplo 17	
nome	**importe**
FAMILIAR GRATUITA	30.00 0.00

Esta consulta devolve as cotas con importe fóra do rango [50-200]. Os corchetes nos extremos indican que o número está incluído no rango.

2.9.4. Predicado LIKE

O predicado LIKE determina se unha cadea de caracteres coincide con un modelo especificado. Devolverá VERDADEIRO se a expresión coincide co modelo, ou FALSO se non coincide.

Sintaxe LIKE
`expresión [NOT] LIKE modelo [ESCAPE carácter_de_escape]`

- **expresión:** É calquera expresión alfanumérica.
- **modelo:** O modelo é unha constante alfanumérica (encerrada entre comiñas simples) que pode conter caracteres normais e caracteres comodíns. Os comodíns, que se poden usar varios no mesmo modelo, son:

CARÁCTER COMODÍN	DESCRICIÓN	EXEMPLO
%	O tanto por cen substitúe a 0 ou máis caracteres.	`WHERE nome LIKE '%tenis%' busca as actividades con nomes que conteñan a palabra tenis.`

_	O guión baixo substitúe a 1 carácter	`WHERE ape1 LIKE '_az' busca os socios con primeiros apelidos de 3 letras que rematen en az.`
[ ]	Calquera carácter individual do intervalo ([a-f]) ou do conxunto ([abcdef]) que se especificou.	`WHERE ape1 LIKE '[P-T]az' busca socios con primeiros apelidos de 3 letras que rematen en az e comecen por calquera letra entre P e Z.`
[^]	Calquera carácter individual que non se atope no intervalo ([^a-f]) ou o conxunto ([^abcdef]) que se especificou.	`WHERE ape1 LIKE '[^P]az' busca socios con primeiros apelidos de 3 letras que rematen en az e non comecen por P.`

- **Carácter de escape:** É un carácter que se pon diante dun carácter comodín para indicar que este non debe interpretarse como un comodín, senón como un carácter normal. Por exemplo WHERE descricion LIKE '%_%' ESCAPE '\', buscará as aulas con descricións que conteñan o carácter '_' guión baixo.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo da 18 á 22, no editor de consultas do SSMS.

- **Consulta de exemplo 18:** Buscamos o NIF, nome completo e nome da vía dos socios que viven en vías con nome que comece pola palabra SAN. Neste exemplo vese como utilizar o comodín %.

```
SELECT nome, ape1, ape2, nome_via_enderezo
FROM SOCIO
WHERE nome_via_enderezo LIKE 'SAN %';
```

Resultado da consulta do exemplo 18			
nome	**ape1**	**ape2**	**nome_via_enderezo**
MANUEL CARLA	SIEIRO VIEITO	CAMPOS GIL	SAN ROQUE SAN ROQUE

- **Consulta de exemplo 19:** Nome, superficie e estado das aulas cuxa descrición empeza pola palabra *aula*, ten só 8 caracteres e remata na letra R. Neste exemplo comprobamos o uso do comodín _.

```
SELECT descricion, superficie, estado
FROM AULA
WHERE descricion LIKE 'AULA___R';--Hai 3 guións baixos no modelo
```

Resultado da consulta do exemplo 19		
descricion	**superficie**	**estado**
AULA SUR	80	M

- **Consulta de exemplo 20:** Nesta consulta buscamos o NIF, nome e apelidos dos socios cuxo primeiro apelido comece por C, D, E, F ou G. Usaremos o comodín dos corchetes no LIKE.

```
SELECT nif, nome, ape1, ape2
FROM SOCIO
WHERE ape1 LIKE '[C-G]%';
```

Resultado da consulta do exemplo 20			
nif	**nome**	**ape1**	**ape2**
11111112A 33333334C	MARÍA JORGE	GRAÑA DEL CARMEN	UMIA LÉREZ

- **Consulta de exemplo 21:** Agora, ao contrario que na anterior consulta, buscamos o NIF, nome e apelidos dos socios cuxo primeiro apelido non comece por C, D, E, F ou G. Usaremos o comodín dos corchetes no LIKE, pero agora co símbolo ^.

```
SELECT nif, nome, ape1, ape2
FROM SOCIO
WHERE ape1 LIKE '[^C-G]%';
```

Resultado da consulta do exemplo 21			
nif	**nome**	**ape1**	**ape2**
22222223B 44444445D	MANUEL CARLA	SIEIRO VIEITO	CAMPOS GIL

- **Consulta de exemplo 22:** Nesta consulta buscamos o nome, a data de inicio con formato dd/mm/aaaa e as observacións daquelas actividades con observación con _ (guión baixo). Para que o símbolo _ (guión baixo) non sexa considerado un comodín, deberemos poñer antes do símbolo un carácter de escape, neste caso \.

```
SELECT nome, convert(char(10),data_ini,103) as data_ini,
       observacions
FROM ACTIVIDADE
WHERE observacions LIKE '%\_%' ESCAPE '\';
```

Resultado da consulta do exemplo 22		
nome	**data_ini**	**observacions**
INICIACIÓN Á INFORMÁTICA	01/03/2015	Impartirase SW_libre

2.9.5. Predicado IN

Sintaxe1 IN
`expresión [NOT] IN (valor1 [,valor2,...,valorN])`

Este predicado devolve VERDADEIRO se a expresión ten como valor un dos da lista especificada entre parénteses.

Se indicamos NOT IN devolve VERDADEIRO se a expresión ten como valor un distinto dos da lista.

Poderíase indicar un único valor entre parénteses, pero sería equivalente a usar o símbolo igual (*expresión IN (valor)* é o mesmo que *expresión=valor*)

Sintaxe2 IN
`expresión [NOT] IN (consulta subordinada)`

Nesta segunda sintaxe, o predicado IN devolverá VERDADEIRO se a expresión ten como valor un dos devoltos pola consulta subordinada escrita dentro dos parénteses. É importante que a consulta subordinada devolva só unha columna.

Este tipo de consultas veranse en detalle na actividade 5 de Consultas con subconsultas, desta mesma unidade.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 23 e 24, no editor de consultas do SSMS.

- **Consulta de exemplo 23:** Nesta consulta buscamos o nome e prezo das actividades que teñan prezo 50 ou 80.

```
SELECT nome, prezo
FROM ACTIVIDADE
WHERE prezo IN (50,80);
```

Resultado da consulta do exemplo 23	
nome	**prezo**
REPOSTARÍA XADREZ	50.00 80.00

- **Consulta de exemplo 24:** Amosarase a descrición, a superficie e o estado das aulas con estado diferente a bo (B) ou regular (R). Como se pode ver nesta consulta os valores cos que comparar a expresión poden ser tamén alfanuméricos.

```
SELECT descricion, superficie, estado
FROM AULA
WHERE estado NOT IN ('B','R');
```

Resultado da consulta do exemplo 24		
descricion	**superficie**	**estado**
AULA SUR	80	M

2.9.6. Predicado EXISTS

Se indicamos NOT IN devolve VERDADEIRO se a expresión ten como valor un distinto dos da lista.

Sintaxe EXISTS
`[NOT] EXISTS (consulta subordinada)`

O predicado EXISTS devolverá VERDADEIRO se a subconsulta devolve polo menos unha fila. O número de columnas devoltas pola consulta subordinada non inflúe no resultado do predicado, só o número de filas. Devolve FALSO se a subconsulta non devolve ningunha fila.

Este tipo de consultas e os exemplos veranse en detalle na actividade 5 de Consultas con subconsultas, desta mesma unidade.

2.10. Predicados compostos

Os predicados compostos son aqueles que son combinacións doutros predicados, simples ou compostos, cos operadores lóxicos AND, OR e NOT.

Cando nunha instrución hai máis dun operador lóxico, primeiro avalíase NOT, logo AND e, finalmente, OR. Se a orde de avaliación dos operadores ten que ser diferente empregaranse parénteses, xa que sempre, independentemente dos operadores, avaliaranse primeiro as instrución contidas nos parénteses.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 25 e 26, no editor de consultas do SSMS.

- **Consulta de exemplo 25:** Esta consulta devolve o identificador, nome e número de prazas das actividades impartidas polo docente con número identificador 100 ou impartidas na aula 2.

```
SELECT identificador, nome, num_prazas
FROM ACTIVIDADE
WHERE num_profesorado_imparte=100 OR
      num_aula=2;
```

Resultado da consulta do exemplo 25		
identificador	**nome**	**num_prazas**
10 20 30	TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ	15 20 10

- **Consulta de exemplo 26:** Esta consulta devolve o nome, datas de inicio e fin en formato dd/mm/aaaa, número de prazas e prezo, das actividades con máis de 10 prazas e sen observacións.

```
SELECT nome, convert(char(10),data_ini,103) as data_ini,
       convert(char(10),data_fin,103) as data_fin,
       num_prazas, prezo
FROM ACTIVIDADE
WHERE num_prazas>10 AND
      observacions IS NULL;
```

Resultado da consulta do exemplo 26				
nome	**data_ini**	**data_fin**	**num_prazas**	**prezo**
REPOSTARÍA	15/02/2015	15/03/2015	20	50.00

2.11. Consultas empregando asistentes

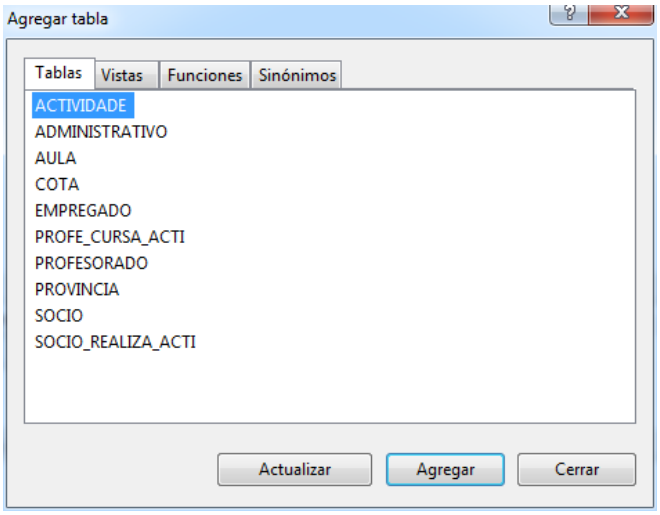
Ademais do editor de consultas, o SSMS proporciona o *deseñador de consultas* un asistente para a realización das mesmas. Para acceder a este asistente escolleremos no menú a seguinte opción:

Opción do menú	Combinación de teclas
`Consulta ----> Diseñar consulta en el editor...`	`CTRL+MAIÚSC+Q`

Para ilustrar o uso do editor deseñaremos a consulta de exemplo 5, na que se pedía o nome, prezo e número de prazas das actividades con prezo menor a 100€ ordenadas de maior a menor prezo.

Deseño da consulta de exemplo 5 co asistente

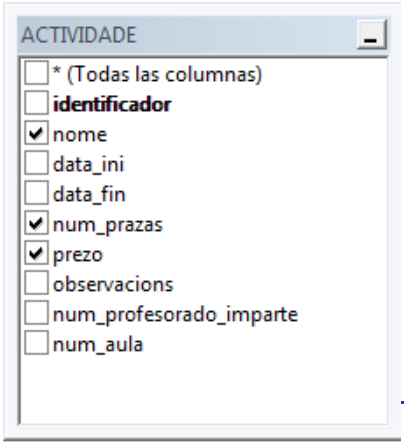
- **Paso 1. Escoller táboas:** Unha vez escollida a opción do menú aparece unha ventá onde deberase seleccionar a táboa ou táboas que se necesitan para realizar a consulta; neste caso a táboa ACTIVIDADE.



Prememos o botón Agregar e despois pechamos a ventá.

- **Paso 2. Escoller os campos:** A continuación aparece o *deseñador de consultas* coa táboa que escollemos no paso anterior. No caso de precisar máis táboas podemos facelo desde a opción *Agregar táboa* do menú contextual da ventá do deseñador (situándonos ao lado da actual táboa, nunha zona baleira da ventá e premendo o botón dereito do rato).

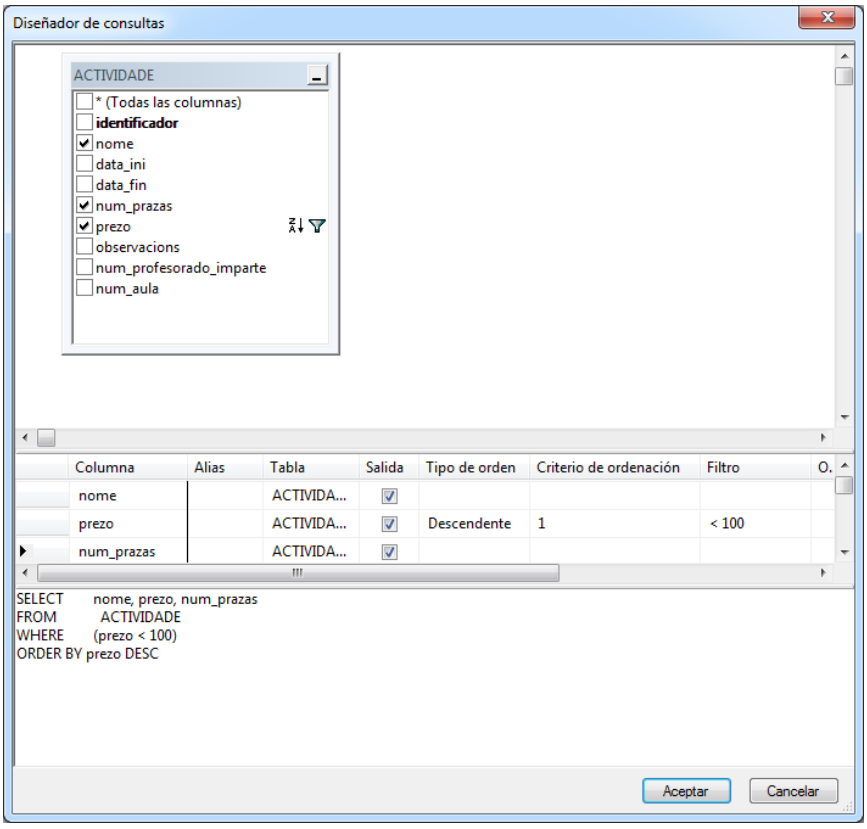
  Cada campo da táboa ACTIVIDADE ten á esquerda un cadro de selección. Marcaranse todos os campos que se vaian utilizar na consulta, en calquera cláusula. Para a nosa consulta necesitamos marcar os campos nome, data_ini, num_prazas e prezo.



Por defecto todos os campos seleccionados formarán parte do resultado. Se algún deles non queremos que apareza como columna no resultado final, deberase desmarcar da columna *Saída* da lista de columnas que aparecen debaixo da sección de táboas do deseñador.

- **Paso 3. Filtros e ordenación.** Na consulta débese amosar unicamente as actividades con prezo inferior a 100, así que na fila do atributo prezo, na columna Filtro indicaremos a condición prezo<100.

  Para ordenar o resultado, na fila do atributo de ordenación, neste caso prezo, teremos que escoller en Tipo de orden se é ascendente ou descendente. Ao ser só un campo de ordenación automaticamente en criterio de ordenación pon 1, senón nesa columna indicaramos a orde que ocupa no ORDER BY.



Ao darlle ao botón aceptar aparecerá no editor de consultas a mesma consulta en T-SQL que se amosa no deseñador. Para executala agora actuarase como con calquera outra consulta.

2.12. Tarefa de consultas simples en T-SQL

Unha vez copiados, executados e probados os exemplos das explicacións, realizarase o código T-SQL adecuado para obter a información que se pide en cada unha das consultas propostas na BD SOCIEDADE_CULTURAL ou na BD EMPRESA. Aínda que non se indique, nos resultados *todas as columnas terán un nome que identifique correctamente a información que amosan*.

- Consultas propostas na **BD SOCIEDADE_CULTURAL**.
  - **Proposta 1.** Nome e apelidos (cada un nunha columna) de todos os empregados.
  - **Proposta 2.** Número, nome e apelidos (cada un nunha columna) de todos empregados.
  - **Proposta 3.** Número, nome e apelidos (cada un nunha columna) de todos os empregados por orde alfabética de apelidos e nome.
  - **Proposta 4.** Número, nome e apelidos (cada un nunha columna) de todos os empregados por orde alfabética de apelidos e nome. Os nomes das columnas no resultado serán: *num_socio, nome_socio, apelido1, apelido2*.
  - **Proposta 5.** Número, nome completo (os 4 campos nunha única columna, de nome *socio,* co formato numero - ape1 ape2, nome) e salario de todos os empregados. No resultado deberán aparecer primeiro os que máis cobran.
  - **Proposta 6.** Número, nome completo (os 4 campos nunha única columna, de nome *socio,* co formato numero - ape1 ape2, nome) e salario de todo o profesorado. No resultado deberán aparecer primeiro os que máis cobran. O campo cargo contén PRF para o profesorado, e ADM se é un ou unha administrativo.
  - **Proposta 7.** Número identificador do profesorado que imparte clases. Como é lóxico, se un profesor imparte máis dunha actividade, o seu número só pode aparecer unha vez.
  - **Proposta 8.** Número identificador das actividades ás que asiste profesorado, é dicir, cursadas por profesorado.
  - **Proposta 9.** Nome, importe, e importe rebaixado un 20%, da actividade de nome *xadrez*.
  - **Proposta 10.** NIF, nome e apelidos dos socios dos que non temos teléfono gardado.
  - **Proposta 11.** NIF, nome, apelidos e data de nacemento dos socios nados entre 1980 e 1990, ambos incluídos.
  - **Proposta 12.** Todos os datos das actividades cuxo nome contén a letra T.
  - **Proposta 13.** Nome e importe das cotas cun custo de 30 ou 100 euros.
  - **Proposta 14.** Nome e número de prazas das actividades que non teñen nin 15 nin 20 prazas.
- Consultas propostas na **BD EMPRESA**.
  - **Proposta 15.** Nome de todos os clientes por orde alfabética.
  - **Proposta 16.** Nome das rexións nas que ten sucursais a empresa.

- **Proposta 17.** Identificador dos produtos que nos pediron nalgún momento. No resultado debe aparecer nunha soa columna o código do fabricante e o identificador do produto separados por un guión. A columna do resultado deberá chamarse *produtos*.
- **Proposta 18.** Información completa das sucursais non dirixidas polo empregado número 108.
- **Proposta 19.** Nome e límite de crédito do cliente número 1107.
- **Proposta 20.** Número e data dos pedidos feitos entre o 1 de agosto e o 31 de decembro de 2014. Só debe aparecer a data de cada pedido, sen a hora, co formato *dd-mm-aaaa*. Deben aparecer primeiro no resultado os pedidos máis recentes. Para resolver esta consulta non se poden utilizar operadores de comparación (>, <, >=, <=, < >, !=).
- **Proposta 21.** Código e nome dos fabricantes cuxo nome ten por segunda letra O.
- **Proposta 22.** Descrición e prezo dos produtos dos que non temos existencias.
- **Proposta 23.** Número identificador e nome completo dos empregados que non teñen xefe.
- **Proposta 24.** Descrición e unidades existentes, dos produtos con existencias maiores de 10 unidades e menores de 100. Para resolver esta consulta non se poden utilizar operadores de comparación (>, <, >=, <=, < >, !=).

3. Consultas resumo

As consultas resumo son denominadas así porque son un resumo das filas da táboa orixe.

Usaremos o SXBD (Sistema Xestor de Base de Datos) MS SQLServer, polo que a sintaxe que se explicará será a da linguaxe Transact-SQL (T-SQL en adiante), propia do xestor.

Explicarase a sintaxe T-SQL para a realización de consultas resumo sobre unha única táboa coas seguintes funcións de agregado:

- COUNT.
- SUM.
- AVG.
- MIN e MAX.

Tamén se explicará a orde de execución das consultas SELECT con agrupamento.

3.1. Funcións colectivas ou de agregado

Son aquelas funcións cuxo argumento é unha colección de valores tomados dos pertencentes a unha ou máis columnas. Por iso se chaman tamén funcións de columnas.

Sintaxe
`nome_función_colectiva ([DISTINCT] expresión)`

Aplícanse á colección de valores do argumento expresión e producen un único resultado a partir deles. Son as seguintes:

- AVG: Calcula a media dos valores da colección.
- COUNT: Calcula o número total de valores que hai na colección (valores non nulos).
- MAX: Devolve o valor máximo dos valores da colección.
- MIN: Devolve o valor mínimo dos valores da colección.
- SUM: Calcula a suma dos valores da colección.

Se a colección de valores está valeira, COUNT devolve 0 e as outras funcións devolven NULL.

Sintaxe COUNT
`COUNT(*)`

Esta sintaxe é só válida para a función COUNT e devolve o número de filas que hai no grupo.

Estas funcións nunca se poderán usar no WHERE, só no SELECT e na cláusula HAVING que veremos despois.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 1, 2 e 3, no editor de consultas do SSMS.

- **Consulta de exemplo 1:** Amosaremos o prezo medio das actividades da sociedade cultural. Neste caso empregamos a función de agregado AVG

```
SELECT avg(prezo) as prezo_medio
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 1
prezo_medio
107.887500

- **Consulta de exemplo 2:** Como na consulta anterior, amosaremos o prezo medio das actividades, e ademais o prezo mínimo, o máximo e a suma total dos prezos de todas as actividades.

```
SELECT avg(prezo) as prezo_medio,
       min(prezo) as prezo_minimo,
       max(prezo) as prezo_maximo,
       sum(prezo) as prezo_total
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 2			
prezo_medio	prezo_minimo	prezo_maximo	prezo_total
107.887500	0.00	301.55	431.55

- **Consulta de exemplo 3:** Nesta consulta obtemos o número de actividades que hai na BD.
  - Na primeira solución poñemos entre parénteses o nome do campo clave primaria, deste xeito asegurámonos que non ten valores nulos e que contará as filas totais da táboa.
  - Na segunda utilizamos a sintaxe que consiste en indicar un * entre os parénteses. Deste xeito indicámoslle ao xestor que conte as filas.

```
--Solución1
SELECT count(identificador) as numero_de_actividades
FROM ACTIVIDADE;
--Solución2
SELECT count(*) as numero_de_actividades
FROM ACTIVIDADE;
```

As dúas consultas devolven o mesmo resultado:

Resultado da consulta do exemplo 3
numero_de_actividades
4

3.2. Consultas con agrupamento de filas

As consultas resumo son aquelas nas que se forman grupos de filas de acordo con un criterio determinado, para poder despois aplicarlle funcións colectivas a estes grupos.

Nas consultas anteriores as funcións de agregado aplicábanse a un único grupo formado por todas as filas da táboa.

Para especificar grupos diferentes á táboa enteira terase que usar a cláusula GROUP BY.

3.2.1. Cláusula GROUP BY

GROUP BY é un cláusula opcional que serve para agrupar filas. Se existe vai sempre despois do WHERE. Se a cláusula WHERE non existe irá despois do FROM.

Na seguinte táboa amósase a sintaxe da cláusula GROUP BY:

Sintaxe T-SQL

```
SELECT ......
FROM ......
[WHERE ......]
[GROUP BY col1 [, col2, ..., colN]     ]
[ORDER BY ......];
```

Na sintaxe anterior *col1, col2, ..., colN* son nomes de atributos ou columnas chamadas *columnas de agrupamento*.

Esta cláusula indica que se agruparán as filas da táboa, de tal xeito que formarán un grupo todas as que teñan iguais valores nas columnas de agrupamento.

Hai que ter en conta que:

- Poden existir grupos dunha soa fila.
- Os valores nulos considéranse iguais, é dicir, estarán no mesmo grupo.
- Cada grupo produce unha fila no resultado.
- Todo as columnas da cláusula SELECT deben estar no GROUP BY, excepto as que están dentro de funcións colectivas. É dicir, as columnas do SELECT que non aparezan no GROUP BY deben aparecer como argumentos de funcións colectivas.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 4 e 5, no editor de consultas do SSMS.

- **Consulta de exemplo 4:** Buscamos o número de aulas que hai nos distintos estados. Nunha columna aparecerá cada estado diferente e noutra o número de aulas de cada estado.

```
SELECT estado, COUNT(*) as cantidade_aulas
FROM AULA
GROUP BY estado;
```

Resultado da consulta do exemplo 4	
estado	**cantidade_aulas**
B M R	3 1 1

- **Consulta de exemplo 5:** Buscamos o salario medio do profesorado e o dos administrativos.

```
--Solución1
SELECT avg(salario_mes) as salario_medio
FROM EMPREGADO
GROUP BY cargo;
--Solución2
SELECT cargo, avg(salario_mes) as salario_medio
FROM EMPREGADO
GROUP BY cargo;
```

Resultado da consulta do exemplo 5. Solución 1
salario_medio
600.000000 1066.966666

Nesta solución obtemos o salario medio de cada cargo, pero non podemos saber cal corresponde ao profesorado e cal aos administrativos.

Resultado da consulta do exemplo 5. Solución 2	
cargo	**salario_medio**
PRF ADM	600.000000 1066.966666

Agora, na segunda solución, ao engadir no SELECT a columna do cargo, podemos coñecer a que cargo pertence o salario medio de cada fila.

3.2.2. Cláusula HAVING

HAVING é unha cláusula opcional que serve para descartar grupos de filas, xa que serán eliminados do resultado todos aqueles grupos que non cumpran as condicións especificadas no HAVING.

Na seguinte táboa amósase a sintaxe da cláusula HAVING:

Sintaxe T-SQL

`SELECT ......`
`FROM ......`
`[WHERE ......]`

```
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

Se existe GROUP BY irá xusto despois del, senón irá despois do WHERE, ou do FROM se o WHERE non existe na consulta.

- Normalmente a cláusula HAVING acompaña ao GROUP BY. No caso de que exista o HAVING pero non GROUP BY, o HAVING actuará como o WHERE pero considerando como un único grupo todas as filas da táboa.
- Primeiro executarase o GROUP BY, é dicir, agruparanse as filas en distintos grupos. Despois o HAVING descarta aqueles grupos que non cumpran a condición ou condicións indicadas no HAVING.
- O habitual é que na condición do HAVING se indique algunha función colectiva (senón funcionaría como un WHERE e non tería sentido usalo).
- A *condición* do HAVING é un predicado, simple ou composto.
- Os atributos da condición do HAVING ou ben están no GROUP BY ou deben especificarse como argumento dunha función colectiva ou de agregado. Por exemplo, non sería correcto unha consulta como esta:

  SELECT sum(prezo)

  FROM ACTIVIDADE

  HAVING num_profesorado_imparte=100;

  xa que *num_profesorado_imparte* non está na cláusula GROUP BY, nin tampouco nunha función de agregado.
- A *condición* do HAVING pode ter sentencias subordinadas.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 6, no editor de consultas do SSMS.

- **Consulta de exemplo 6:** Esta consulta devolve o identificador das actividades realizadas por máis dun socio.

```
SELECT id_actividade
FROM SOCIO_REALIZA_ACTI
GROUP BY id_actividade
HAVING count(*)>1;
```

Resultado da consulta do exemplo 6. Con HAVING
id_actividade
10 30

Na táboa SOCIO_REALIZA_ACTI hai unha fila por matrícula de socio. Ao agrupar por *id_actividade* conseguimos que se fagan tantos grupos como actividades diferentes nas que hai socios matriculados. Se executamos a consulta sen HAVING o resultado sería o

seguinte:

Resultado da consulta do exemplo 6. Sen HAVING
id_actividade
10 30 40

A diferenza co resultado final é que non aparece a actividade 40, porque o HAVING conta as filas de cada grupo, e o número de socios matriculados na actividade 40 non é maior que 1, senón igual a 1, e polo tanto elimina do resultado ese grupo, porque non cumpre a condición do HAVING.

3.2.3. Orde de execución das sentenzas SELECT con agrupamento

As cláusulas dunha consulta SELECT, execútanse internamente no servidor, seguindo a orde que se indica na seguinte táboa.

Orde de execución	Cláusula
5º	SELECT
1º	FROM
2º	WHERE
3º	GROUP BY
4º	HAVING
6º	ORDER BY

No caso de que falte algunha cláusula a orde de execución manteríase, por exemplo, nunha consulta do tipo SELECT..... FROM .... ORDER BY...; executaríanse as cláusulas na seguinte orde:

2º SELECT..... 1º FROM .... 3º ORDER BY...;

- **5º SELECT:** Avalía as expresións para cada grupo. Se comeza por DISTINCT elimina as filas repetidas. No caso dunha consulta sen ORDER BY, esta sería a última cláusula en ser executada.
- **1º FROM:** Selecciona a táboa (ou táboas como veremos más adiante).
- **2º WHERE:** Elimina as filas que non cumpren as condicións indicadas na mesma.
- **3º GROUP BY:** Forma grupos coas filas do 2º paso que teñan valores iguais nas columnas de agrupamento indicadas no GROUP BY.
- **4º HAVING:** Descarta os grupos formados no 3º paso que non cumpran as condicións do HAVING.
- **6º ORDER BY:** Presenta a táboa resultante ordenada conforme as columnas do ORDER BY. Esta é sempre, se existe na a consulta, a última cláusula en ser executada.

3.3. Tarefa de consultas resumo en T-SQL

Unha vez copiados, executados e probados os exemplos das explicacións, realizarase o código T-SQL adecuado para obter a información que se pide en cada unha das consultas propostas na BD EMPRESA. Aínda que non se indique, nos resultados todas as columnas terán un nome que identifique correctamente a información que amosan.

- Consultas propostas na **BD EMPRESA**.
  - **Proposta 1.** Media de unidades vendidas de cada vendedor. O resultado terá dúas columnas, na primeira o número identificador do empregado (vendedor) e nunha segunda columna a media de unidades vendidas (campo *cantidade*) nos seus pedidos.
  - **Proposta 2.** Prezo máis barato de produto, prezo máis caro, prezo medio, suma total dos prezos de produto, e número de produtos distintos existentes.
  - **Proposta 3.** Número de pedidos realizados polo cliente 1103.
  - **Proposta 4.** Número de pedidos realizados por cada cliente. No resultado aparecerá o identificador do cliente e na segunda columna o número de pedidos que leva feitos cada cliente ata o de agora.
  - **Proposta 5.** Repite a consulta anterior, pero agora no resultado só poderán aparecer os clientes que fixeron máis de 2 pedidos.
  - **Proposta 6.** Repite a consulta anterior, pero agora no resultado só poderán aparecer os clientes que fixeron máis de 2 pedidos e que ademais teñen unha media de unidades mercadas (*cantidade*) inferior a 10.
  - **Proposta 7.** Cantidade total de sucursais que hai por rexión. Aparecerá o nome da rexión e na mesma columna separado por un guión, a cantidade de sucursais situadas nesa rexión.

4. Combinacións internas, externas e cruzadas

Este apartado ten como finalidade aprender a realizar consultas nas que participan máis dunha táboa.

Usaremos o SXBD (Sistema Xestor de Base de Datos) MS SQLServer, polo que a sintaxe que se explicará será a da linguaxe Transact-SQL (T-SQL en adiante), propia do xestor.

Explicarase a sintaxe T-SQL para a realización de consultas multitáboa empregando:

- combinacións (tamén chamadas *reunións* ou *joins*) internas.
- Combinacións externas.
- Combinacións cruzadas ou produtos cartesianos.

4.1. Consultas sobre varias táboas

Cando hai varias táboas involucradas nunha consulta, indicaremos os nomes das mesmas na cláusula FROM separadas por comas.

Sintaxe

```
FROM táboa1,..., táboaN
```

Ao existir varias táboas pode darse ambigüidade nos nomes dos atributos, é dicir, en dúas táboas diferentes poden existir dous campos co mesmo nome. Para evitar esta confusión empregaremos *nomes locais* ou *de correlación*. Os nomes locais consisten en indicar diante do nome do atributo o nome da táboa seguido dun punto.

Sintaxe de nomes locais

```
SELECT táboaExemplo1.nome, táboaExemplo2.nome
FROM táboaExemplo1, táboaExemplo2;
--Se non indicamos o nomes das táboas diante o servidor daría un erro indicando
--que o nome de columna 'nome' é ambiguo.
```

Alias das táboas

Igual que vimos cos alias dos atributos, tamén podemos dar alias ás táboas. Neste caso porase o nome abreviado despois do nome da táboa deixando un espazo en branco. tamén se podería poñer a palabra *as* como facemos cos atributos.

Úsanse para abreviar nomes de táboas moi longos e tamén para facer as consultas máis lexibles. Tomando o exemplo anterior quedaría como segue:

Sintaxe de nomes locais con alias

```
SELECT t1.nome, t2.nome
FROM táboaExemplo1 [as] t1, táboaExemplo2 [as] t1;
--Se non indicamos o nomes das táboas diante o servidor daría un erro indicando
--que o nome de columna 'nome' é ambiguo.
--Poderíase indicar tamén as palabra as antes do alias.
```

4.2. Combinacións internas

En álxebra relacional defínese a operación de *join natural* que nos permite combinar datos dunhas táboas coas outras por unha condición de igualdade.

Na linguaxe SQL o *join natural* denomínase combinación, reunión ou *join interno* ou tamén *inner join*.

Necesitamos empregar combinacións internas cando na consulta facemos referencia a datos que están en máis dunha táboa.

Nas BBDD relacionais as táboas relaciónanse a través das claves primarias e foráneas. Nas combinacións deben existir unha ou máis condicións de combinacións, que son condicións de igualdade entre os atributos que relacionan.

Sintaxe1 con dúas táboas

```
SELECT columnas das táboas do FROM
FROM táboa1, táboa2
WHERE táboa1.columnaA = táboa2.columnaB
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

Sintaxe2 con tres táboas

```
SELECT columnas das táboas do FROM
FROM táboa1, táboa2, táboa3
WHERE táboa1.columnaA = táboa2.columnaB AND
      táboa2.columnaC = táboa3.columnaD
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

Sintaxe3 con INNER JOIN con dúas táboas

```
SELECT columnas das táboas do FROM
FROM táboa1 INNER JOIN táboa2
     ON táboa1.columnaA = táboa2.columnaB
[WHERE predicado/s]
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

Sintaxe4 con INNER JOIN tres táboas

```
SELECT columnas das táboas do FROM
FROM (táboa1 INNER JOIN táboa2 ON táboa1.columnaA = táboa2.columnaB)
      INNER JOIN táboa3 ON táboa2.columnaC = táboa3.columnaD
[WHERE predicado/s]
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

No exemplo da imaxe seguinte vemos como no resultado só aparecen as filas das dúas táboas que teñen o número de aula igual. A aula 5 que non ten actividade asociada porque non aparece na columna actividade.num_aula, non aparece no resultado. Do mesmo xeito a actividade 'PROBA PARA FULL JOIN' que ten como num_aula NULL, tampouco aparecerá no resultado.

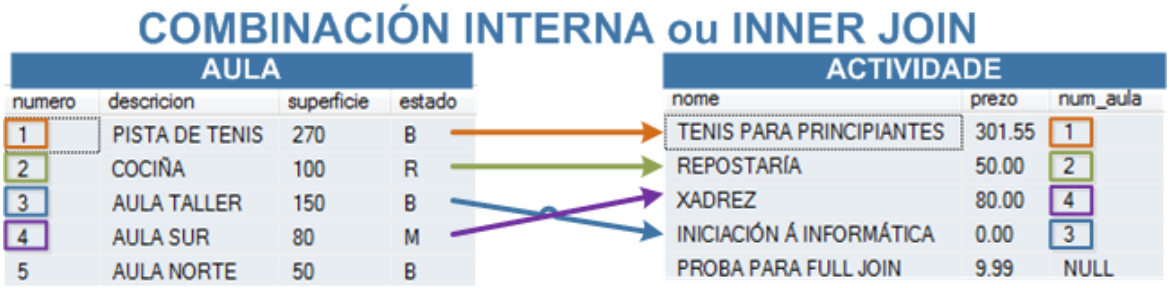
Para poder facer estes exemplos fixemos a inserción da actividade PROBA PARA FULL JOIN con num_aula a NULL. E tamén houbo que facer unha modificación no campo actividade.num_aula para que admitise nulos.

```
/****CAMBIOS PREVIOS exemplos JOINS internos e externos****/
--Cambio da columa num_aula
--da táboa ACTIVIDADE a opcional
ALTER TABLE ACTIVIDADE
ALTER COLUMN num_aula int null;

--Inserción dunha actividade sen aula asignada
INSERT INTO ACTIVIDADE (identificador, nome, data_ini, data_fin,
                        num_prazas, prezo, observacions,
                        num_profesorado_imparte, num_aula)
VALUES (99, 'PROBA PARA FULL JOIN', '12-02-2015 16:30','12-02-2015 17:30',
                        99, 9.990, 'Fila de proba para full join',
                        300, null);

--Ao rematar as probas dos exemplos habería que lanzar
--as seguintes instrucións
--Cambio da columa num_aula
--da táboa ACTIVIDADE a opcional
ALTER TABLE ACTIVIDADE
ALTER COLUMN num_aula int not null;

--Borrado da actividade sen aula asignada
DELETE FROM ACTIVIDADE
WHERE num_aula IS NULL;
```




```
--INNER JOIN
SELECT au.numero, au.descricion, au.superficie, au.estado,
       ac.nome, ac.prezo, ac.num_aula
FROM AULA au INNER JOIN ACTIVIDADE ac
     ON au.numero=ac.num_aula;
```

RESULTADO

numero	descricion	superficie	estado	nome	prezo	num_aula
1	PISTA DE TENIS	270	B	TENIS PARA PRINCIPIANTES	301.55	1
2	COCIÑA	100	R	REPOSTARÍA	50.00	2
4	AULA SUR	80	M	XADREZ	80.00	4
3	AULA TALLER	150	B	INICIACIÓN Á INFORMÁTICA	0.00	3

Normas para a realización de combinacións internas

As normas que se presentan de seguido, son válidas tamén para as combinacións externas.

- Pódense unir tantas táboas como desexemos.
- Na cláusula SELECT pódense citar columnas de calquera táboa do FROM.
- Se hai columnas co mesmo nome nas distintas táboas do FROM, deben identificarse usando a cualificación de nomes.
- Aínda que os nomes dos atributos non se repitan en distintas táboas, antepoñer o nome ou alias da táboa a cada atributo, favorecerá que as consultas se entendan mellor.
- Cando combinamos N táboas no FROM, terá que haber polo menos N-1 condicións de *join* ou de igualdade.
- A primeira cláusula en executarse nunha consulta é o FROM, e cando hai máis dunha táboa, internamente, o que está a facer o servidor é un produto cartesiano entre as táboas do FROM. Son as condicións de combinación as que reducen o número de filas devoltas xa que de todas as do produto cartesiano só aparecerán no resultado as que cumpran a ou as condicións de igualdade ou *join*.
- Na sintaxe na que as condicións de combinación van no WHERE, hai que ter en conta que primeiro debemos poñer todas aquelas condicións que non sexan de combinación. Deste xeito o produto cartesiano previo ao *join* terá menos filas que combinar.

- Hai que especificar as condicións de *join* en base a unha clave primaria e a unha clave foránea.
- Se unha táboa ten unha clave primaria composta, débese facer referencia á clave enteira na condición de combinación.
- Os atributos da igualdade da condición de combinación deben ser do mesmo tipo.
- É importante limitar no posible o número de táboas nun *join*, porque a máis táboas, máis tempo vai tomar o xestor para resolver a consulta.
- No FROM podemos usar varias veces a mesma táboa, o que se coñece como *autocombinación*, e neste caso si é obrigatorio o uso de alias das táboas para evitar ambigüidades.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 1, 2, 3 e 4, no editor de consultas do SSMS.

- **Consulta de exemplo 1:** Amosaremos o nome e apelidos daqueles socios que deben algunha actividade.

  Na primeira solución poñemos o nome completo diante dos campos, aínda que neste caso non se repiten. Deste xeito conséguese que calquera que vexa a consulta, incluso nós pasado un tempo sen vela, poidamos saber de que táboas proceden os atributos sen consultar a estrutura das mesmas.

  Na segunda solución empregamos alias das táboas, que aínda facilita máis o entendemento da consulta.

  Observa como no WHERE aparece primeiro a condición que non é de combinación.

  Utilizamos DISTINCT porque o mesmo socio pode deber máis dunha actividade, e polo tanto aparecería máis dunha vez.

```
--Solución1
SELECT DISTINCT SOCIO.ape1, SOCIO.ape2, SOCIO.nome
FROM SOCIO, SOCIO_REALIZA_ACTI
WHERE  SOCIO_REALIZA_ACTI.pagada ='N' AND
       SOCIO.numero = SOCIO_REALIZA_ACTI.num_socio
ORDER BY SOCIO.ape1, SOCIO.ape2, SOCIO.nome;
--Solución2 con alias das táboas
SELECT DISTINCT s.ape1, s.ape2, s.nome
FROM SOCIO s, SOCIO_REALIZA_ACTI sra
WHERE  sra.pagada ='N' AND
       s.numero = sra.num_socio
ORDER BY s.ape1, s.ape2, s.nome;
```

O resultado é o mesmo para as dúas solucións.

Resultado da consulta do exemplo 1		
ape1	**ape2**	**nome**
DEL CARMEN SIEIRO	LÉREZ CAMPOS	JORGE MANUEL

- **Consulta de exemplo 2:** Nome propio do/a profesor/a que imparte a actividade número 20.

```
--Solución1
SELECT e.nome
FROM EMPREGADO e, ACTIVIDADE a
WHERE a.identificador=20 AND
      e.numero=a.num_profesorado_imparte;
--Solución2
SELECT e.nome
FROM EMPREGADO e INNER JOIN ACTIVIDADE a
     ON e.numero=a.num_profesorado_imparte
WHERE a.identificador=20;
```

Resultado da consulta do exemplo 2
nome
CARLOS

- **Consulta de exemplo 3:** Nesta consulta obtemos o nome propio dos socios que realizan a actividade de nome 'TENIS PARA PRINCIPIANTES'.
  - Na segunda solución usamos a sintaxe de INNER JOIN, e empregamos parénteses ao ser máis de 2 táboas.

```
--Solución1
SELECT s.nome
FROM SOCIO AS s, SOCIO_REALIZA_ACTI  r, ACTIVIDADE  a
WHERE a.nome= 'TENIS PARA PRINCIPIANTES' AND
      s.numero=r.num_socio AND
      r.id_actividade=a.identificador;
--Solución2
SELECT s.nome
FROM (SOCIO AS s INNER JOIN SOCIO_REALIZA_ACTI  r
      ON s.numero=r.num_socio)
     INNER JOIN ACTIVIDADE  a
     ON r.id_actividade=a.identificador
WHERE a.nome= 'TENIS PARA PRINCIPIANTES';
```

As dúas consultas, que se diferencian na sintaxe da combinación interna, devolven o mesmo resultado:

Resultado da consulta do exemplo 3
nome
MARÍA MANUEL

- **Consulta de exemplo 4:** Buscamos o nome propio dos empregados que cursan actividades. Como segunda columna aparecerá o nome da actividade cursada por cada un deles.

```
--Solución1
SELECT e.nome as socio, a.nome as actividade
FROM EMPREGADO e, PROFE_CURSA_ACTI c, ACTIVIDADE a
WHERE e.numero=c.num_profesorado AND
      c.id_actividade=a.identificador;
--Solución2
SELECT e.nome as socio, a.nome as actividade
FROM (EMPREGADO e INNER JOIN PROFE_CURSA_ACTI c
      ON e.numero=c.num_profesorado) INNER JOIN ACTIVIDADE a
      ON c.id_actividade=a.identificador;
```

As dúas consultas devolven o mesmo resultado:

Resultado da consulta do exemplo 4	
socio	**actividade**
MARÍA CARLOS	INICIACIÓN Á INFORMÁTICA XADREZ

4.3. Combinacións externas

Ao contrario que as combinacións internas, as externas teñen en conta as filas desparelladas. Nun *join* interno buscamos a información que teñen en común dúas táboas. Nos *joins* externos no resultado aparecen tamén filas que non teñen a súa parella na outra.

Os tipos de combinacións externas ou *joins* externos son:

- LEFT JOIN.
- RIGHT JOIN.
- FULL JOIN.
-

A sintaxe é a mesma que a que xa se veu para o INNER JOIN, só que agora cambiamos INNER por LEFT OUTER, RIGHT OUTER ou FULL OUTER segundo sexa o caso:

Sintaxe T-SQL

```
SELECT columnas das táboas do FROM
FROM táboa_esquerda {RIGHT|LEFT|FULL} [OUTER] JOIN táboa_dereita
     ON táboa_esquerda.columnaA opComparación táboa_dereita.columnaB
[WHERE predicado/s]
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

opComparación será un dos seguintes operadores =, <, >, <=, >=, <>, !=

4.3.1. LEFT JOIN

Participan todas as filas da táboa esquerda do *join*. Permítenos obter todos os rexistros da táboa que se referencia á esquerda nunha consulta e completa as filas con NULL no caso de que nos exista un valor igual almacenado na táboa da dereita.

Se nos fixamos na consulta da imaxe seguinte, vemos como aparece a aula 5 aínda que non existen actividades impartidas nesa aula, porque a táboa do lado esquerdo do *join* é AULA, e polo tanto deben aparecer todas as aulas aínda que non teñan a súa correspondencia. Isto fai que os campos do resultado para ese valor de aula 5 teñan que estar a NULL.

Da táboa da dereita, ACTIVIDADE, só aparecen no resultado, aquelas que teñen num_aula, por iso no resultado non vai aparecer a actividade 'PROBA PARA FULL JOIN' xa que en AULA non pode existir ningún número de aula igual a NULL, entre outras cousas porque AULA.numero é clave primaria.


```
--LEFT JOIN
SELECT au.numero, au.descricion, au.superficie, au.estado,
       ac.nome, ac.prezo, ac.num_aula
FROM AULA au LEFT JOIN ACTIVIDADE ac
     ON au.numero=ac.num_aula;
```

RESULTADO

numero	descricion	superficie	estado	nome	prezo	num_aula
1	PISTA DE TENIS	270	B	TENIS PARA PRINCIPIANTES	301.55	1
2	COCIÑA	100	R	REPOSTARÍA	50.00	2
3	AULA TALLER	150	B	INICIACIÓN Á INFORMÁTICA	0.00	3
4	AULA SUR	80	M	XADREZ	80.00	4
5	AULA NORTE	50	B	NULL	NULL	NULL

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 5, no editor de consultas do SSMS.

- **Consulta de exemplo 5:** Buscamos o nif, o nome completo e o cargo de todos os empregados, cos identificadores das actividades que cursan. No caso de que un empregado curse máis dunha actividade aparecerá no resultado tantas veces como actividades curse. Aínda que un empregado non curse ningunha actividade tamén debe aparecer. Debe ser indiferente que o empregado sexa profesorado ou administrativo.

```
SELECT e.nif, e.nome, e.ape1, e.ape2, c.id_actividade
FROM EMPREGADO e LEFT JOIN PROFE_CURSA_ACTI c
      ON e.numero=c.num_profesorado;
```

A consulta devolve os 4 empregados da BD, pero na columna id_actividade nos empregados que non cursan ningunha actividade, debe poñer NULL:

Resultado da consulta do exemplo 5				
nif	**nome**	**ape1**	**ape2**	**id_actividade**
11111111A	MARÍA	GARCÍA	PÉREZ	40
22222222B	CARLOS	REGO	PENA	30
33333333C	JUANA	POSE	VARELA	NULL
44444444D	JOSÉ	GONZÁLEZ	ÍNSUA	NULL

4.3.2. RIGHT JOIN

É equivalente ao LEFT pero neste caso no resultado aparecen todas as filas da táboa da dereita.

No exemplo seguinte a táboa da que deben saír todas as filas no resultado, teñan ou non teñan correspondencia na da esquerda, é ACTIVIDADE, a da dereita. Os campos de AULA aparecerán con valor NULL, e a aula 5 non aparece.


```
--RIGHT JOIN
SELECT au.numero, au.descricion, au.superficie, au.estado,
       ac.nome, ac.prezo, ac.num_aula
FROM AULA au RIGHT JOIN ACTIVIDADE ac
     ON au.numero=ac.num_aula;
```

RESULTADO

numero	descricion	superficie	estado	nome	prezo	num_aula
1	PISTA DE TENIS	270	B	TENIS PARA PRINCIPIANTES	301.55	1
2	COCIÑA	100	R	REPOSTARÍA	50.00	2
4	AULA SUR	80	M	XADREZ	80.00	4
3	AULA TALLER	150	B	INICIACIÓN Á INFORMÁTICA	0.00	3
NULL	NULL	NULL	NULL	PROBA PARA FULL JOIN	9.99	NULL

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 6, no editor de consultas do SSMS.

- **Consulta de exemplo 6:** O RIGHT JOIN actúa como o LEFT JOIN, por iso esta consulta é a mesa que a do exemplo 5 coa orde das táboas do FROM intercambiada, e substituíndo LEFT por RIGHT.

```
SELECT e.nif, e.nome, e.ape1, e.ape2, c.id_actividade
FROM PROFE_CURSA_ACTI c RIGHT JOIN EMPREGADO e
       ON e.numero=c.num_profesorado;
```

A consulta devolve o mesmo que a consulta de exemplo 5.

4.3.3. FULL JOIN

No resultado participan todas as filas das táboas da combinación ou *join*, dereita e esquerda.

Na imaxe seguinte vemos como aparecen todas as filas de AULA e todas as de ACTIVIDADE


```
--FULL JOIN
SELECT au.numero, au.descricion, au.superficie, au.estado,
       ac.nome, ac.prezo, ac.num_aula
FROM ACTIVIDADE ac FULL JOIN AULA au
     ON au.numero=ac.num_aula;
```

RESULTADO

numero	descricion	superficie	estado	nome	prezo	num_aula
1	PISTA DE TENIS	270	B	TENIS PARA PRINCIPIANTES	301.55	1
2	COCIÑA	100	R	REPOSTARÍA	50.00	2
4	AULA SUR	80	M	XADREZ	80.00	4
3	AULA TALLER	150	B	INICIACIÓN Á INFORMÁTICA	0.00	3
NULL	NULL	NULL	NULL	PROBA PARA FULL JOIN	9.99	NULL
5	AULA NORTE	50	B	NULL	NULL	NULL

4.3.4. Evitar valores NULL nos resultados

Nas consultas con combinacións externas nos resultados aparece NULL naquelas columnas con información descoñecida. Un usuario básico non ten porque saber o significado dese NULL e pode interpretar que houbo un erro na carga dos datos.

Isto non só ocorre nas consultas cos *joins*, só hai que pensar nos segundos apelidos, que son opcionais. Se alguén nos pide unha listaxe para publicar na que deben aparecer todos os socios, cos seus nomes completos e hai algún que non ten segundo apelido, non ten moito sentido que na listaxe apareza un NULL, en todo caso debería aparecer en branco.

Para evitar estes problemas usaremos a función ISNULL.

Función ISNULL

Esta función, substitúe os valores NULL que atopa na columna ou expresión que se lle pasa, polo valor de substitución que se especifique na mesma.

Sintaxe T-SQL
`ISNULL(columna_ou_expresión_con_null, valor_de_substitución)`

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 7 e 8, no editor de consultas do SSMS.

- **Consulta de exemplo 7:** Imos coller a consulta do exemplo 6 pero agora no caso de que un empregado non curse actividades debe aparecer o número 0 na columna da actividade.

```
SELECT e.nif, e.nome, e.ape1, e.ape2,
       ISNULL(c.id_actividade, 0) as activ_cursada
FROM PROFE_CURSA_ACTI c RIGHT JOIN EMPREGADO e
      ON e.numero=c.num_profesorado;
```

A consulta devolve o mesmo que a consulta 6, pero na columna activ_cursada nos empregados que non cursan ningunha actividade aparece un 0:

Resultado da consulta do exemplo 7				
nif	**nome**	**ape1**	**ape2**	**activ_cursada**
11111111A 22222222B 33333333C 44444444D	MARÍA CARLOS JUANA JOSÉ	GARCÍA REGO POSE GONZÁLEZ	PÉREZ PENA VARELA ÍNSUA	40 30 0 0

- **Consulta de exemplo 8:** Imos coller a consulta do exemplo 6 pero agora no caso de que un empregado non curse actividades debe aparecer a frase 'Sen actividade.' na columna da actividade.

  Se nos limitamos a facer o mesmo que na solución anterior, é dicir, indicar o valor de substitución, neste caso a frase entre comiñas simples 'Sen actividade.', o servidor devolverá o seguinte erro:

```
Error de conversión al convertir el valor varchar 'Sen actividade.' al tipo de datos int.
```

  Isto ocorre porque a columna id_actividade que estamos intentando amosar no resultado, é de tipo int, e queremos engadir nesa columna valores alfanuméricos, unha frase. pero nunha mesma columna non poden existir datos de diferentes tipos.

  Para solucionalo usaremos a función *cast* (tamén poderiamos ter empregado *convert*). Temos que cambiarlle o tipo á columna id_actividade para que ao calcular o resultado o servidor amose os identificadores de actividades como caracteres e non como números. Para o servidor o 40 será '40', e o 30 será '30'. A efectos prácticos non nos inflúe porque non imos facer operacións con esa información.

```
--Solución1 CON ERRO
SELECT e.nif, e.nome, e.ape1, e.ape2,
       ISNULL(c.id_actividade, 'Sen actividade.') as activ_cursada
FROM PROFE_CURSA_ACTI c RIGHT JOIN EMPREGADO e
      ON e.numero=c.num_profesorado;
--Solución2 CORRECTA
SELECT e.nif, e.nome, e.ape1, e.ape2,
       ISNULL(cast(c.id_actividade as varchar(15)), 'Sen actividade.')
       as activ_cursada
FROM PROFE_CURSA_ACTI c RIGHT JOIN EMPREGADO e
      ON e.numero=c.num_profesorado;
```

A consulta devolve o mesmo que a consulta 6, pero na columna id_actividade nos empregados que non cursan ningunha actividade aparece un 0:

Resultado da consulta do exemplo 8				
nif	**nome**	**ape1**	**ape2**	**activ_cursada**
11111111A	MARÍA	GARCÍA	PÉREZ	40
22222222B	CARLOS	REGO	PENA	30
33333333C	JUANA	POSE	VARELA	Sen actividade.
44444444D	JOSÉ	GONZÁLEZ	ÍNSUA	Sen actividade.

4.4. Combinacións cruzadas ou produtos cartesianos

As combinacións cruzadas devolven todas as combinacións posibles das filas de cada táboa especificada. Non fai falla unha fila común. Devolve un produto cartesiano no que os elementos dese produto son as filas.

Cando facemos unha consulta cun *join* interno empregando a sintaxe sen *inner join*, se nos esquecemos de indicar no WHERE a condición de combinación, o que estamos a facer é un produto cartesiano.

O resultado terá un número de filas igual ao resultado de multiplicar o número de filas das dúas táboas: *num_filas_taboa1 x num_filas_taboa2= num_filas_produto_cartesiano.*

É importante indicar que non é habitual usalo en BBDD relacionais.

Sintaxe T-SQL

```
SELECT columnas das táboas do FROM
FROM táboa1 CROSS JOIN táboa2
[WHERE predicado/s]
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 9, no editor de consultas do SSMS.

- **Consulta de exemplo 9:** Produto cartesiano entre os socios e as cotas. É dicir, imos combinar cada fila da táboa SOCIO con cada fila de COTA.

  Farémolo de dous xeitos, cun *join* interno sen condición de combinación, e coa sintaxe CROSS JOIN.

```
SELECT s.nif, s.nome, s.ape1, s.ape2, s.cod_cota,
       c.codigo, c.nome, c.importe
FROM SOCIO s, COTA c;
--Solución2 CORRECTA
SELECT s.nif, s.nome, s.ape1, s.ape2, s.cod_cota,
       c.codigo, c.nome, c.importe
FROM SOCIO s CROSS JOIN COTA c;
```

  A consulta devolve o mesmo nos dous casos. Vemos como se repiten as 4 filas dos socios para cada cota. No resultado hai 16 filas que é o resultado de multiplicar 4x4, 4 filas de SOCIO por 4 filas de COTA.

Resultado da consulta do exemplo 9

nif	nome	ape1	ape2	cod_cota	codigo	nome	importe
11111112A	MARÍA	GRAÑA	UMIA	11	11	DE HONRA	100.00
22222223B	MANUEL	SIEIRO	CAMPOS	12	11	DE HONRA	100.00
33333334C	JORGE	DEL CARMEN	LÉREZ	99	11	DE HONRA	100.00
44444445D	CARLA	VIEITO	GIL	11	11	DE HONRA	100.00
11111112A	MARÍA	GRAÑA	UMIA	11	12	FAMILIAR	30.00
22222223B	MANUEL	SIEIRO	CAMPOS	12	12	FAMILIAR	30.00
33333334C	JORGE	DEL CARMEN	LÉREZ	99	12	FAMILIAR	30.00
44444445D	CARLA	VIEITO	GIL	11	12	FAMILIAR	30.00
11111112A	MARÍA	GRAÑA	UMIA	11	13	HABITUAL	50.00
22222223B	MANUEL	SIEIRO	CAMPOS	12	13	HABITUAL	50.00
33333334C	JORGE	DEL CARMEN	LÉREZ	99	13	HABITUAL	50.00
44444445D	CARLA	VIEITO	GIL	11	13	HABITUAL	50.00
11111112A	MARÍA	GRAÑA	UMIA	11	99	GRATUITA	0.00
22222223B	MANUEL	SIEIRO	CAMPOS	12	99	GRATUITA	0.00
33333334C	JORGE	DEL CARMEN	LÉREZ	99	99	GRATUITA	0.00
44444445D	CARLA	VIEITO	GIL	11	99	GRATUITA	0.00

4.5. Autocombinación ou self join

A autocombinación consiste en facer *join* dunha táboa consigo mesma. Neste caso o que estamos a facer é combinar filas dunha táboa con filas da mesma táboa. Úsanse cando no *join* interveñen interrelacións reflexivas, como o exemplo típico de *ser_xefe_de.*

A autocombinación pode ser de calquera tipo, é dicir interna ou externa, INNER, LEFT, RIGHT, FULL ou CROSS. O único que cambia e que agora as táboas a combinar son a mesma, e polo tanto é imprescindible empregar alias. Internamente estariamos a traballar con dúas táboas diferentes con estrutura e contido coincidentes. O xestor executa a consulta empregando os alias.

Sintaxe T-SQL autocombinacións internas

```
--Sintaxe1
SELECT columnas das táboas do FROM
FROM táboa1 t1, táboa1 t2
WHERE t1.columnaA = t2.columnaB
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
--Sintaxe2 con INNER JOIN
SELECT columnas das táboas do FROM
FROM táboa1 t1 INNER JOIN táboa1 t2
     ON t1.columnaA = t2.columnaB
[WHERE predicado/s]
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

Sintaxe T-SQL autocombinacións externas

```
SELECT columnas das táboas do FROM
FROM táboa1 t1 {RIGHT|LEFT|FULL} [OUTER] JOIN táboa1 t2
     ON t1.columnaA opComparación t2.columnaB
[WHERE predicado/s]
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

Sintaxe T-SQL autocombinacións en produtos cartesianos

```
SELECT columnas das táboas do FROM
FROM táboa1 t1 CROSS JOIN táboa1 t2
[WHERE predicado/s]
[GROUP BY col1 [, col2, ..., colN]     ]
[HAVING condición/s]
[ORDER BY ......];
```

4.6. Tarefa de consultas con combinacións internas, externas e cruzadas en T-SQL

Unha vez copiados, executados e probados os exemplos das explicacións, realizarase o código T-SQL adecuado para obter a información que se pide en cada unha das consultas propostas na BD EMPRESA. Aínda que non se indique, nos resultados todas as columnas terán un nome que identifique correctamente a información que amosan.

- Consultas propostas na **BD EMPRESA**.
  - **Proposta 1.** Nome de todos os fabricantes dos que se fixeron pedidos. Debes propoñer dúas solucións, unha coa sintaxe coa condición de combinación no WHERE, e outra coa sintaxe coa condición de combinación no FROM.
  - **Proposta 2.** Nome de todos os fabricantes, fixéranse ou non pedidos. Se tiveron pedidos aparecerá o nome e nunha segunda columna o número de pedido. Se dun fabricante se fixeron máis dun pedido, aparecerá tantas veces como pedidos se lle fixeron. No caso de non ter pedido, como número de pedido deberá aparecer o valor 99.
  - **Proposta 3.** Nome de todos os fabricantes, fixéranse ou non pedidos. Se tiveron pedidos aparecerá o nome e nunha segunda columna o número de pedido. Se dun fabricante se fixeron máis dun pedido, aparecerá tantas veces como pedidos se lle fixeron. No caso de non ter pedido, como número de pedido deberá aparecer a frase 'Sen pedidos.'.
  - **Proposta 4.** Código dos produtos (co formato *cod_fabricante-id_produto*) e descrición, dos produtos que non foron pedidos nunca.
  - **Proposta 5.** Produto cartesiano entre a táboa de sucursais e a de empregados. Nunha primeira columna aparecerá a cidade da sucursal e na segunda o nome completo do empregado (co formato *nome ape1 ape2*). Débense propoñer dúas solucións, segundo a sintaxe empregada para o produto cartesiano.
  - **Proposta 6.** Número e nome completo (co formato *nome ape1 ape2*) de todos os empregados, así como a cidade da sucursal que dirixen, se é que dirixen algunha. Na terceira columna, de nome sucursal_que_dirixe, nas filas dos empregados que non son directores de sucursais, deberá aparecer a frase 'Non é director.'.
  - **Proposta 7.** Número e nome completo dos empregados que teñen xefe, co número e o nome completo do seu xefe nunha segunda columna. (Revisa o concepto
  - Autocombinación ou self join). Nas columnas aparecerán o número separado do nome completo por un guión.
  - **Proposta 8.** Número e nome completo de todos os empregados, co número e o nome completo do seu xefe nunha segunda columna. Nas columnas aparecerán o número separado do nome completo por un guión. Se algún empregado non tivese xefe, na segunda columna debe aparecer a frase 'Xefe por designar.'.

- **Proposta 9.** Nome completo de todos os empregados co nome do cliente que teñen asignado. No caso de que non tivesen ningún cliente aparecerá no nome do cliente a frase 'Sen cliente.'. Do mesmo xeito se un cliente non ten empregado asignado, na columna do empregado aparecerá 'Sen vendedor.'. É importante que aparezan todos os empregados, teñan ou non clientes e todos os clientes teñan ou non empregados.
- **Proposta 10.** Escolle unha das túas solucións das consultas propostas nas que empregaches un LEFT JOIN, e modifícaa usando RIGHT JOIN.

5. Consultas con subconsultas

Este apartado ten como fin aprender a realizar consultas que inclúen outras consultas nas cláusulas WHERE e HAVING.

Usaremos o SXBD (Sistema Xestor de Base de Datos) MS SQLServer, polo que a sintaxe que se explicará será a da linguaxe Transact-SQL (T-SQL en adiante), propia do xestor.

Explicarase a sintaxe T-SQL para a realización de consultas con subconsultas que inclúen:

- Predicado IN.
- Predicado EXISTS.
- Predicados cuantificados (ALL, SOME, ANY).
- Sentencias encadeadas.
- Consultas correlacionadas.

5.1. Subconsultas ou consultas subordinadas

Cando nunha consulta necesitamos empregar outra consulta SELECT na cláusula WHERE ou HAVING, esta denomínase consulta subordinada ou subconsulta.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 1, no editor de consultas do SSMS.

- **Consulta de exemplo 1:** Nomes completos dos empregados cuxo salario supere en máis do 50% ao do empregado con NIF '22222222B'. Para resolvela non se poden utilizar joins.

```
SELECT nome, ape1, ape2
FROM EMPREGADO
WHERE salario_mes > (SELECT salario_mes * 1.5
                     FROM EMPREGADO
                     WHERE nif = '22222222B');
```

Resultado da consulta do exemplo 1		
nome	**ape1**	**ape2**
JUANA	POSE	VARELA

5.2. Predicado IN

Na actividade A02 de consultas simples xa vimos o predicado IN e o seu uso. Lembremos as dúas posibles sintaxes:

Sintaxe1 IN
`expresión [NOT] IN (valor1 [,valor2,...,valorN])`
Sintaxe2 IN
`expresión [NOT] IN (consulta subordinada)`

Na segunda sintaxe, o predicado IN devolverá verdadeiro se a expresión ten como valor un dos devoltos pola consulta subordinada escrita dentro dos parénteses. É importante que a consulta subordinada devolva só unha columna.

Se indicamos NOT IN devolve VERDADEIRO se a expresión ten como valor un distinto dos devoltos pola consulta subordinada.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 2 e 3, no editor de consultas do SSMS.

- **Consulta de exemplo 2:** Código das cotas que están asignadas a un socio polo menos e cuxo prezo está entre 50 e 100€. Para resolvela non se poden utilizar joins.

```
SELECT DISTINCT cod_cota
FROM SOCIO
WHERE  cod_cota IN (SELECT codigo
                    FROM COTA
                    WHERE importe BETWEEN 50 and 100);
```

Resultado da consulta do exemplo 2
cod_cota
11

- **Consulta de exemplo 3:** NIF e nome completo dos empregados que non asisten a actividades. Para resolvela non se poden utilizar joins.

```
/*** CONSULTA DE EXEMPLO 3 ***/
--NIF e nome completo dos empregados
--que non asisten a actividades.
SELECT NIF, nome, ape1, ape2
FROM EMPREGADO
WHERE  numero NOT IN (SELECT num_profesorado
                      FROM PROFE_CURSA_ACTI);
```

Resultado da consulta do exemplo 3			
NIF	**nome**	**ape1**	**ape2**
33333333C 44444444D	JUANA JOSÉ	POSE GONZÁLEZ	VARELA ÍNSUA

5.3. Predicado EXISTS

Sintaxe EXISTS
`expresión [NOT] EXISTS (consulta subordinada)`

Este predicado devolverá VERDADEIRO se o resultado da subconsulta devolve unha ou máis filas. Devolverá FALSO se o resultado da consulta subordinada é unha táboa valeira.

Cando usamos o predicado EXISTS na cláusula WHERE debemos ter en conta o seguinte:

- A consulta subordinada pode ter un número calquera de columnas.
- Este tipo de predicados non poden tomar un valor descoñecido.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 4 e 5, no editor de consultas do SSMS.

- **Consulta de exemplo 4:** Datos completos das aulas se existe algunha en bo estado (estado vale B). Para resolvela non se poden utilizar joins.

  Na primeira solución na consulta subordinada devolvemos todos os campos da táboa, pero o resultado é o mesmo na segunda solución na que o SELECT devolve a clave primaria. É importante que se a subconsulta só devolve un campo, este debe ser obrigatorio, normalmente usaremos un campo que forme parte dunha clave primaria.

```
--Solución1
SELECT *
FROM AULA
WHERE  EXISTS (SELECT *
               FROM AULA
               WHERE estado='B');
--Solución2
SELECT *
FROM AULA
WHERE  EXISTS (SELECT numero
               FROM AULA
               WHERE estado='B');
```

O resultado é o mesmo para as dúas solucións:

Resultado da consulta do exemplo 4			
NIF	**descricion**	**superficie**	**estado**
1	PISTA DE TENIS	270	B
2	COCIÑA	100	R
3	AULA TALLER	150	B
4	AULA SUR	80	M
5	AULA NORTE	50	B

- **Consulta de exemplo 5:** Datos completos das aulas se non existen aulas en bo estado (estado vale B). Para resolvela non se poden utilizar joins. O resultado non devolverá nada, xa que a subconsulta devolve algunha fila e estamos usando NOT EXISTS, é dicir, se non devolve nada.

```
SELECT *
FROM AULA
WHERE  NOT EXISTS (SELECT numero
                   FROM AULA
                   WHERE estado='B');
```

A consulta non devolve ningún resultado.

Resultado da consulta do exemplo 5			
NIF	**descricion**	**superficie**	**estado**

5.4. Predicados cuantificados

Cando usamos unha sentenza SELECT subordinada no WHERE ou no HAVING e hai un predicado de comparación, o resultado debe ser un valor único. Pero admitiremos que teña varios valores se a subconsulta vai precedida das *palabras cuantificadoras*: ANY, SOME ou ALL.

Os predicados así construídos denomínanse cuantificados.

Sintaxe

```
SELECT columnas das táboas do FROM
FROM ....
WHERE expresión1 opComparación ALL|SOME|ANY (consulta subordinada)
```

- **ALL:** Se usamos ALL o predicado cuantificado é VERDADEIRO se a comparación é certa para todos e cada un dos valores resultantes da sentencia subordinada.
- **SOME/ANY:** Se usamos SOME ou ANY (son equivalentes) o predicado cuantificado é VERDADEIRO se a comparación é certa para un calquera dos valores resultantes da sentenza subordinada.

Ao existir varias táboas pode darse ambigüidade nos nomes dos atributos, é dicir, en dúas táboas diferentes poden existir dous campos co mesmo nome. Para evitar esta confusión empregaremos *nomes locais* ou *de correlación*. Os nomes locais consisten en indicar diante do nome do atributo o nome da táboa seguido dun punto.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 6 e 7, no editor de consultas do SSMS.

- **Consulta de exemplo 6:** Nome e data de inicio das actividades con prezo menor que a cota máis cara existente. Se é menor que algunha será menor q a máis cara.

```
SELECT nome, convert(char(10), data_ini, 105) as data_ini
FROM ACTIVIDADE
WHERE  prezo < SOME (SELECT importe
                     FROM COTA);
```

Resultado da consulta do exemplo 6	
nome	**data_ini**
REPOSTARÍA XADREZ INICIACIÓN Á INFORMÁTICA PROBA PARA FULL JOIN	15-02-2015 20-03-2014 01-03-2015 12-02-2015

- **Consulta de exemplo 7:** Nome e data de inicio das actividades con prezo maior que a cota máis cara existente. Se é maior que todas será menor q a máis cara.

```
SELECT nome, convert(char(10), data_ini, 105) as data_ini
FROM ACTIVIDADE
WHERE  prezo > ALL (SELECT importe
                    FROM COTA);
```

Resultado da consulta do exemplo 7	
nome	**data_ini**
TENIS PARA PRINCIPIANTES	10-02-2014

5.5. Sentencias encadeadas

Unha sentenza subordinada doutra pode ter á vez unha subordinada dela. Cando isto sucede dise que son *sentenzas encadeadas*. Neste tipo de consultas a sentenza que non é subordinada de ningunha outra denomínase *sentenza externa*.

- Pódense considerar en varios niveis:
  - sentencia externa => 1º nivel
  - cada sentenza subordinada nova => 1 nivel máis

- Cando existen sentenzas encadeadas, diremos que unha sentenza é *antecedente* doutra, cando esta é unha subordinada directa, ou subordinada das súas subordinadas a calquera nivel.

Sentenza 1	Sentenza 2	Sentenza 3
SELECT ..... FROM ..... WHERE	(SELECT ..... FROM ..... WHERE	(SELECT ..... FROM ..... WHERE .....))
Sentenza externa 1º nivel	2º nivel	3º nivel

  - A sentenza 1 é antecedente da 2.
  - A sentenza 2 é antecedente da 3.
  - O número dado a cada sentenza correspóndese co nivel de cada unha delas.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 8, no editor de consultas do SSMS.

- **Consulta de exemplo 8:** NIF e nome completo dos socios que realizan actividades impartidas por profesores que cursan algunha actividade. Para resolvela non se poden utilizar joins.

```
SELECT NIF, nome, ape1, ape2
FROM   SOCIO
WHERE  numero IN (SELECT num_socio
                  FROM   SOCIO_REALIZA_ACTI
                  WHERE  id_actividade IN (SELECT identificador
                                           FROM   ACTIVIDADE
                                           WHERE  num_profesorado_imparte IN (SELECT num_profesorado
                                                                              FROM PROFE_CURSA_ACTI)));
```

Resultado da consulta do exemplo 8			
NIF	**nome**	**ape1**	**ape2**
11111112A 22222223B 44444445D	MARÍA MANUEL CARLA	GRAÑA SIEIRO VIEITO	UMIA CAMPOS GIL

5.6. Consultas correlacionadas

Nas consultas subordinadas, cando se especifique algunha columna dalgunha táboa do FROM nunha sentenza antecedente, diremos que a consulta é *correlacionada*.

Esta consulta non poderá avaliarse independentemente, senón que depende do valor dos campos na táboa da consulta antecedente.

Cando nunha consulta subordinada especificamos un nome de columna sen calificar, é dicir sen antepoñerlle o nome da táboa, interprétase que pertence á primeira táboa que a conteña, tendo en conta a seguinte orde:

- 1º) Nas táboas do seu propio FROM.
- 2º) Nas táboas do FROM da sentenza antecedente inmediata.
- 3º) Nas táboas do FROM da antecedente anterior e así sucesivamente ata chegar ao FROM da consulta externa.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 9, no editor de consultas do SSMS.

- **Consulta de exemplo 9:** NIF dos empregados cuxos salarios mensuais son maiores que a suma dos prezos das actividades que imparte. Para resolvela hai que usar unha subconsulta correlacionada e non se poden utilizar joins.

```
SELECT NIF
FROM EMPREGADO
WHERE salario_mes > (SELECT sum(prezo)
                     FROM ACTIVIDADE
                     WHERE num_profesorado_imparte=numero);
```

Na cláusula WHERE da subconsulta facemos referencia ao campo *numero* que non existe en ACTIVIDADE, polo que o servidor busca ese campo nunha táboa dunha consulta antecedente, neste caso a consulta externa ou de primeiro nivel sobre EMPREGADO. Neste caso calcularase o total dos prezos das actividades para cada fila de EMPREGADO, porque o campo *numero* varía para cada fila da táboa.

Resultado da consulta do exemplo 9
NIF
11111112A 22222223B 44444445D

5.7. Subconsultas na cláusula HAVING

Aínda que é máis habitual atopar subconsultas na cláusula WHERE, tamén podemos usalas no HAVING, como xa dixemos na introdución.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 10, no editor de consultas do SSMS.

- **Consulta de exemplo 10:** NIF e gasto en actividades realizadas dos socios que levan gastado en actividades máis do valor da cota máxima. Para resolvela non se poden utilizar joins.

```
SELECT s.nif, sum(a.prezo) as gasto_en_actividades
FROM SOCIO s,SOCIO_REALIZA_ACTI r, ACTIVIDADE a
WHERE s.numero=r.num_socio AND
      r.id_actividade=a.identificador
GROUP BY s.nif
HAVING sum(a.prezo) > (SELECT max(importe)
                       FROM COTA);
```

Resultado da consulta do exemplo 10	
nif	**gasto_en_actividades**
11111112A 22222223B	301.55 381.55

5.8. Tarefa de consultas con subconsultas en T-SQL

Unha vez copiados, executados e probados os exemplos das explicacións, realizarase o código T-SQL adecuado para obter a información que se pide en cada unha das consultas propostas na BD EMPRESA. Aínda que non se indique, nos resultados todas as columnas terán un nome que identifique correctamente a información que amosan.

- Consultas propostas na BD EMPRESA.

  - **Proposta 1.** Nome de todos os fabricantes dos que hai produtos na BD. Non se permite usar combinacións nesta consulta.
  - **Proposta 2.** Nome de todos os fabricantes dos que non hai produtos na BD. Non se permite usar combinacións nesta consulta.
  - **Proposta 3.** Número de pedido, cantidade e data de pedido para aqueles pedidos recibidos nos días en que un novo empregado foi contratado. Non se permite usar combinacións nesta consulta.
  - **Proposta 4.** Cidade e obxectivo das sucursais cuxo obxectivo supera a media das cotas de todos os vendedores da BD. Non se permite usar combinacións nesta consulta.
  - **Proposta 5.** Número de empregado e cantidade media dos pedidos daqueles empregados cuxa cantidade media de pedido é superior á cantidade media global (de todos os pedidos). Non se permite usar combinacións nesta consulta.
  - **Proposta 6.** Nome dos clientes que aínda non fixeron pedidos. Non se permite usar combinacións nesta consulta.
  - **Proposta 7.** Nome completo dos empregados cuxas cotas son iguais ou superiores ao obxectivo da sucursal da cidade de Vigo. Ten en conta que se a cota dun vendedor (empregado) é nula debemos considerala como un 0, e do mesmo xeito actuaremos co obxectivo da sucursal. Non se permite usar combinacións nesta consulta.
  - **Proposta 8.** Nome dos produtos para os que existe polo menos un pedido que ten unha cantidade de polo menos 20 unidades. Hai que lembrar que a identificación dun produto faise pola combinación do código do fabricante e o do produto. A solución deberá facerse empregando o predicado EXISTS cunha subconsulta correlacionada. Non se permite usar combinacións.
  - **Proposta 9.** Cidades das sucursais onde exista algún empregado cuxa cota de vendas represente máis do 80% do obxectivo da oficina onde traballa. Para resolver esta consulta deberase empregar unha subconsulta correlacionada precedida de ANY.
  - **Proposta 10.** Nome dos clientes cuxos empregados asignados traballan en sucursais da rexión OESTE. Non se poden usar joins, só subconsultas encadeadas.

6. Consultas con funcións integradas no xestor

O obxectivo deste apartado é aprender a usar nas consultas SQL funcións incorporadas no xestor de base de datos.

Usaremos o SXBD (Sistema Xestor de Base de Datos) MS SQLServer, polo que a sintaxe que se explicará será a da linguaxe Transact-SQL (T-SQL en adiante), propia do xestor.

Explicarase a sintaxe T-SQL e uso das funcións integradas máis usadas, atendendo á seguinte clasificación:

- De conversión de tipo.
- De cadea de carácteres.
- De tratamento de datas.
- Funcións matemáticas.
- De sistema.
- Funcións diversas.

6.1. Función integrada. Definición

Unha función, no contexto informático, defínese como un grupo de instrucións deseñadas para realizar unha tarefa específica, e que poden devolver un valor.

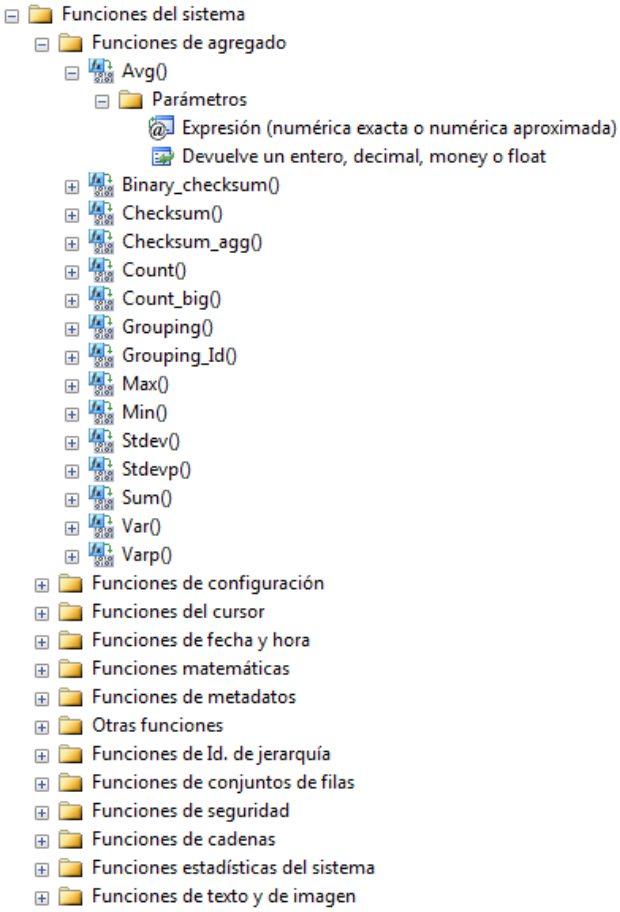
As funcións poden recibir valores, datos, a partires dos que se realizan as instrucións internas da función. Estes valores que se di que se lle pasan á función denomínanse *parámetros de entrada*.

A linguaxe T-SQL permite deseñar funcións, coñecidas como funcións definidas polo usuario, e que se verán nunha unidade didáctica posterior. As que aquí nos ocupan, as funcións integradas, son aquelas que xa veñen programadas no xestor, e que podemos dicir que nos aforran traballo, xa que non as temos que codificar nós.

Para usar unha función bastará con executalas nunha consulta, tendo en conta os parámetros que recibe e o tipo de dato que devolve. O feito de executar unha función coñecese como chamar unha función. Falaremos de usar ou chamar unha función, ou da execución ou chamada dunha función.

Para coñecer a clasificación das distintas funcións que fai o propio SQL Server, nada máis sinxelo que acceder ao SSMS, e no seu explorador de obxectos, nunha BD calquera escollemos o cartafol Programación, e baixamos dentro da árbore de cartafoles ata o cartafol de Funcións do sistema. Cada un dos cartafoles ten as funcións integradas de cada tipo cos parámetros que acepta e o valor que devolve. Podemos ver na seguinte imaxe un exemplo para a función de agregado AVG, que xa

vimos no apartado sobre consultas resumo.



Listaxe de funcións integradas do explorador de obxectos do xestor SQL Server 2014

A nosa clasificación non se corresponde coa que fai o xestor SQL Server, para non depender das distintas versións do mesmo, pero está ben poder acceder de xeito rápido a esta clasificación para localizar unha función da que, por exemplo, non recordamos o nome exacto, pero si para que se usa.

A continuación explicaremos a sintaxe T-SQL e o uso das funcións integradas no xestor SQL Server máis comunmente usadas.

Nalgunhas consultas empregarase a instrución SELECT só para amosar o resultado das funcións, sen obter información de ningunha táboa.

6.2. Funcións de conversión de tipo

As funcións de conversión de tipos permiten converter unha expresión dun tipo de datos a outro de forma explícita, é dicir, indicando o tipo de dato no que imos converter.

As dúas que máis imos empregar nas nosas consultas son *cast* e *convert*.

Tanto a función *cast* como a función *convert* empréganse para a conversión de tipos de datos. A continuación explícase a súa sintaxe e exemplos do seu uso.

Función cast:

Sintaxe función cast	
`cast(expresión AS tipo_dato)`	
Parámetros de entrada	▪ expresión é calquera expresión válida como un valor constante, o nome dun campo, unha operación ou a chamada a unha función que devolve un valor. ▪ tipo_dato é o tipo de dato de destino.
Saída	▪ Devolve a expresión traducida ao tipo de dato de destino tipo_dato.

A función *cast* transforma a *expresión* indicada entre parénteses no *tipo_dato* indicado despois do AS.

Utilizarémola principalmente nas operacións de concatenación de cadeas para transformar os datos que non son cadeas en tipo char ou varchar, para poder facer a concatenación, como podemos ver no exemplo 1.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 1, no editor de consultas do SSMS.

- **Consulta de exemplo 1:** Esta consulta de exemplo amosa nunha soa columna de nome *datos_empregados,* o número identificador e o nome completo de cada empregado co formato:

  número identificador - apelido1 apelido2, nome

```
--Solución CON ERROS
SELECT numero+' - '+ape1+' '+ape2+', '+nome as datos_empregados
FROM EMPREGADO;
--Solución CORRECTA
SELECT cast(numero as varchar(5))+' - '+ape1+' '+ape2+', '+nome
       as datos_empregados
FROM EMPREGADO;
```

  Na primeira solución, sen empregar a función *cast*, o xestor intenta facer unha suma e transformar o valor de ape1 en *varchar*, por iso devolve o seguinte erro:

```
Error de conversión al convertir el valor varchar ', ' al tipo de datos int.
```

  Na segunda solución ao utilizar a función *cast* para transformar o número nunha cadea de caracteres, o xestor fai correctamente a concatenación. A solución correcta devolverá o seguinte resultado.

Resultado da consulta do exemplo 1
datos_empregados

100 - GARCÍA PÉREZ, MARÍA
200 - REGO PENA, CARLOS
300 - POSE VARELA, JUANA
400 - GONZÁLEZ ÍNSUA, JOSÉ

Función convert:

Sintaxe función convert	
`convert(tipo_dato, expresión [,estilo])`	
Parámetros de entrada	▪ tipo_dato é o tipo de dato de destino. ▪ expresión é calquera expresión válida como un valor constante, o nome dun campo, unha operación ou a chamada a unha función que devolve un valor. ▪ estilo é unha expresión de tipo enteiro que especifica como convert traducirá expresión.
Saída	▪ Devolve a expresión traducida ao tipo de dato de destino tipo_dato.

A función *convert* transforma a *expresión* no *tipo_dato* indicado como primeiro parámetro xusto despois do parénteses.

O valor do terceiro parámetro *estilo* dependerá do tipo de dato que indiquemos. Utilizarase principalmente nos tipos de dato de datas. Permítenos escoller o formato co que representar unha data como dd/mm/aaaa, dd-mm-aaaa, dd mes aa. Verémolo no seguinte exemplo.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 2, no editor de consultas do SSMS.

- **Consulta de exemplo 2:** Esta consulta de exemplo amosa nunha soa columna o nome de todas as actividades e o día que comezan co formato:

  nome_actividade comeza o *data_inicio_da_actividade*

```
--Solución1
SELECT nome+' comeza o '+convert(char(20),data_ini)
       as acti_sen_estilo_data
FROM ACTIVIDADE;
--Solución2
SELECT nome+' comeza o '+convert(char(20),data_ini,103)
       as acti_estilo_data_103
FROM ACTIVIDADE;
--Solución3
SELECT nome+' comeza o '+convert(char(20),data_ini,113)
       as acti_estilo_data_113
FROM ACTIVIDADE;
--Solución4
SELECT nome+' comeza ás '+convert(char(20),data_ini,108)
       as acti_estilo_data_108
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 2. Solución 1
acti_sen_estilo_data
TENIS PARA PRINCIPIANTES comeza o Feb 10 2014 4:00PM REPOSTARÍA comeza o Feb 15 2015 5:00PM XADREZ comeza o Mar 20 2014 4:30PM INICIACIÓN Á INFORMÁTICA comeza o Mar 1 2015 4:30PM
Resultado da consulta do exemplo 2. Solución 2
acti_sen_estilo_data_103
TENIS PARA PRINCIPIANTES comeza o 10/02/2014 REPOSTARÍA comeza o 15/02/2015 XADREZ comeza o 20/03/2014 INICIACIÓN Á INFORMÁTICA comeza o 01/03/2015
Resultado da consulta do exemplo 2. Solución 3
acti_sen_estilo_data_113
TENIS PARA PRINCIPIANTES comeza o 10 Feb 2014 16:00:00 REPOSTARÍA comeza o 15 Feb 2015 17:00:00 XADREZ comeza o 20 Mar 2014 16:30:00 INICIACIÓN Á INFORMÁTICA comeza o 01 Mar 2015 16:30:00
Resultado da consulta do exemplo 2. Solución 4
acti_sen_estilo_data_108
TENIS PARA PRINCIPIANTES comeza ás 16:00:00 REPOSTARÍA comeza ás 17:00:00 XADREZ comeza ás 16:30:00 INICIACIÓN Á INFORMÁTICA comeza ás 16:30:00

Na primeira solución, non indicamos ningún estilo, polo que amosa o formato predeterminado que se corresponde co número de *estilo* 0 ou 100.

Nas seguintes solucións, en función do *estilo* indicado no *convert*, así se amosa a data cun formato ou outro, con hora ou sen ela, ou a hora unicamente.

Para consultar o resto dos estilos posibles da función *convert* pódese consultar os libros de axuda online de SQL Server na seguinte ligazón https://msdn.microsoft.com/es-es/library/ms187928

6.3. Funcións de cadea de caracteres

As seguintes funcións realizan unha operación sobre unha cadea de caracteres e devolven un valor alfanumérico (de cadea) ou un valor numérico, segundo a tarefa da función.

Función ascii:

Sintaxe función ascii	
`ascii(expresión)`	
Parámetros de entrada	▪ expresión é un tipo de dato char ou varchar.
Saída	▪ Devolve un número enteiro que se corresponde cun código ASCII.

A función *ascii* devolve o valor do código ASCII do carácter que está máis á esquerda da expresión alfanumérica que se indica como parámetro entre parénteses.

Función char:

Sintaxe función char	
`char(expresión)`	
Parámetros de entrada	▪ expresión é un enteiro entre 0 e 255.
Saída	▪ Devolve un carácter (char(1)), o carácter ASCII que se corresponde con expresión. ▪ Devolve NULL no caso de que o número pasado estea fóra do intervalo [0-255].

A función *char* recibe como parámetro un número enteiro entre 0 e 255. Transforma o número que se lle pasa no carácter que lle corresponde a ese número no código ASCII. Devolverá NULL se lle pasamos á función un número que estea fóra do intervalo.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 3, no editor de consultas do SSMS.

- **Consulta de exemplo 3:** Esta consulta amosa o uso das funcións ascii e char para a letra A. A consulta non se fai sobre ningunha táboa, neste caso, como xa explicamos na introdución, empregamos a instrución SELECT só para amosar o resultado, como se fose un print.

```
SELECT ascii('A') as codigo_ascii_de_A,
       char(65) as caracterASCII_do_codigo65;
```

Resultado da consulta do exemplo 3	
codigo_ascii_de_A	**caracterASCII_do_codigo65**
65	A

Función left:

Sintaxe función left	
`left(expresión, N)`	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos. ▪ N é o número de caracteres que queremos extraer da parte esquerda da cadea.
Saída	▪ Devolve varchar se expresión é de tipo de datos non Unicode. ▪ Devolve nvarchar cando expresión é de tipo Unicode.

A función *left* devolve os N caracteres máis á esquerda da expresión pasada como primeiro parámetro.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 4, no editor de consultas do SSMS.

- **Consulta de exemplo 4:** Listaxe abreviada das catro primeiras letras dos nomes das actividades.

```
SELECT left(nome, 4) as nome_abrev_acti
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 4
nome_abrev_acti
TENI REPO XADR INIC

Función len:

Sintaxe función len	
`len(expresión)`	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos. ▪ expresión pode ser un valor constante, unha variable ou unha columna de datos binarios ou de caracteres.
Saída	▪ Devolve o número de caracteres da cadea pasada como parámetro, descontando os espazos en branco do final, se existen. ▪ O tipo de datos devolto será int, exceptuando nos casos de que expresión sexa varchar(max), nvarchar(max) ou varbinary(max), que o tipo de dato devolto será bigint.

Devolve o número de caracteres da expresión, sen contar os espazos en branco da dereita da cadea.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 5, no editor de consultas do SSMS.

- **Consulta de exemplo 5:** Nome e número de caracteres dos nomes dos empregados.

```
SELECT nome, len(nome) as lonxitude_nomes_propios
FROM EMPREGADO;
```

Resultado da consulta do exemplo 5	
nome	**lonxitude_nomes_propios**
MARÍA CARLOS JUANA JOSÉ	5 6 5 4

Función lower:

Sintaxe función lower	
`lower(expresión)`	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos.
Saída	▪ Devolve a expresión de caracteres pasada á función, convertida en minúsculas. O tipo da cadea devolta será varchar ou nvarchar.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 6 e 7, no editor de consultas do SSMS.

- **Consulta de exemplo 6:** Nomes dos empregados tal como están gardados na BD, e nunha segunda columna os nomes dos empregados en minúsculas.

```
SELECT nome, lower(nome) as nome_minusculas
FROM EMPREGADO;
```

Resultado da consulta do exemplo 6	
nome	**nome_minusculas**
MARÍA CARLOS JUANA JOSÉ	maría carlos juana josé

- **Consulta de exemplo 7:** Nomes completos dos socios en minúsculas. Primeiro pasando a minúsculas cada campo, e na segunda columna pasando a minúsculas despois de facer a concatenación dos campos.

```
SELECT lower(nome)+' '+lower(ape1)+' '+isnull(lower(ape2),'') as nome_minusculas1,
       lower(nome+' '+ape1+' '+isnull(ape2,'')) as nome_minusculas2
FROM SOCIO;
```

Resultado da consulta do exemplo 7	
nome_minusculas1	**nome_minusculas2**
maría graña umia manuel sieiro campos jorge del carmen lérez carla vieito gil	maría graña umia manuel sieiro campos jorge del carmen lérez carla vieito gil

Función ltrim:

Sintaxe función ltrim	
`ltrim(expresión)`	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos.
Saída	▪ Devolve a expresión de caracteres pasada á función, despois de quitarlle os espazos en branco iniciais.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 8, no editor de consultas do SSMS.

- **Consulta de exemplo 8:** Nesta consulta amosamos unha cadea con espazos á esquerda na primeira columna, e a mesma cadea sen espazos á esquerda.

```
SELECT '    cadea_exemplo' as cadea_con_espazos,
       ltrim('    cadea_exemplo') as cadea_sen_espazos;
```

Resultado da consulta do exemplo 8	
cadea_con_espazos	**cadea_sen_espazos**
cadea_exemplo	cadea_exemplo

Función nchar:

Sintaxe función nchar	
`nchar(expresión)`	
Parámetros de entrada	▪ expresión é un número enteiro positivo.
Saída	▪ Devolve o carácter Unicode correspondente ao código enteiro pasado.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 9, no editor de consultas do SSMS.

- **Consulta de exemplo 9:** Carácter unicode do código 248.

```
SELECT nchar(248) as caracter_unicode_248;
```

Resultado da consulta do exemplo 9
caracter_unicode_248
ø

Función replace:

Sintaxe función replace	
`replace(cadea, subcadea_buscada, cadea_substitución)`	
Parámetros de entrada	▪ cadea é unha cadea de datos binarios ou alfanuméricos. ▪ subcadea_buscada é o conxunto de caracteres que se vai buscar. Pode ser de datos binarios ou de caracteres, pero no pode ser unha cadea baleira (''). ▪ cadea_substitución é a cadea coa que se vai substituír a subcadea buscada.
Saída	▪ Devolve a cadea inicial modificada coa subcadea_buscada cambiada pola cadea_substitución, tantas veces como aparecese.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 10 e 11, no editor de consultas do SSMS.

- **Consulta de exemplo 10:** Nesta consulta obtemos o nome das actividades e nunha segunda columna o nome das actividades sen acento gráfico no a.

```
SELECT nome,
       replace(nome, 'Á', 'A') nome_sen_acento_noA
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 10	
nome	**nome_sen_acento_noA**
TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ INICIACIÓN Á INFORMÁTICA	TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ INICIACIÓN A INFORMATICA

- **Consulta de exemplo 11:** Nesta consulta obtemos o nome das actividades e nunha segunda columna o nome das actividades sen acento gráfico en ningunha vogal.

```
SELECT nome,
       replace(replace(replace(replace(
       replace(nome,'Á', 'A'),'É', 'E'),'Í','I'),'Ó','O'),'Ú','U')
       as nome_sen_acentos
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 11	
nome	**nome_sen_acentos**
TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ INICIACIÓN Á INFORMÁTICA	TENIS PARA PRINCIPIANTES REPOSTARIA XADREZ INICIACION A INFORMATICA

Función right:

Sintaxe función right	
`right(expresión, N)`	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos. ▪ N é o número de caracteres que queremos extraer da parte dereita da cadea.
Saída	▪ Devolve varchar se expresión é de tipo de datos non Unicode. ▪ Devolve nvarchar cando expresión é de tipo Unicode.

A función *right* devolve os N caracteres máis á dereita da expresión pasada como primeiro parámetro.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 12, no editor de consultas do SSMS.

- **Consulta de exemplo 12:** Listaxe abreviada das catro últimas letras dos nomes das actividades.

```
SELECT right(nome, 4) as nome_abrev_acti
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 12
nome_abrev_acti
NTES ARÍA DREZ TICA

Función rtrim:

Sintaxe función rtrim	
rtrim(expresión)	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos.
Saída	▪ Devolve a expresión de caracteres pasada á función, despois de quitarlle os espazos en branco iniciais.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 13, no editor de consultas do SSMS.

- **Consulta de exemplo 13:** Nesta consulta amosamos unha cadea con espazos á dereita na primeira columna, e a mesma cadea sen espazos á dereita. Para poder verificalo engadimos un punto ao final da cadea.

```
SELECT 'cadea_exemplo     ' +'.' as cadea_con_espazos,
       rtrim('cadea_exemplo     ') +'.' as cadea_sen_espazos;
```

Resultado da consulta do exemplo 13	
cadea_con_espazos	**cadea_sen_espazos**
cadea_exemplo .	cadea_exemplo.

Función substring:

Sintaxe función substring	
substring(expresión, pos_ini, lonxitude)	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos. ▪ pos_ini é un enteiro que especifica a posición do primeiro carácter a devolver. Hai que ter en conta que o primeiro carácter da cadea ocupa a posición 1. ▪ lonxitude é un enteiro que indica o número de caracteres a devolver.
Saída	▪ Devolve unha subcadea de expresión, empezando pola posición pos_ini e de tantos caracteres como indique a lonxitude.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 14, no editor de consultas do SSMS.

- **Consulta de exemplo 14:** Nesta consulta obtemos os segundo, terceiro, cuarto e quinto caracteres do nome das cotas.

```
SELECT nome, substring(nome, 2,4) as subcadea_nome_cota
FROM COTA;
```

Resultado da consulta do exemplo 14	
nome	**subcadea_nome_cota**
DE HONRA FAMILIAR HABITUAL GRATUITA	E HO AMIL ABIT RATU

Función unicode:

Sintaxe función unicode	
unicode(expresión)	
Parámetros de entrada	▪ expresión é un tipo de dato char ou varchar.
Saída	▪ Devolve un número enteiro que se corresponde cun código Unicode.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 15, no editor de consultas do SSMS.

- **Consulta de exemplo 15:** Esta consulta de exemplo amosa o código unicode do símbolo ø.

```
SELECT  unicode('ø') as "codigo_unicode_deø";
```

Resultado da consulta do exemplo 15
codigo_unicode_deø
248

Función upper:

Sintaxe función upper	
upper(expresión)	
Parámetros de entrada	▪ expresión é unha cadea de datos binarios ou alfanuméricos.
Saída	▪ Devolve a expresión de caracteres pasada á función, convertida en maiúsculas. O tipo da cadea devolta será varchar ou nvarchar.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 16, no editor de consultas do SSMS.

- **Consulta de exemplo 16:** Observacións das actividades tal como están gardadas e nunha segunda columna as mesmas observacións en maiúsculas.

```
SELECT observacions, upper(observacions) as observacions_en_maiusculas
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 16	
observacions	**observacions_en_maiusculas**
Precísase raqueta e 1 bote de pelotas NULL NULL Impartirase SW_libre	PRECÍSASE RAQUETA E 1 BOTE DE PELOTAS NULL NULL IMPARTIRASE SW_LIBRE

6.4. Funcións de tratamento de datas

Imos empezar lembrando os tipos de datos de data e hora:

Tipo de datos	Formato	Intervalo	Precisión	Axuste de zona horaria
time	hh:mm:ss[. nnnnnnn]	De 00:00:00.0000000 a 23:59:59.9999999	100 nanosegundos	Non
date	aaaa-mm-dd	De 0001-01-01 a 9999-12-31	1 día	Non
smalldatetime	aaaa-mm-dd hh:mm:ss	De 1900-01-01 a 2079-06-06	1 minuto	Non
datetime	aaaa-mm-dd hh:mm:ss[. nnn]	De 1753-01-01 a 9999-12-31	0,00333 segundos	Non
datetime2	aaaa-mm-dd hh:mm:ss[. nnnnnnn]	De 0001-01-01 00:00:00.0000000 a 9999-12-31 23:59:59.9999999	100 nanosegundos	Non
datetimeoffset	aaaa-mm-dd hh:mm:ss[. nnnnnnn] [+|-]hh:mm	De 0001-01-01 00:00:00.0000000 a 9999-12-31 23:59:59.9999999 (en UTC)	100 nanosegundos	Si

Funcións que obteñen valores de data e hora do sistema

Hai que aclarar antes de nada, que os valores de data e hora, a súa exactitude, dependen da versión de Windows na que se executa a instancia de SQL Server.

Función sysdatetime:

Sintaxe función sysdatetime	
`sysdatetime()`	
Parámetros de entrada	▪ Ningún.
Saída	▪ Devolve en valor datetime2(7) a data e hora do equipo no que se está a executar a instancia de SQL Server. Sen axuste de zona horaria.

Función sysdatetimeoffset:

Sintaxe función sysdatetimeoffset	
`sysdatetimeoffset()`	
Parámetros de entrada	▪ Ningún.
Saída	▪ Devolve en valor datetimeoffset(7) a data e hora do equipo no que se está a executar a instancia de SQL Server. Con axuste de zona horaria.

Función sysutcdatetime:

Sintaxe función sysutcdatetime	
`sysutcdatetime()`	
Parámetros de entrada	▪ Ningún.
Saída	▪ Devolve en valor datetime2(7) a data e hora do equipo no que se está a executar a instancia de SQL Server. A data e hora devólvense como hora universal coordinada (UTC).

Función getdate:

Sintaxe función getdate	
`getdate()`	
Parámetros de entrada	▪ Ningún.
Saída	▪ Devolve en valor datetime a data e hora do equipo no que se está a executar a instancia de SQL Server. Sen axuste de zona horaria.

Función getutcdate:

Sintaxe función getutcdate	
`getutcdate()`	
Parámetros de entrada	▪ Ningún.
Saída	▪ Devolve en valor datetime a data e hora do equipo no que se está a executar a instancia de SQL Server. A data e hora devólvense como hora universal coordinada (UTC).

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 17, no editor de consultas do SSMS.

- **Consulta de exemplo 17:** Nesta consulta amosamos a data do sistema empregando as distintas funcións.

```
SELECT  sysdatetime() as data_formato1,
        sysdatetimeoffset() as data_formato2,
        sysutcdatetime() as data_formato3,
        getdate() as data_formato4,
        getutcdate() as data_formato5;
```

Resultado da consulta do exemplo 17				
data_formato1	**data_formato2**	**data_formato3**	**data_formato4**	**data_formato5**
2015-02-26 17:28:59.1945901	2015-02-26 17:28:59.1945901 +01:00	2015-02-26 16:28:59.1945901	2015-02-26 17:28:59.193	2015-02-26 16:28:59.193

Funcións que obteñen partes de data e hora

Función datename:

Sintaxe función datename	
`datename(parte_da_data_hora, data_hora)`	
Parámetros de entrada	▪ parte_da_data_hora é a parte da data que se quere obter. Os valores poden ser completos ou empregar as abreviaturas indicadas entre parénteses: – year (yy, yyyy) – quarter (qq, q) – month (mm, m) – dayofyear (dy, y) – day (dd, d) – week (wk, ww) – weekday (dw, w) – hour (hh) – minute (mi, n) – second (ss, s)

	– millisecond (ms) – microsecond (mcs) – nanosecond (ns) – TZoffset (tz) – ISO_WEEK (ISOWK, ISOWW) ▪ Data ou hora da que se vai extraer unha parte.
Saída	▪ Devolve unha cadea de caracteres que representa o parámetro parte_da_data_hora, da data_hora pasada á función. ▪ O valor devolto depende do idioma definido con SET LANGUAGE.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 18, no editor de consultas do SSMS.

- **Consulta de exemplo 18:** Esta consulta de exemplo amosa o uso da función *datename* para obter o nome de distintas partes dunha data e hora concreta: nanosegundos, segundos, minutos, hora, día, semana, mes e ano da data '20-01-2015 17:28:59.1945901'.

```
SELECT  datename(ns, '20-01-2015 17:28:59.1945901') as nanosegundos,
        datename(second, '20-01-2015 17:28:59.1945901') as segundos,
        datename(minute, '20-01-2015 17:28:59.1945901') as minutos,
        datename(hour, '20-01-2015 17:28:59.1945901') as hora,
        datename(day, '20-01-2015 17:28:59.1945901') as dia,
        datename(ww, '20-01-2015 17:28:59.1945901') as semana,
        datename(month, '20-01-2015 17:28:59.1945901') as mes,
        datename(year, '20-01-2015 17:28:59.1945901') as ano;
```

Resultado da consulta do exemplo 18							
nanosegundos	**segundos**	**minutos**	**hora**	**dia**	**semana**	**mes**	**ano**
194590100	59	28	17	20	4	Enero	2015

Función datepart:

Sintaxe función datepart	
`datepart(parte_da_data_hora, data_hora)`	
Parámetros de entrada	▪ parte_da_data_hora é a parte da data que se quere obter. Os valores son os mesmos que os da función datename anterior. ▪ Data ou hora da que se vai extraer unha parte.
Saída	▪ Devolve un enteiro que representa o parámetro parte_da_data_hora, da data_hora pasada á función. ▪ O valor devolto depende do idioma definido con SET LANGUAGE.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 19, no editor de consultas do SSMS.

- **Consulta de exemplo 19:** Esta consulta de exemplo amosa o uso da función *datepart* para obter o as distintas partes dunha data e hora concreta: nanosegundos, segundos, minutos, hora, día, semana, mes e ano da data '20-01-2015 17:28:59.1945901'.

```
SELECT  datepart(ns, '20-01-2015 17:28:59.1945901') as nanosegundos,
        datepart(second, '20-01-2015 17:28:59.1945901') as segundos,
        datepart(minute, '20-01-2015 17:28:59.1945901') as minutos,
        datepart(hour, '20-01-2015 17:28:59.1945901') as hora,
        datepart(day, '20-01-2015 17:28:59.1945901') as dia,
        datepart(ww, '20-01-2015 17:28:59.1945901') as semana,
        datepart(month, '20-01-2015 17:28:59.1945901') as mes,
        datepart(year, '20-01-2015 17:28:59.1945901') as ano;
```

Resultado da consulta do exemplo 19							
nanosegundos	**segundos**	**minutos**	**hora**	**dia**	**semana**	**mes**	**ano**
194590100	59	28	17	20	4	1	2015

Función day:

Sintaxe función day	
`day(data_hora)`	
Parámetros de entrada	▪ data_hora que pode ser unha expresión, unha columna, unha variable ou unha constante de tipo data_hora.
Saída	▪ Devolve un enteiro que representa o día (día do mes) da data_hora especificada. ▪ Devolve o mesmo que a funcións datepart(day, data_hora).

Función month:

Sintaxe función month	
`month(data_hora)`	
Parámetros de entrada	▪ data_hora que pode ser unha expresión, unha columna, unha variable ou unha constante de tipo data_hora.
Saída	▪ Devolve un enteiro que representa o número do mes da data_hora especificada. ▪ Devolve o mesmo que a funcións datepart(month, data_hora).

Función year:

Sintaxe función year	
`year(data_hora)`	
Parámetros de entrada	▪ data_hora que pode ser unha expresión, unha columna, unha variable ou unha constante de tipo data_hora.
Saída	▪ Devolve un enteiro que representa o ano da data_hora especificada. ▪ Devolve o mesmo que a funcións datepart(year, data_hora).

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 20 e 21 no editor de consultas do SSMS.

- **Consulta de exemplo 20:** Esta consulta devolve o día, mes e ano da data '20-01-2015 17:28:59.1945901'.

```
SELECT day('20-01-2015 17:28:59.1945901') as dia,
       month('20-01-2015 17:28:59.1945901') as mes,
       year('20-01-2015 17:28:59.1945901') as ano;
```

Resultado da consulta do exemplo 20		
dia	**mes**	**ano**
20	1	2015

- **Consulta de exemplo 21:** Data de inicio das actividades nunha primeira columna, e nas seguintes amosarase o día, mes e ano desa mesma data de inicio.

```
SELECT data_ini,
       day(data_ini) as dia,
       month(data_ini) as mes,
       year(data_ini) as ano
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 21			
data_ini	**dia**	**mes**	**ano**
2014-02-10 16:00:00.000 2015-02-15 17:00:00.000 2014-03-20 16:30:00.000 2015-03-01 16:30:00.000	10 15 20 1	2 2 3 3	2014 2015 2014 2015

Funcións que obteñen a data e hora dadas as partes

Función datefromparts:

Sintaxe función datefromparts	
`datefromparts(ano, mes, día)`	
Parámetros de entrada	▪ Todos os parámetros de entrada son enteiros que especifican o que o nome indica.
Saída	▪ Devolve un valor date para o ano, mes e día especificados entre parénteses.

Función datetimefromparts:

Sintaxe función datetimefromparts	
`datetimefromparts(ano, mes, día, hora, minuto, segundos, milisegundos)`	
Parámetros de entrada	▪ Todos os parámetros de entrada son enteiros que especifican o que o nome indica.
Saída	▪ Devolve un valor datetime da data_hora formado pola partes indicadas entre os parénteses.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 22, no editor de consultas do SSMS.

- **Consulta de exemplo 22:** Esta consulta obtén a data a partir do día 14, do mes 4 e do ano 1999 nunha primeira columna. Nunha segunda columna obterase a data formada polo día 14, do mes 4, do ano 1999, hora 21, minuto 15, 58 segundos e 2 milisegundos.

```
SELECT  datefromparts(1999,4,14) as data_das_partes,
        datetimefromparts(1999,4,14,21,15,58,2) as data_hora_das_partes;
```

Na primeira solución, sen empregar a función *cast*, o xestor intenta facer unha suma e transformar o valor de ape1 en *varchar*, por iso devolve o seguinte erro:

Resultado da consulta do exemplo 22	
data_da_partes	**data_hora_das_partes**
1999-04-14	1999-04-14

Funcións que obteñen diferenzas de data e hora

Función datediff:

Sintaxe función datediff	
`datediff(parte_da_data_hora, data_ini, data_fin)`	
Parámetros de entrada	▪ parte_da_data_hora é a unidade de tempo na que queremos calcular a diferenza entre as datas pasadas á función, anos, días, meses... Os valores poden ser completos ou empregar as abreviaturas indicadas entre parénteses: – year (yy, yyyy) – quarter (qq, q) – month (mm, m) – dayofyear (dy, y) – day (dd, d) – week (wk, ww) – weekday (dw, w) – hour (hh) – minute (mi, n) – second (ss, s) – millisecond (ms) – microsecond (mcs) – nanosecond (ns) ▪ data_ini é a data de inicio do rango do que queremos calcular a diferenza en anos, días, meses, ou o que indiquemos en parte_da_data_hora. ▪ data_fin é a data de fin di rango do que queremos calcular a diferenza en anos, días, meses, ou o que indiquemos en parte_da_data_hora.
Saída	▪ Devolve a diferenza entre a data de inicio e a data de fin calculada en anos, días, meses, ou o que indiquemos en parte_da_data_hora.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 23, no editor de consultas do SSMS.

- **Consulta de exemplo 23:** Obtemos as datas de inicio e fin, e duración en días, meses, semanas e horas de todas as actividades.

```
SELECT  data_ini, data_fin,
        datediff(day, data_ini, data_fin) as dias_duracion,
        datediff(month, data_ini, data_fin) as meses_duracion,
        datediff(week, data_ini, data_fin) as semanas_duracion,
        datediff(hour, data_ini, data_fin) as horas_duracion
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 23					
data_ini	**data_fin**	**dias_duracion**	**meses_duracion**	**semanas_duracion**	**horas_duracion**
2014-02-10 16:00:00.000	2014-10-10 20:00:00.000	242	8	34	5812
2015-02-15 17:00:00.000	2015-03-15 19:00:00.000	28	1	4	674
2014-03-20 16:30:00.000	2014-06-20 17:30:00.000	92	3	13	2209
2015-03-01 16:30:00.000	2015-05-01 17:30:00.000	61	2	8	1465

Funcións que modifican valores de data hora

Función dateadd:

Sintaxe función dateadd	
`dateadd(parte_da_data_hora, N, data_hora)`	
Parámetros de entrada	▪ parte_da_data_hora é a unidade de tempo que lle queremos engadir, ou quitar á data pasada á función, anos, días, meses... Os valores son os mesmos que os da función datediff anterior. ▪ N, se é positivo será o que lle engadimos á data_hora, e se é negativo será o que se lle quita á data_hora. ▪ data_hora que queremos atrasar ou adiantar.
Saída	▪ Devolve a data_hora retrasada en N, ou adiantada en -N anos, días, meses, ou o que indiquemos en parte_da_data_hora.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 24, no editor de consultas do SSMS.

- **Consulta de exemplo 24:** Listaxe coa data de inicio das actividades na primeira columna. Nas seguintes aparecerán: a data de inicio adiantada 1 ano, a data de inicio adiantada 3 meses, a data de inicio retrasada 4 días e a data de inicio retrasada 2 horas.

```
SELECT  data_ini,
        dateadd(year, -1, data_ini) as data_ini_adiantada_1ano,
        dateadd(month, -3, data_ini) as data_ini_adiantada_3meses,
        dateadd(day, 4, data_ini) as data_ini_retrasada_4dias,
        dateadd(hour, 2, data_ini) as data_ini_retrasada_2horas
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 24

data_ini	data_ini_adiantada1ano	data_ini_adiantada3meses	data_ini_retrasada4dias	data_ini_retrasada2horas
2014-02-10 16:00:00.000	2013-02-10 16:00:00.000	2013-11-10 16:00:00.000	2014-02-14 16:00:00.000	2014-02-10 18:00:00.000
2015-02-15 17:00:00.000	2014-02-15 17:00:00.000	2014-11-15 17:00:00.000	2015-02-19 17:00:00.000	2015-02-15 19:00:00.000
2014-03-20 16:30:00.000	2013-03-20 16:30:00.000	2013-12-20 16:30:00.000	2014-03-24 16:30:00.000	2014-03-20 18:30:00.000
2015-03-01 16:30:00.000	2014-03-01 16:30:00.000	2014-12-01 16:30:00.000	2015-03-05 16:30:00.000	2015-03-01 18:30:00.000

Funcións que validan valores de data e hora

Función isdate:

Sintaxe función isdate	
`isdate(expresión)`	
Parámetros de entrada	▪ expresión é unha cadea de caracteres.
Saída	▪ Devolve 1 se expresión é un valor de tipo time, date ou datetime válido, se non o é devolve 0. ▪ Tamén devolve 0 se a expresión é un valor datetime2.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 25, no editor de consultas do SSMS.

- **Consulta de exemplo 25:** Na seguinte consulta comprobamos a validez de diferentes datas. As columnas co nome *data_erronea* teñen un formato incorrecto.

```
SELECT  isdate('30/02/2015') as data_erronea,
        isdate('28/02/2015') as data_OK,
        isdate('30041951') as data_erronea,
        isdate('19510430') as data_OK,
        isdate('01-01-1800') as data_OK,
        isdate('01-01-2014 28:00:00') as data_erronea,
        isdate('01-01-2014 23:00:00') as data_OK;
```

Na primeira solución, sen empregar a función *cast*, o xestor intenta facer unha suma e transformar o valor de ape1 en *varchar*, por iso devolve o seguinte erro:

Resultado da consulta do exemplo 25						
data_erronea	data_OK	data_erronea	data_OK	data_OK	data_erronea	data_OK
0	1	0	1	1	0	1

6.5. Funcións matemáticas

Función ceiling:

Sintaxe función ceiling	
`ceiling(expresión)`	
Parámetros de entrada	▪ expresión numérica.
Saída	▪ Devolve o enteiro máis pequeno maior ou igual que a expresión numérica especificada, é dicir, fai unha aproximación por exceso.

Función floor:

Sintaxe función floor	
`floor(expresión)`	
Parámetros de entrada	▪ expresión numérica.
Saída	▪ Devolve o enteiro máis grande que sexa menor ou igual que a expresión numérica especificada, é dicir, fai unha aproximación por defecto

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 26, no editor de consultas do SSMS.

- **Consulta de exemplo 26:** Esta consulta aproxima por exceso e por defecto o número 134.873.

```
SELECT  ceiling(134.873) as num_aproximado_porexceso,
        floor(134.873) as num_aproximado_pordefecto;
```

Resultado da consulta do exemplo 26	
num_aproximado_porexceso	num_aproximado_pordefecto
135	134

Función power:

Sintaxe función power	
power(expresión, y)	
Parámetros de entrada	▪ expresión é de tipo float. ▪ y é a potencia á que se eleva a expresión.
Saída	▪ Devolve o valor de expresión elevado á potencia y, especificada. É dicir, obteremos expresión^{y}

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 27, no editor de consultas do SSMS.

- **Consulta de exemplo 27:** Nesta consulta realízanse varias potencias.

```
SELECT  power(2,2) as potencia1,
        power(2,3) as potencia2,
        power(7.2,2) as potencia3,
        power(7.2,3) as potencia4;
```

Resultado da consulta do exemplo 27			
potencia1	**potencia2**	**potencia3**	**potencia4**
4	8	51.8	373.2

Función sign:

Sintaxe función sign	
sign(expresión)	
Parámetros de entrada	▪ expresión numérica.
Saída	▪ Devolve +1 se o signo da expresión é positivo, 0 se expresión é 0 ou -1 se a expresión é negativa.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 28, no editor de consultas do SSMS.

- **Consulta de exemplo 28:** Verificación de se as expresións 18, -5, 0 e NULL, son positivas ou negativas.

```
SELECT  sign(18) as numero1,
        sign(-5) as numero2,
        sign(0) as numero3,
        sign(null) as numero4;
```

Resultado da consulta do exemplo 28			
numero1	**numero2**	**numero3**	**numero4**
1	-1	0	NULL

Función sqrt:

Sintaxe función sqrt	
sqrt(expresión)	
Parámetros de entrada	▪ expresión é de tipo float.
Saída	▪ Devolve a raíz cadrada de expresión.

Función square:

Sintaxe función square	
square(expresión)	
Parámetros de entrada	▪ expresión é de tipo float.
Saída	▪ Devolve o cadrado de expresión.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 29, no editor de consultas do SSMS.

- **Consulta de exemplo 29:** Esta devolve a raíz cadrada do número 49, e o cadrado do número 7.

```
SELECT  sqrt(49) as raiz_cadrada,
        square(7) as cadrado;
```

Resultado da consulta do exemplo 29	
raiz_cadrada	**cadrado**
7	49

6.6. Funcións do sistema

Función isnull:

Sintaxe función isnull	
isnull(expresión, valor_de_substitución)	
Parámetros de entrada	▪ expresión que se vai validar se é nula. Pode ser de calquera tipo. ▪ valor_de_substitución é polo que se vai substituír expresión cando teña valor null.
Saída	▪ Devolve o valor que teña expresión, salvo se é nulo, caso no que devolverá o valor_de_substitución.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 30, no editor de consultas do SSMS.

- **Consulta de exemplo 30:** Nome das actividades e as súas observacións. Nas actividades que non teñan observacións deberá aparecer a frase 'Sen observacións'.

```
SELECT  nome,
        isnull(observacions, 'Sen observacións') as observacions
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 30	
nome	**observacions**
TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ INICIACIÓN Á INFORMÁTICA	Precísase raqueta e 1 bote de pelotas Sen observacións Sen observacións Impartirase SW_libre

Función isnumeric:

Sintaxe función isnumeric	
`isnumeric(expresión)`	
Parámetros de entrada	▪ expresión que se vai evaluar se é numérica.
Saída	▪ Determina se unha expresión é un tipo numérico válido. Devolve 1 se expresión é de un tipo de dato numérico válido, e se non o era devolve 0.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 31, no editor de consultas do SSMS.

- **Consulta de exemplo 31:** Esta consulta de exemplo comproba se o contido das columnas seguintes da táboa SOCIO son ou non numéricas.

```
SELECT  isnumeric(numero) as numero,
        isnumeric(NIF) as NIF,
        isnumeric(cod_cota) as cod_cota
FROM SOCIO;
```

Resultado da consulta do exemplo 31		
numero	**NIF**	**cod_cota**
1 1 1 1	0 0 0 0	1 1 1 1

Función @@language:

Sintaxe función @@language	
`@@language`	
Saída	▪ Devolve o nome do idioma en uso.

Función @@max_connections:

Sintaxe función @@max_connections	
`@@max_connections`	
Saída	▪ Devolve o número máximo de conexións de usuario simultáneas que se permitan nunha instancia de SQL Server.

Función @@servername:

Sintaxe función @@servername	
@@servername	
Saída	▪ Devolve o nome do servidor local no que se executa SQL Server.

Función @@spid:

Sintaxe función @@spid	
@@spid	
Saída	▪ Devolve o identificador (Id) de sesión do proceso de usuario actual.

Función @@textsize:

Sintaxe función @@textsize	
@@textsize	
Saída	▪ Devolve o valor actual da opción TEXTSIZE, que especifica o tamaño dos datos varchar(max), nvarchar(max), varbinary(max), text, ntext e image devoltos por unha instrución SELECT.

Función @@version:

Sintaxe función @@version	
@@version	
Saída	▪ Devolve información do sistema e a compilación para a instalación actual de SQL Server.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 32, no editor de consultas do SSMS.

- **Consulta de exemplo 32:** Nesta consulta amosaremos os valores que devolven as funcións do sistema que acabamos de expoñer: @@language, @@max_connections, @@servername, @@spid, @@textsize e @@version.

```
SELECT  @@language as linguaxe,
        @@max_connections as limite_conexions,
        @@servername as nome_servidor,
        @@spid as id_sesion,
        @@textsize as tamano_texto,
        @@version as versionSQLServer;
```

Resultado da consulta do exemplo 32					
linguaxe	**limite_conexions**	**nome_servidor**	**id_sesion**	**tamano_texto**	**versionSQLServer**
Español	32767	PC01\INSTANCIA1	53	2147483647	Microsoft SQL Server 2014 - 12.0.2000.8 (X64) Feb 20 2014 20:04:26 Copyright (c) Microsoft Corporation Enterprise Edition (64-bit) on Windows NT 6.1 <X64> (Build 7601: Service Pack 1)

6.7. Funcións diversas

Funcións lóxicas

Función case:

Sintaxe función case	
Sintaxe1: CASE expresión WHEN valor THEN result1 [WHEN valor THEN result2 … WHEN valor THEN resultN] [ELSE outro_resultado] END **Sintaxe2:** CASE WHEN condición THEN result [WHEN condición THEN result2 … WHEN condición THEN resultN] [ELSE outro_resultado] END	
Parámetros de entrada	▪ expresión, é a expresión avaliada.
Saída	▪ Na primeira sintaxe avalía unha expresión e compáraa cunha lista de valores e devolve un dos resultados posibles. ▪ Na segunda sintaxe avalía cada unha das condicións e devolve o resultado da condición verdadeira.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 33, no editor de consultas do SSMS.

- **Consulta de exemplo 33:** Repetimos a consulta 30 pero empregando CASE.

```
SELECT  nome,
        CASE
            WHEN observacions is not null then observacions
            ELSE 'Sen observacións'
        END as observacions
FROM ACTIVIDADE;
```

Resultado da consulta do exemplo 33	
nome	**observacions**
TENIS PARA PRINCIPIANTES REPOSTARÍA XADREZ INICIACIÓN Á INFORMÁTICA	Precísase raqueta e 1 bote de pelotas Sen observacións Sen observacións Impartirase SW_libre

Función coalesce:

Sintaxe función coalesce	
`coalesce(expresión1 [,expresión2,...,expresiónN)`	
Parámetros de entrada	▪ as expresións poden ser de calquera tipo

Saída	▪ Avalía os argumentos en orde e devolve o valor actual da primeira expresión que non é NULL.
	▪ Se todos os argumentos pasados á función son NULL, coalesce devolve NULL.
	▪ A función COALESCE é un método abreviado de CASE. Internamente o servidor reescribe o código de COALESCE como o CASE seguinte:
	`CASE`
	`  WHEN expresión1 is not null THEN expresión1`
	`  WHEN expresión2 is not null THEN expresión2`
	`  …`
	`  ELSE expresiónN`
	`END`

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 34, no editor de consultas do SSMS.

- **Consulta de exemplo 34:** Nome, apelidos e teléfono de todos os socios. Deberá amosarse o teléfono1, se o ten, senón o teléfono2, e se non ten teléfono amosarase a frase 'Sen teléfono'. A vantaxe de coalesce con respecto a isnull, e que podemos poñer máis de 2 argumentos.

```
SELECT nome, ape1, ape2,
       coalesce(telefono1, telefono2, 'Sen teléfono') as telefono
FROM SOCIO;
```

Resultado da consulta do exemplo 34			
nome	**ape1**	**ape2**	**telefono**
MARÍA	GRAÑA	UMIA	981111112
MANUEL	SIEIRO	CAMPOS	981222223
JORGE	DEL CARMEN	LÉREZ	639333334
CARLA	VIEITO	GIL	Sen teléfono

Función iif:

Sintaxe función iif	
`iif(expresión, valor_true, valor_false)`	
Parámetros de entrada	▪ expresión booleana válida. ▪ valor_true é o valor que se vai devolver se a expresión que se avalía é verdadeira. ▪ valor_false é o valor que se vai devolver se a expresión que se avalía é falsa.
Saída	▪ Devolve un dos valores, dependendo de se a expresión se avalía como verdadeira (true) ou como falsa (false).

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 35, no editor de consultas do SSMS.

- **Consulta de exemplo 35:** NIF e cargo dos empregados. Se é profesor/a aparecerá PROFESORADO como cargo, senón ADMINISTRATIVO.

```
SELECT nif, iif(cargo='PRF', 'PROFESORADO', 'ADMINISTRATIVO') as cargo
FROM EMPREGADO;
```

- Resultado da consulta de exemplo 35:

Resultado da consulta do exemplo 35	
nif	**cargo**
11111111A 22222222B 33333333C 44444444D	PROFESORADO PROFESORADO PROFESORADO ADMINISTRATIVO

Funcións de tipos de datos

Función datalength:

Sintaxe función datalength	
`datalength(expresión)`	
Parámetros de entrada	▪ expresión é calquera tipo de dato.
Saída	▪ Devolve o número de bytes usados para representar calquera expresión.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 36, no editor de consultas do SSMS.

- **Consulta de exemplo 36:** Cantidade de bytes do número, nif e nome do socio número 1001.

```
SELECT  datalength(numero) as bytes_numero,
        datalength(nif) as bytes_nif,
        datalength(nome) as bytes_nome
FROM SOCIO
WHERE numero=1001;
```

Resultado da consulta do exemplo 36		
bytes_numero	**bytes_nif**	**bytes_nome**
4	9	6

Función @@identity:

Sintaxe función @@identity	
`@@identity`	
Saída	▪ Devolve o último valor de identidade xerado para calquera táboa na sesión actual.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 37, no editor de consultas do SSMS.

- **Consulta de exemplo 37:** Para comprobar o funcionamento desta función crearemos unha táboa temporal #t1 cun campo identity. Despois de engadir dúas filas na táboa, comprobamos coa función o último valor de identidade xerado na BD.

```
CREATE TABLE #t1
(codigo int identity(1,1),
 descricion varchar(10));
--Insertamos 2 filas.
insert into #t1 (descricion)
values('descri1'),
      ('descri2');
--Comprobamos o contido da táboa temporal.
SELECT * FROM #t1;
--Comprobamos o último valor de identidade xerado na BD
SELECT @@identity valor_identidade;
```

Resultado da consulta do exemplo 37
valor_identidade
2

Función ident_current:

Sintaxe función ident_current	
ident_current(táboa_vista)	
Parámetros de entrada	▪ táboa é o nome da táboa da que queremos coñecer o seu último valor de identidade.
Saída	▪ Devolve o último valor de identidade xerado para a táboa ou vista especificada.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 38, no editor de consultas do SSMS.

- **Consulta de exemplo 38:** Creamos unha nova táboa temporal #t2 e engadímoslle unha fila e despois comprobamos o valor de identidade da táboa #t1, e tamén o último valor de identidade engadido na BD. É importante poñer entre comiñas simples o nóme da táboa que lle pasamos entre parénteses á función.

```
CREATE TABLE #t2
(codigo int identity(1,1),
 descricion varchar(10));
--Insertamos 2 filas.
insert into #t2 (descricion)
values('descri1');
--Comprobamos o contido da táboa temporal.
SELECT * FROM #t2;
--Comprobamos o último valor de identidade xerado na BD
SELECT ident_current('#t1') as "valor_identidade_#t1",
       @@identity valor_identidadeBD;
```

Resultado da consulta do exemplo 38	
valor_identidade_#t1	**valor_identidadeBD**
2	1

Funciones de metadatos

Función col_name:

Sintaxe función col_name	
`col_name(táboa_id, columna_id)`	
Parámetros de entrada	▪ táboa_id é o número de identificación da táboa que contén á columna. ▪ columna_id é o número de identificación da columna, da posición que ocupa na táboa.
Saída	▪ Devolve o nome dunha columna a partires do número de identificación da táboa e do número de identificación de columna pasados á función.

Función db_id:

Sintaxe función db_id	
`db_id(nome_bd)`	
Parámetros de entrada	▪ nome_bd é o nome da BD da que queremos coñecer o identificador.
Saída	▪ Devolve o número identificador da base de datos.

Función db_name:

Sintaxe función db_name	
`db_name(id_bd)`	
Parámetros de entrada	▪ id_db é o número identificador da bd da que queremos coñecer o nome.
Saída	▪ Devolve o nome da BD que se corresponde co identificador pasado á función.

Función object_id:

Sintaxe función object_id	
`object_id(nome_obxecto[, tipo_obxecto])`	
Parámetros de entrada	▪ nome_obxecto é o obxecto do que queremos coñecer o identificador. ▪ tipo_obxecto é un tipo de obxecto de BD.
Saída	▪ Devolve o número de identificación do obxecto de base de datos.

Función object_name:

Sintaxe función object_name	
`object_name(id_obxecto[, database_id])`	
Parámetros de entrada	▪ id_obxecto é o identificador do obxecto do que queremos saber o nome. ▪ database_id é o identificador da bd onde se vai buscar o obxecto.
Saída	▪ Devolve o nome do obxecto do identificador pasado, da bd especificada.

Función parsename:

Sintaxe función parsename	
`parsename(nome_obxecto, parte_obxecto)`	
Parámetros de entrada	▪ nome_obxecto é o nome do obxecto do que se desexa recuperar a parte especificada. ▪ parte_obxecto é a parte do obxecto de bd que se vai devolver. Pode ter un dos valores seguintes: – 1 = Nome do obxecto – 2 = Nome do esquema – 3 = Nome da base de datos – 4 = Nome do servidor
Saída	▪ Devolve a parte especificada dun nome de obxecto.

A continuación realizarase, executarase e comprobarase o resultado da consulta de exemplo 39, no editor de consultas do SSMS.

- **Consulta de exemplo 39:** Nesta consulta amosamos o uso das funcións de metadatos aquí explicadas.

```
SELECT col_name(object_id('SOCIO'),5) as Nome_columna,
       db_id('SOCIEDADE_CULTURAL') as identificadorBD,
       db_name(8) as nomeBD,
       object_id('SOCIO') as id_obxecto,
       object_name(341576255) as nome_obxecto,
       parsename('SOCIEDADE_CULTURAL..AULA',3) as nome_bd;
```

Resultado da consulta do exemplo 39

col_name	identificadorBD	nomeBD	id_obxecto	nome_obxecto	nome_bd
ape1	8	SOCIEDADE_CULTURAL	341576255	SOCIO	SOCIEDADE_CULTURAL

6.8. Tarefa de consultas con funcións integradas no xestor

Unha vez copiados, executados e probados os exemplos das explicacións, realizarase o código T-SQL adecuado para obter a información que se pide en cada unha das consultas propostas na BD EMPRESA. Aínda que non se indique, nos resultados todas as columnas terán un nome que identifique correctamente a información que amosan.

- Consultas propostas na **BD EMPRESA**.
  - **Proposta 1.** Desexamos coñecer o código ASCII da vogal E. Na consulta deberás devolver nunha columna a vogal en maiúscula, e nunha segunda o código ASCII que lle corresponde.
  - **Proposta 2.** Consulta que devolve o carácter que lle corresponde aos seguintes códigos ASCII: 70, 80, 90.
  - **Proposta 3.** Queremos obter unha listaxe que en cada liña teña o seguinte texto: *O empregado con nome e apelidos X ten que acadar unha cota de vendas anual de Y*. Sendo X o nome e os apelidos do empregado, e Y a cota de vendas. É importante fixarse no segundo apelido. A listaxe terá por título *Empregados e cotas.*
  - **Proposta 4.** Consulta que devolva as datas nas que se contrataron empregados. O formato das diferentes datas será dd-mm-aaaa e o nome da columna *Datas de contratación.*
  - **Proposta 5.** Queremos obter un nome abreviado das sucursais. Ese nome comporase polos tres primeiros caracteres da cidade, os dous últimos da rexión e separado por un guión baixo, o número de caracteres do nome da cidade.
  - **Proposta 6.** Queremos obter un nome abreviado dos produtos. Ese nome comporase polo segundo carácter do código do fabricante en minúscula, máis o terceiro, cuarto, quinto e sexto da descrición do produto. Nunha primeira columna o código aparecerá en minúsculas, e nunha segunda en maiúsculas.
  - **Proposta 7.** Listaxe cos nomes dos empregados co formato *ape1 ape2, nome*. Se algún empregado non ten segundo apelido, por exemplo Susanne Smith, no resultado aparecerá *Smith, Sussane*, sen espazos antes da coma.
  - **Proposta 8.** Queremos amosar os distintos títulos dos nosos empregados en castelán, e para iso deberemos substituír a palabra *VENDAS* por *VENTAS.*
  - **Proposta 9.** Consulta que devolva a seguinte información de tempo en distintas columnas co nome adecuado cada unha:
    - data e hora actuais sen axuste de zona horaria,
    - data e hora actuais con axuste de zona horaria,
    - mes actual en número,
    - mes actual en número *(emprega unha función diferente á da anterior columna)*,
    - ano actual,
    - mes actual en nome,

  - hora actual,
  - nanosegundos actuais.
- **Proposta 10.** Listaxe que devolva o nome de todos os empregados (nome, ape1, ape2), a data de contrato, e nunha última columna a data de contrato adiantada un ano. O formato das dúas datas será dd/mm/aaaa (con barras).
- **Proposta 11.** Listaxe que devolva o número de cada pedido coa data de pedido. Nunha terceira columna deberá aparecer a mesma data de pedido pero retrasada dous meses. O formato das dúas datas será dd-mm-aaaa (con guións).
- **Proposta 12.** Listaxe que devolva o nome e apelidos (nome, ape1, ape2) de cada empregado, a data de contrato e o número de anos que hai que leva traballando na empresa cada un deles.
- **Proposta 13.** Consulta que devolva a descrición de cada produto co seu prezo nunha segunda columna, e ademais deberán amosarse en columnas diferentes:
  - o prezo como un enteiro aproximado por defecto,
  - o prezo como un enteiro aproximado por exceso,
  - a raíz cadrada do prezo,
  - o cadrado do prezo, e,
  - o cubo do prezo.
- **Proposta 14.** Repite a consulta anterior pero agora só amosaremos a descrición, o prezo e a raíz cadrada, pero a raíz cadrada deberá amosarse con como moito 4 cifras na parte enteira e 3 na decimal.
- **Proposta 15.** Consulta que devolva a seguinte información do servidor no que está a nosa instancia de SQL Server: idioma, número máximo de conexións permitidas, nome do servidor e da instancia e versión do xestor.
- **Proposta 16.** Consulta que amose a descrición do produto e as súas existencias. Nunha terceira columna de nome estado_existencias amosarase o seguinte:
  - Se o número de existencias é superior a 20 aparecerá a palabra *Suficientes*.
  - Se o número de existencias é menor ou igual a 20 aparecerá *Insuficientes*.

  Esta consulta deberás resolvela de dous xeitos posibles, en dúas consultas diferentes, empregando dúas funcións lóxicas distintas.

7. Consultas compostas

O obxectivo deste apartado é aprender a deseñar consultas compostas por outras consultas.

Usaremos o SXBD (Sistema Xestor de Base de Datos) MS SQLServer, polo que a sintaxe que se explicará será a da linguaxe Transact-SQL (T-SQL en adiante), propia do xestor.

Explicarase a sintaxe T-SQL e uso das consultas compostas máis usadas que son:

- a unión.
- A intersección.
- A diferenza.

En SQL Server utilizaremos os operadores de conxuntos UNION, INTERSECT e EXCEPT para realizar consultas compostas de unión, intersección e diferenza,respectivamente.

Denomínanse operadores de conxuntos porque combinan os resultados doutras consultas nun só conxunto de resultados.

7.1. Unión de consultas

A unión de consultas, consiste en obter un conxunto de filas que proveñen de varias consultas. No resultado, a menos que o indiquemos explicitamente, non aparecerán as filas repetidas.

En Transact-SQL a palabra reservada para realizar a unión entre consultas é UNION.

Para que se poida realizar unha unión entre consultas é necesario que as dúas consultas cumpran as seguintes condicións:

- ter o mesmo número de columnas,
- que as columnas teñan o mesmo tipo ou tipos compatibles. É dicir, a primeira columna da primeira consulta deberá ter un tipo compatible coa primeira columna da segunda consulta; e así sucesivamente con cada par de columnas.

Sintaxe union

```
SELECT col1 [,..., colN]
FROM ...
[WHERE...]
[GROUP BY...]
[HAVING ...]
UNION [ALL]
SELECT col1 [,..., colN]
FROM ...
[WHERE...]
[GROUP BY...]
[HAVING ...]
[UNION [ALL]]
```

```
SELECT col1 [,..., colN]
FROM ...
[WHERE...]
[GROUP BY...]
[HAVING ...]
[ORDER BY ...] -- Un único order by
```

- **ALL:** Cando engadimos a palabra ALL despois de UNION estamos indicando que queremos que no resultado aparezan todas as filas, repetidas ou non.
- **ORDER BY:** A cláusula ORDER BY só se poderá incluír ao final da última consulta, non en cada unha delas. Nesta cláusula, sempre se fará referencia ao nome dos campos da primeira consulta, ou ben indicaremos a posición. Como xa dixemos sempre se entenderá mellor a consulta se indicamos o nome e non a posición.

A continuación realizaranse, executaranse e comprobaranse os resultados das consultas de exemplo 1 e 2, no editor de consultas do SSMS.

- **Consulta de exemplo 1:** Esta consulta de exemplo amosa o nome completo dos socios morosos, que son aqueles que deben a cota anual ou algunha actividade. O resultado aparecerá ordenado por orde alfabética de apelidos e nome:

```
SELECT ape1, ape2, nome
FROM SOCIO
WHERE abonada ='N'
UNION
SELECT s.ape1, s.ape2, s.nome
FROM SOCIO s, SOCIO_REALIZA_ACTI r
WHERE r.pagada ='N' AND
      s.numero = r.num_socio
ORDER BY ape1, ape2, nome;
```

Resultado da consulta do exemplo 1		
ape1	**ape2**	**nome**
DEL CARMEN SIEIRO	LÉREZ CAMPOS	JORGE MANUEL

- **Consulta de exemplo 2:** Repetimos a consulta de exemplo 1 pero engadindo a palabra ALL no UNION.

```
--Engadimos ALL
SELECT ape1, ape2, nome
FROM SOCIO
WHERE abonada ='N'
UNION ALL
SELECT s.ape1, s.ape2, s.nome
FROM SOCIO s, SOCIO_REALIZA_ACTI r
WHERE r.pagada ='N' AND
      s.numero = r.num_socio
ORDER BY ape1, ape2, nome;
```

Resultado da consulta do exemplo 2		
ape1	**ape2**	**nome**
DEL CARMEN SIEIRO SIEIRO	LÉREZ CAMPOS CAMPOS	JORGE MANUEL MANUEL

Vemos como agora aparece dúas veces *Sieiro Campos Manuel*. Se executamos as consultas de xeito independente entenderémolo. Este socio debe a cota anual pero tamén debe unha actividade, polo tanto aparece no resultado das dúas consultas que unimos. Se non utilizamos ALL as filas repetidas descártanse, pero ao poñelo amósanse no resultado.

7.2. Intersección de consultas

A intersección de consultas, consiste en obter as filas que teñen en común dúas consultas, só de dúas, non como no caso da unión na que se permite unir máis de dúas consultas.

En Transact-SQL a palabra reservada para realizar a intersección de consultas é INTERSECT.

Para que se poida realizar unha intersección de consultas é necesario que as dúas consultas cumpran as mesmas condicións que no caso da unión:

- ter o mesmo número de columnas, e,
- que as columnas teñan o mesmo tipo ou tipos compatibles.

Sintaxe intersect

```
SELECT col1 [,..., colN]
FROM ...
[WHERE...]
[GROUP BY...]
[HAVING ...]
INTERSECT
SELECT col1 [,..., colN]
FROM ...
[WHERE...]
[GROUP BY...]
[HAVING ...]
[ORDER BY ...]
```

- **ORDER BY:** A cláusula ORDER BY só se poderá incluír ao final da segunda consulta, non en cada unha delas.

- **Consulta de exemplo 3:** Repetimos a consulta de exemplo 1, pero agora cambiamos a palabra reservada *union* por *intersect* e comprobamos o resultado.

```
SELECT ape1, ape2, nome
FROM SOCIO
WHERE abonada ='N'
INTERSECT
SELECT s.ape1, s.ape2, s.nome
FROM SOCIO s, SOCIO_REALIZA_ACTI r
WHERE r.pagada ='N' AND
      s.numero = r.num_socio
ORDER BY ape1, ape2, nome;
```

Como era de esperar no resultado aparece só *Sieiro Campos Manuel*, porque cos datos actuais é o único que ademais de deber a cota anual tamén debe algunha actividade.

Resultado da consulta do exemplo 3		
ape1	**ape2**	**nome**
SIEIRO	CAMPOS	MANUEL

- **Consulta de exemplo 4:** Nesta consulta obtemos o nome, data de inicio e prezo daquelas actividades non gratuítas do mes de febreiro de calquera ano. Para obter o mes dunha data utilizamos a función integrada no xestor month. O resultado deberá aparecer ordenado por data de inicio, de máis antiga a máis recente.

```
SELECT nome, data_ini, prezo
FROM ACTIVIDADE
WHERE prezo>0
INTERSECT
SELECT nome, data_ini, prezo
FROM ACTIVIDADE
WHERE month(data_ini)=2
ORDER by data_ini;
```

No resultado aparecen as actividades que cumpren as condicións das dúas consultas. Evidentemente esta consulta podería terse resolto cun único SELECT coas dúas condición nun único WHERE, separadas por AND, pero o que se pretende con este exemplo é só ilustrar o uso da intersección.

Resultado da consulta do exemplo 4		
nome	**data_ini**	**prezo**
TENIS PARA PRINCIPIANTES REPOSTARÍA	2014-02-10 16:00:00.000 2015-02-15 17:00:00.000	301.55 50.00

7.3. Diferenza de consultas

A diferenza de consultas consiste en obter as filas que aparecen na primeira consulta da diferenza pero non na segunda. Igual que no caso da intersección só afecta a dúas consultas.

En Transact-SQL a palabra reservada para realizar a intersección de consultas é EXCEPT.

Para que se poida realizar unha diferenza de consultas é necesario que as dúas consultas cumpran as mesmas condicións que no caso da unión e da intersección:

- ter o mesmo número de columnas, e,
- que as columnas teñan o mesmo tipo ou tipos compatibles.

Sintaxe except

```
SELECT col1 [,..., colN]
FROM ...
[WHERE...]
[GROUP BY...]
[HAVING ...]
EXCEPT
SELECT col1 [,..., colN]
FROM ...
[WHERE...]
[GROUP BY...]
[HAVING ...]
[ORDER BY ...]
```

- **ORDER BY:** A cláusula ORDER BY só se poderá incluír ao final da segunda consulta, non en cada unha delas.

- **Consulta de exemplo 5:** Repetimos a consulta de exemplo 1, pero agora cambiamos a palabra reservada *union* por *except* e comprobamos o resultado.

```
SELECT ape1, ape2, nome
FROM SOCIO
WHERE abonada ='N'
EXCEPT
SELECT s.ape1, s.ape2, s.nome
FROM SOCIO s, SOCIO_REALIZA_ACTI r
WHERE r.pagada ='N' AND
      s.numero = r.num_socio
ORDER BY ape1, ape2, nome;
```

No resultado non aparece ningún socio, xa que o único socio que debe a cota anual, é dicir aparece no resultado do primeiro SELECT, tamén debe algunha actividade, polo que no resultado final non aparecerá ningún socio.

Resultado da consulta do exemplo 5

ape1	ape2	nome

- **Consulta de exemplo 6:** Repetimos agora a consulta do exemplo 4 cambiando a palabra intersect por except.

```
SELECT nome, data_ini, prezo
FROM ACTIVIDADE
WHERE prezo>0
EXCEPT
SELECT nome, data_ini, prezo
FROM ACTIVIDADE
WHERE month(data_ini)=2
ORDER by data_ini;
```

No resultado aparecen as actividades que cumpren a primeira condición pero non a segunda. Evidentemente esta consulta podería terse resolto cun único SELECT coas dúas condición nun único WHERE separadas por AND e negando a segunda, pero o que se pretende con este exemplo é ilustrar o uso da diferenza.

Resultado da consulta do exemplo 6		
nome	**data_ini**	**prezo**
XADREZ	2014-03-20 16:30:00.000	80.00

7.4. Tarefa de consultas compostas

Unha vez copiados, executados e probados os exemplos das explicacións, realizarase o código T-SQL adecuado para obter a información que se pide en cada unha das consultas propostas na BD EMPRESA. Aínda que non se indique, nos resultados todas as columnas terán un nome que identifique correctamente a información que amosan.

- Consultas propostas na **BD EMPRESA**.
  - **Proposta 1.** Empregando unha consulta composta realizar unha listaxe do código do fabricante e identificador daqueles produtos con prezo superior a 60€ ou que teñan pedidos de cantidade inferior a 5 unidades. O resultado aparecerá ordenado por fabricante e para o mesmo fabricante por produto.
  - **Proposta 2.** Empregando unha consulta composta amosar os código dos empregados que non fixeron pedidos. Deberán aparecer primeiro os empregados con código maior.
  - **Proposta 3.** Empregando unha consulta composta amosar o código dos clientes que fixeron pedidos e con límite de crédito maior ou igual a 40000. Usa unha diferenza para resolver esta consulta.
  - **Proposta 4.** Empregando unha consulta composta amosar os código dos clientes que fixeron pedidos e con límite de crédito maior ou igual a 40000. Usa unha intersección para resolver esta consulta. Ordena o resultado por código de cliente en orde ascendente.
  - **Proposta 5.** Empregando unha consulta composta amosar o código dos empregados que son directores dalgunha sucursal ou que teñen unha cota de vendas superior a 250000€.
  - Debes propoñer dúas solucións:
    - na primeira só pode aparecer unha vez cada empregado no resultado, e,
    - na segunda se un empregado é director dunha sucursal e ademais ten unha cota superior a 250000€, aparecerá no resultado máis dunha vez.

8. Consultas complexas optimizadas

Despois de estudar as consultas simples e consultas resumo, con combinacións, con subconsultas, con funcións integradas no xestor e as consultas compostas, imos aprender nesta actividade a realizar consultas que combinen estes tipos de xeito optimizado.

8.1. Consultas complexas

Para a organización da aprendizaxe das consultas de recuperación da información das bases de datos, denominaranse consultas complexas a aquelas que combinan algúns ou todos os tipos de consultas que vimos nos apartados anteriores:

- Consultas simples.
- Consultas resumo.
- Combinacións internas e externas..
- Consultas con subconsultas.
- Consultas con funcións integradas no xestor.
- Consultas compostas.

Nesta actividade analizaranse:

- enunciados de consultas para aprender a resolvelos.
- Código de consultas T-SQL para detectar posibles erros.
- Código de consultas T-SQL para detectar problemas de optimización.

As consultas de estudo realizaranse sobre as bases de datos SOCIEDADE_CULTURAL e EMPRESA, bases de datos de traballo que xa se usaron nos apartados anteriores.

8.1.1. Enunciados de consultas complexas. Estudo e resolución

A continuación analizaranse enunciados de consultas complexas, codificaranse en T-SQL, executaranse e comprobarase o resultado das mesmas, no editor de consultas do SSMS.

GO. Separador de lotes. Determina o final dun conxunto (lote) de instrucións que se executan xuntas. Usarase despois da instrución de selección da BD para separala do código da consulta.

- **Consulta de exemplo 1:** BD SOCIEDADE_CULTURAL. Listaxe da duración en días de tódalas actividades. Aparecerá o nome da actividade e nunha segunda columna de nome *Duracion_dias* o número de días. O resultado aparecerá por orde alfabética do nome da actividade.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Non é necesaria.

Táboas (FROM)	A información buscada está na táboa ACTIVIDADE, daquela poñerase esa táboa no FROM.
Columnas (SELECT)	O nome e a duración en días. O nome está no atributo *nome*, pero a duración en días terase que calcular.
Combinacións (*joins*)	Non son necesarios porque só hai unha táboa no FROM.
Funcións	Na táboa ACTIVIDADE non existe ningún campo co número de días que dura cada unha delas. A información da que dispoñemos son as datas de inicio e fin. Aplicando a función *datediff* a esas dúas datas obteremos os días de duración.
Filtros de filas (WHERE)	Pídense tódalas actividades, polo que non hai que limitar o resultado por ningunha condición de enunciado.
Agrupamentos (GROUP BY)	Non se pide nin contar filas, nin sumar, nin calcular a media, nin mínimos nin máximos de columnas.
Filtro de grupos (HAVING)	Se non hai grupos, non hai que filtralos.
Ordenación (ORDER BY)	Ordenaremos pola columna nome alfabeticamente, que é a ordenación por defecto.
Alias	**De columna:** A segunda columna é calculada e hai que utilizar un alias para que non apareza a frase *Sin nombre de columna*. Neste caso o nome proporciónase no enunciado, *Duracion_dias.* **De táboa:** Non é necesario porque no FROM só hai unha táboa.

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT nome, datediff(day,data_ini, data_fin) as Duracion_dias
FROM ACTIVIDADE
ORDER BY nome;
```

Resultado da consulta do exemplo 1	
nome	**Duracion_dias**
INICIACIÓN Á INFORMÁTICA REPOSTARÍA TENIS PARA PRINCIPIANTES XADREZ	61 28 242 92

- **Consulta de exemplo 2:** BD SOCIEDADE_CULTURAL. Nome completo dos profesores e profesoras nunha columna *Nome_completo,* co formato *ape1 ape2, nome*, en maiúsculas. Nunha segunda columna deberá aparecer o nome da súa especialidade en minúsculas. Os datos deben aparecer en orde alfabética do nome completo do profesorado e no caso de que coincidan os nomes de varios docentes, ordenarase en orde alfabética inversa da especialidade (da letra Z á letra A).

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Non é necesaria.
Táboas (FROM)	A información buscada está nas táboas EMPREGADO (ape1, ape2 e nome) e PROFESORADO (especialidade), daquela poñeranse esas dúas táboas no FROM.
Columnas (SELECT)	O nome, primeiro e segundo apelido e a especialidade.

Combinacións (*joins*)	Farase *join* entre o número de empregado (clave primaria, CP en adiante) de EMPREGADO e o *num_prof* (clave foránea, CF en adiante) de PROFESORADO.
Funcións	Hai que converter en maiúsculas a primeira columna con upper e en minúsculas a segunda con *lower*. Ademais necesitamos substituír os posibles nulos do segundo apelido con *isnull*. Supoñamos agora que nun dos resultados houbese un segundo apelido nulo, despois de facer *isnull* aparecería un espazo en branco antes da coma, polo que hai que eliminar ese espazo en branco coa función *rtrim*.
Filtros de filas (WHERE)	Poderiamos pensar en filtrar polo cargo para quedarnos co profesorado unicamente, pero non é necesario porque ao facer o *join* coa táboa de PROFESORADO xa desaparecen do resultado todos os que non están nesta táboa, é dicir, os que non sexan profesores ou profesoras. Daquela non precisamos de ningunha condición no WHERE.
Agrupamentos (GROUP BY)	Non se pide nin contar filas, nin sumar, nin calcular a media, nin mínimos nin máximos de columnas.
Filtro de grupos (HAVING)	Se non hai grupos, non hai que filtralos.
Ordenación (ORDER BY)	Ordenaremos pola columna do nome composto alfabeticamente e en orde descendente da especialidade, xa que nos piden orde alfabética inversa.
Alias	**De columna:** A primeira columna terá por alias *Nome_completo* e a segunda especialidade, porque ao aplicarlle a función *lower* non terá nome que a identifique.
	De táboa: Ao haber máis dunha táboa no FROM usaranse sempre alias para evitar posibles ambigüidades con nomes de campos comúns a diferentes táboas, e tamén para facilitar a comprensión do código da consulta.

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
--Coa sintaxe da condición de combinación no WHERE
SELECT upper(rtrim(e.ape1+' '+isnull(e.ape2,''))+', '+e.nome) as Nome_completo,
       lower(p.especialidade) as especialidade
FROM EMPREGADO e, PROFESORADO p
WHERE e.numero=p.num_prof
ORDER BY Nome_completo, especialidade DESC;
--Coa sintaxe da condición de combinación no FROM
SELECT upper(rtrim(e.ape1+' '+isnull(e.ape2,''))+', '+e.nome) as Nome_completo,
       lower(p.especialidade) as especialidade
FROM EMPREGADO e INNER JOIN PROFESORADO p
     ON e.numero=p.num_prof
ORDER BY Nome_completo, especialidade DESC;
```

Resultado da consulta do exemplo 2	
Nome_completo	**especialidade**
GARCÍA PÉREZ, MARÍA POSE VARELA, JUANA REGO PENA, CARLOS	deportes informática cociña

- **Consulta de exemplo 3:** BD EMPRESA. Unidades totais de produtos compradas polo cliente APPS INFOR, SL. A columna debe chamarse *Unidades_totais*.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Non é necesaria.
Táboas (FROM)	As unidades compradas están na táboa PEDIDO. O nome do cliente está en CLIENTE. As táboas do FROM serán PEDIDO e CLIENTE.
Columnas (SELECT)	Haberá unha única columna cos unidades totais.
Combinacións (*joins*)	Farase *join* entre o número de cliente (CP) de CLIENTE e o *num_cliente* (CF) de PEDIDO.
Funcións	Precísase a función de agregado *sum* que sume as unidades de produtos.
Filtros de filas (WHERE)	Temos que quedarnos só cos pedidos do cliente APPS INFOR, polo que filtraremos por ese nome.
Agrupamentos (GROUP BY)	Non é necesario agrupar porque todas as filas resultantes pertencen ao mesmo cliente.
Filtro de grupos (HAVING)	Se non hai grupos, non hai que filtralos.
Ordenación (ORDER BY)	A consulta só devolve un valor e non é preciso ordenar.
Alias	**De columna:** A columna do *sum* chamarase Unidades_totais.
	De táboa: Ao haber máis dunha táboa no FROM usaranse sempre alias para evitar posibles ambigüidades con nomes de campos comúns a diferentes táboas, e tamén para facilitar a comprensión do código da consulta.

```
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT sum(p.cantidade) as Unidades_totais
FROM CLIENTE c, PEDIDO p
WHERE c.nome='APPS INFOR, SL' AND
      c.numero=p.num_cliente;
```

Resultado da consulta do exemplo 3
Unidades_totais
34

- **Consulta de exemplo 4:** BD EMPRESA. Unidades totais de produtos compradas por cada cliente. No resultado aparecerá o nome da empresa cliente e nunha segunda columna as unidades totais. A segunda columna chamarase *Unidades_totais*. No resultado deben aparecer primeiro os clientes que máis unidades mercaron.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Non é necesaria.
Táboas (FROM)	As unidades compradas están na táboa PEDIDO. Os nomes do clientes está en CLIENTE. As táboas do FROM serán PEDIDO e CLIENTE.
Columnas (SELECT)	Haberá dúas columnas, o nome dos clientes en CLIENTE e as unidades totais.
Combinacións (*joins*)	Farase *join* entre o número de cliente (CP) de CLIENTE e o *num_cliente* (CF) de PEDIDO.

Funcións	Precísase a función de agregado *sum* que sume as unidades de produtos.
Filtros de filas (WHERE)	Non hai que limitar o resultado por ningunha condición de enunciado.
Agrupamentos (GROUP BY)	Será necesario agrupar porque temos que devolver un resultado por cliente. Teremos que xuntar as filas dos pedidos de cada cliente en diferentes grupos, así a suma das unidades calcularase de cada grupo, é dicir, de cada cliente.
Filtro de grupos (HAVING)	Non se nos esixe eliminar ningún grupo do resultado.
Ordenación (ORDER BY)	No resultado deben aparecer primeiro os clientes que máis unidades mercaron, polo que hai que ordenar pola columna de Unidades_totais en orde descendente para que primeiro apareza no resultado os clientes que mercaron máis unidades.
Alias	**De columna:** A columna do *sum* chamarase Unidades_totais.
	De táboa: Ao haber máis dunha táboa no FROM usaranse sempre alias para evitar posibles ambigüidades con nomes de campos comúns a diferentes táboas, e tamén para facilitar a comprensión do código da consulta.

```
--Seleccionamos a BD
USE EMPRESA;
GO
--Solución 1 con ORDER BY pola expresión
SELECT c.nome, sum(p.cantidade) as Unidades_totais
FROM CLIENTE c, PEDIDO p
WHERE c.numero=p.num_cliente
GROUP BY c.numero, c.nome
ORDER BY sum(p.cantidade) DESC;
--Solución 2 con ORDER BY polo nome
SELECT c.nome, sum(p.cantidade) as Unidades_totais
FROM CLIENTE c, PEDIDO p
WHERE c.numero=p.num_cliente
GROUP BY c.numero, c.nome
ORDER BY Unidades_totais DESC;
```

Cando se agrupa hai que escoller ben o campo, sempre mellor claves primarias e non nomes. Neste exemplo se agrupamos só polo nome a consulta funciona igual, o erro produciríase se dous ou máis clientes tivesen o mesmo nome, xa que nese caso formaríase un só grupo cos pedidos de todos os clientes que comparten nome. O resultado non sería correcto pois sumaríanse as unidades dos clientes que se chaman igual como se fosen un só.

Resultado da consulta do exemplo 4

nome	Unidades_totais
PC MAX	99
PC OK	65
APPS INFOR, SL	34
INFOR MAX	30
PeM INFORMÁTICA	26
TODO HW	21
PIP INFORMÁTICA	19
O TEU PC	17
HW & SW OK	13
MERCADO PC	11
TODO INFOR	7

PC CAIXA, SL	6
SENRA PC	5
PC POR PEZAS, SL	2

- **Consulta de exemplo 5:** BD EMPRESA. Nesta consulta pídese o mesmo que na consulta de exemplo 4, pero agora debemos amosar só aqueles clientes que mercaron máis de 20 unidades.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*. O único que cambia con respecto a análise da consulta de exemplo 4, é que agora hai que eliminar grupos.

Análise do enunciado	
Composición de consultas	Non é necesaria.
Táboas (FROM)	As unidades compradas están na táboa PEDIDO. Os nomes do clientes está en CLIENTE. As táboas do FROM serán PEDIDO e CLIENTE.
Columnas (SELECT)	Haberá dúas columnas, o nome dos clientes en CLIENTE e as unidades totais.
Combinacións (*joins*)	Farase *join* entre o número de cliente (CP) de CLIENTE e o *num_cliente* (CF) de PEDIDO.
Funcións	Precísase a función de agregado *sum* que sume as unidades dos pedidos.
Filtros de filas (WHERE)	Non hai que limitar o resultado por ningunha condición de enunciado.
Agrupamentos (GROUP BY)	É necesario agrupar porque temos que devolver un resultado por cliente. Teremos que xuntar as filas dos pedidos de cada cliente en diferentes grupos, así a suma das unidades calcularase de cada grupo, é dicir, de cada cliente.
Filtro de grupos (HAVING)	Só aparecerán os clientes con más de 20 unidades mercadas, así que teremos que quedarnos cos grupos que verifiquen esa condición. As condicións que afectan aos grupos fanse sempre despois de agrupar na cláusula HAVING. Non se pode filtrar no WHERE posto que aínda non estarían os grupos feitos e polo tanto as sumas das unidades de cada cliente calculadas. Pódese comprobar na solución con erro proposta despois da análise.
Ordenación (ORDER BY)	No resultado deben aparecer primeiro os clientes que máis unidades mercaron, polo que hai que ordenar pola columna de Unidades_totais en orde descendente para que primeiro apareza no resultado os clientes que mercaron máis unidades.
Alias	**De columna:** A columna do *sum* chamarase *Unidades_totais*.
	De táboa: Ao haber máis dunha táboa no FROM usaranse sempre alias para evitar posibles ambigüidades con nomes de campos comúns a diferentes táboas, e tamén para facilitar a comprensión do código da consulta.

```
--Seleccionamos a BD
USE EMPRESA;
GO
--Solución 1 ERRÓNEA
SELECT c.nome, sum(p.cantidade) as Unidades_totais
FROM CLIENTE c, PEDIDO p
WHERE   sum(p.cantidade)>20 AND
        c.numero=p.num_cliente
GROUP BY c.numero, c.nome
ORDER BY Unidades_totais DESC;
--Solución 2 CORRECTA
SELECT c.nome, sum(p.cantidade) as Unidades_totais
FROM CLIENTE c, PEDIDO p
WHERE c.numero=p.num_cliente
GROUP BY c.numero, c.nome
HAVING sum(p.cantidade)>20
ORDER BY Unidades_totais DESC;
```

Proponse unha primeira solución na que se intenta filtrar no WHERE. A consulta xera un erro xa que no WHERE non se poden incluír funcións colectivas como *sum*. O correcto é filtrar grupos no HAVING, unha vez están feitos.

Resultado da consulta do exemplo 5	
nome	**Unidades_totais**
PC MAX	99
PC OK	65
APPS INFOR, SL	34
INFOR MAX	30
PeM INFORMÁTICA	26
TODO HW	21

- **Consulta de exemplo 6:** BD SOCIEDADE_CULTURAL. Nome de todas aquelas actividades que non foron pagadas por algún socio con cota de nome FAMILIAR. O resultado aparecerá nunha columna de nome *actividade_adeudada*.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Non é necesaria.
Táboas (FROM)	O nome das actividades atópase na táboa ACTIVIDADE. Se se pagaron ou non aparece en SOCIO_REALIZA_ACTI. O nome da cota FAMILIAR está en COTA. Para poder saber que socios pagan e que cota teñen asignada debemos consultar tamén a táboa SOCIO.
Columnas (SELECT)	Haberá unha columna, o nome das actividades debidas.
Combinacións (*joins*)	Hai catro táboas así que como mínimo debe haber tres condicións de *join*, unha menos que táboas. Farase *join* entre o identificador (CP) da ACTIVIDADE e o *id_actividade* (CF) de SOCIO_REALIZA_ACTI. Entre o *num_socio* (CF) de SOCIO_REALIZA_ACTI e numero (CP) de SOCIO. Entre o *cod_cota* (CF) de SOCIO e o *codigo* (CP) de COTA.

Funcións	Non se precisan.
Filtros de filas (WHERE)	Hai que quedarse só coas actividades non pagadas. Só hai que escoller os socios con cota 'FAMILIAR'.
Agrupamentos (GROUP BY)	Non se pide nin contar filas, nin sumar, nin calcular a media, nin mínimos nin máximos de columnas.
Filtro de grupos (HAVING)	Se non hai grupos, non hai que filtralos.
Ordenación (ORDER BY)	Non é preciso ordenar.
Alias	**De columna:** A columna da actividade chamarase *actividade_adecuada*.
	De táboa: Ao haber máis dunha táboa no FROM usaranse sempre alias para evitar posibles ambigüidades con nomes de campos comúns a diferentes táboas, e tamén para facilitar a comprensión do código da consulta.

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT a.nome as actividade_adeudada
FROM ACTIVIDADE a, SOCIO_REALIZA_ACTI sr,
     SOCIO s, COTA c
WHERE    sr.pagada='N' AND
         c.nome='FAMILIAR' AND
         a.identificador=sr.id_actividade AND
         sr.num_socio=s.numero AND
         s.cod_cota=c.codigo;
```

Resultado da consulta do exemplo 6
actividade_adeudada
TENIS PARA PRINCIPIANTES

- **Consulta de exemplo 7:** BD EMPRESA. Descrición de todos os produtos co número de unidades vendidas de cada un, ordenados por unidades (de menos a máis), e no caso de que coincidan as unidades ordenarase alfabeticamente pola descrición dos produtos. Se hai produtos que non se venderon, na columna de unidades vendidas aparecerá 0.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Non é necesaria.
Táboas (FROM)	O nome das actividades atópase na táboa ACTIVIDADE. Se as actividades se pagaron ou non aparece en SOCIO_REALIZA_ACTI. O nome da cota FAMILIAR está en COTA. Para poder saber que socios pagan e que cota teñen asignada debemos consultar tamén a táboa SOCIO.
Columnas (SELECT)	Haberá unha columna, o nome das actividades debidas.
Combinacións (*joins*)	Hai que facer *join* entre PRODUTO e PEDIDO. A CP de PRODUTO está formada por dous atributos así que hai que facer *join* entre as dúas táboas igualando os pares de atributos da CP de PRODUTO coa correspondente foránea de PEDIDO.

	Se facemos un INNER JOIN teremos o problema de que só sairán os produtos dos que fixemos pedidos, é dicir, os que aparecen nas dúas táboas, por iso debemos facer un OUTER JOIN, LEFT ou RIGHT segundo poñamos a táboa PRODUTO á esquerda ou á dereita.
Funcións	Precísase a función de agregado *sum* que sume as unidades dos pedidos. Ao haber produtos sen pedidos aparecería NULL na columna de unidades vendidas, así que para evitar isto deberemos empregar tamén *isnull*.
Filtros de filas (WHERE)	Non hai que limitar o resultado por ningunha condición de enunciado.
Agrupamentos (GROUP BY)	É necesario agrupar porque temos que devolver un resultado por produto. Teremos que xuntar as filas dos pedidos de cada produto en diferentes grupos, así a suma das unidades calcularase de cada grupo, é dicir, de cada produto. Non se pode agrupar só polo identificador do produto, porque se pode repetir para distintos fabricantes. Ademais como amosamos a descrición tamén debemos poñelo no GROUP BY.
Filtro de grupos (HAVING)	Non se nos esixe eliminar ningún grupo do resultado.
Ordenación (ORDER BY)	Ordenamos por unidades vendidas e pola descrición do produto.
Alias	**De columna:** A columna das unidades debe ter un alias.
	De táboa: Ao haber máis dunha táboa no FROM usaranse sempre alias para evitar posibles ambigüidades con nomes de campos comúns a diferentes táboas, e tamén para facilitar a comprensión do código da consulta.

```
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT pr.descricion, isnull(sum(p.cantidade),0) as Unidades_vendidas
FROM PRODUTO pr left join PEDIDO p
     on pr.identificador=p.id_produto AND
        pr.cod_fabricante=p.cod_fabricante
GROUP BY pr.cod_fabricante, pr.identificador, pr.descricion
ORDER BY Unidades_vendidas, descricion;
```

Resultado da consulta do exemplo 7	
descricion	**Unidades_vendidas**
DDR3 4GB PC1600 CL11 DIMM. SRX8	0
DDR3 SO DIMM 4GB PC1333 CL9 SR	0
Disco Duro interno Toshiba MQ Series 1TB 2,5' SATA	0
Portátil 13,3' Satellite Z30-A-1DG Intel Core i5 4210U	0
Portátil Satellite Click 2 Pro 13,3' P30W-B-108 Intel Core i5 4210U	0
Rato Toshiba W30 Óptico sen fíos negro	0
Tablet 8' ME581C-1B007A Wi-Fi 16 GB	0
Tarxeta gráfica SVGA Asus NVIDIA GeForce 210 Silent DI/1GD3/V2(LP)	0
HD SSD 120GB 2.5 SATA3 v300	2
Pen Drive Daichi 32 GB 3.0 azul	3
Disco Duro portátil Toshiba 2 TB USB 3.0	5
Tarxeta de memoria SD PRO Clase 10 UHS-I de 16 GB	6
Multifunción Láser Xpress C460W Wi-Fi	7
HDD 1TB 7200rpm 64MB SATA3 6gbps	10
3D PRO Joystick	11
Bluetooth Audio Adapter	15
Cable USB a micro USB	15
HD 2.5 500GB 8MB 5400rpm SATA2	16
USB Headset H540	30
	32

Funda Book Cover para Samsung Galaxy Tab S 8,4' marrón	35
rato óptico logitech b100 negro	36
Portátil convertible 2 en 1 ASUS 10,1' T100TA-DK048H Intel Quad Core Atom Bay Trail-T Z3775	64
mk270 combo teclado con rato óptico	68
HD Webcam C270	

- **Consulta de exemplo 8:** BD SOCIEDADE_CULTURAL. Nome completo dos socios morosos, por orde alfabética de apelido e nome (ape1, ape2, nome en tres columnas). Son morosos os socios que deben a cota anual ou non pagaron algunha actividade.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Neste caso hai que decatarse que temos que compoñer un resultado a partires doutros dous, os morosos por non pagares algunha actividade e os morosos que non pagaron a cota anual. Precisarase unha unión dos resultados (UNION).
Táboas (FROM)	Consulta dos socios que deben a cota anual: na táboa SOCIO está o campo *abonada*. Consulta dos socios que deben actividades: *pagada* está en SOCIO_REALIZA_ACTI, pero pídese apelidos e nome dos socios que hai que buscalas en SOCIO.
Columnas (SELECT)	Primeiro apelido, segundo apelido e nome.
Combinacións (*joins*)	Só haberá *join* na consulta dos socios que deben actividades: entre *num_socio* (CF) de SOCIO_REALIZA_ACTI e *numero* de SOCIO.
Funcións	Non son necesarias.
Filtros de filas (WHERE)	Consulta dos socios que deben a cota anual: filtrarase por *abonada*='N'. Consulta dos socios que deben actividades: filtrarase por SOCIO.*pagada*='N'
Agrupamentos (GROUP BY)	Non se pide nin contar filas, nin sumar, nin calcular a media, nin mínimos nin máximos de columnas.
Filtro de grupos (HAVING)	Se non hai grupos, non hai que filtralos.
Ordenación (ORDER BY)	Ordenaremos polas columnas ape1, ape2 e nome.
Alias	**De columna:** Non.
	De táboa: Na segunda consulta da UNION ao haber máis dunha táboa no FROM usaranse sempre alias para evitar posibles ambigüidades con nomes de campos comúns a diferentes táboas, e tamén para facilitar a comprensión do código da consulta.

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT ape1, ape2, nome
FROM SOCIO
WHERE abonada='N'
UNION
SELECT ape1, ape2, nome
FROM SOCIO s, SOCIO_REALIZA_ACTI sr
WHERE sr.pagada='N' AND
      s.numero=sr.num_socio
ORDER BY ape1, ape2, nome;
```

Resultado da consulta do exemplo 8		
ape1	**ape2**	**nome**
DEL CARMEN SIEIRO	LÉREZ CAMPOS	JORGE MANUEL

- **Consulta de exemplo 9:** BD EMPRESA. Número de clientes que fixeron algún pedido.

  Unha vez lido o enunciado empezamos coa *análise do mesmo*.

Análise do enunciado	
Composición de consultas	Non é necesaria.
Táboas (FROM)	Só a táboa de PEDIDO, na que están os códigos dos clientes que fixeron pedidos.
Columnas (SELECT)	A cantidade total de clientes que fixeron pedidos.
Combinacións (*joins*)	Non son necesarios porque só hai unha táboa no FROM.
Funcións	Na táboa PEDIDO non existe ningún campo co número de clientes que fixeron pedidos. Necesitarase contar os distintos clientes que mercaron, coa función *count*. O *count* conta as filas, e se para que conte valores distintos terase que indicar a palabra DISTINCT. Na solución propoñemos as dúas solucións sen e con DISTINCT para que se vexa a diferenza no resultado. Sen DISTINCT devolve o número de filas de PEDIDO, e con DISTINCT devolve só as filas con valores distintos na columna de cliente.
Filtros de filas (WHERE)	Non hai que limitar o resultado por ningunha condición de enunciado.
Agrupamentos (GROUP BY)	Non hai que agrupar porque temos que contar os clientes de todos os pedidos.
Filtro de grupos (HAVING)	Non se nos esixe eliminar ningún grupo do resultado.
Ordenación (ORDER BY)	A consulta só devolve un valor e non é preciso ordenar.
Alias	**De columna:** A columna do resultado deberá ter nome e escollerase *clientes_con_pedido*.
	De táboa: Non é necesario porque no FROM só hai unha táboa.

```
--Seleccionamos a BD
USE EMPRESA;
GO
--Solución 1. Sen DISTINCT no count
SELECT count(num_cliente) as clientes_con_pedido
FROM PEDIDO;
--Solución 2. Con DISTINCT no count. CORRECTA
SELECT count(DISTINCT num_cliente) as clientes_con_pedido
FROM PEDIDO;
```

Resultado da consulta do exemplo 9. Solución 1
clientes_con_pedido
29
Resultado da consulta do exemplo 9. Solución 2. SOLUCIÓN CORRECTA DA CONSULTA
clientes_con_pedido
14

Metodoloxía. O procedemento que acabamos de ver é útil para o proceso de aprendizaxe, pero non se debe facer unha táboa de análise da consulta cada vez que se desexe obter información dunha BD.
Débese automatizar este proceso de análise para facelo de xeito automático.

8.1.2. Código de consultas complexas. Detección e corrección de erros

A continuación proporanse enunciados de consultas complexas coa súa codificación en T-SQL incorrecta. As tarefas consisten en executalas, estudar o resultado, sexa un erro ou un resultado non esperado, e corrixir a consulta, no editor de consultas do SSMS.

- **Consulta de exemplo 10:** Nome e límite de crédito dos clientes que teñen como segundo e terceiro caracteres 'er'. A solución proposta é a seguinte:

```
--Seleccionamos a BD
USE EMPRESA;
GO
--Solución PROPOSTA
SELECT nome, limite_de_credito
FROM CLIENTE
WHERE substring(nome,2,1)='er';
```

Executamos a consulta proposta e analizamos o resultado. Non se produce ningún erro, pero o resultado no é o esperado, porque se executamos unha consulta sobre toda a táboa de clientes, comprobamos que si existen clientes con segundo e terceiro caracteres er.

O erro está na función *substring*, xa que se lle está indicando que busque aqueles clientes que teñen como segundo carácter a cadea er, e é imposible que unha soa posición sexa ocupada por dous caracteres.

```
--Solución CORRECTA
SELECT nome, limite_de_credito
FROM CLIENTE
WHERE substring(nome,2,2)='er';
```

- **Consulta de exemplo 11:** Para cada sucursal con 2 ou máis empregados calcula a cota media das vendas dos empregados que traballan na sucursal. Amósase a cidade da sucursal e a cota media de vendas dos seus empregados. A solución proposta é a seguinte:

```
--Seleccionamos a BD
USE EMPRESA;
GO
--Solución PROPOSTA
SELECT s.cidade, avg(e.cota_de_vendas) as media_cota_vendas
FROM SUCURSAL s, EMPREGADO e
WHERE  count(s.identificador)>=2 AND
       s.identificador=e.id_sucursal_traballa
GROUP BY s.identificador, s.cidade;
```

Executamos a consulta proposta e analizamos o resultado. Prodúcese o seguinte erro.

```
No puede aparecer un agregado en la cláusula WHERE si no es en una subconsulta
contenida en una cláusula HAVING o en una lista de selección,
y siempre que la columna agregada sea una referencia externa.
```

O erro prodúcese por utilizar a función de agregado *count* na cláusula WHERE. En realidade se queremos quedarnos coas sucursais de 2 ou máis empregados, debemos calcular o *count* sobre os empregados de cada sucursal, é dicir sobre os grupos feitos. Deberemos cambiar a condición do WHERE para o HAVING.

```
--Solución CORRECTA
SELECT s.cidade, avg(e.cota_de_vendas) as media_cota_vendas
FROM SUCURSAL s, EMPREGADO e
WHERE  s.identificador=e.id_sucursal_traballa
GROUP BY s.identificador, s.cidade
HAVING count(s.identificador)>=2;
```

- **Consulta de exemplo 12:** Número, data e unidades dos pedidos que non se emitiran o 24 de febreiro do 2014 nin o 14 de abril do 2014. No resultado deberán aparecer os pedidos máis recentes primeiro. A solución proposta é a seguinte:

```
--Seleccionamos a BD
USE EMPRESA;
GO
--Solución PROPOSTA
SELECT numero, data_pedido, cantidade
FROM PEDIDO
WHERE convert(char(10),data_pedido,105) !='24-02-2014' OR
      convert(char(10),data_pedido,105) !='14-04-2014'
order by data_pedido DESC;
```

Executamos a consulta proposta e analizamos o resultado. Non se produce erro algún, pero podemos ver no resultado como seguen aparecendo pedidos nos días que non queremos que aparezan. De feito o resultado devolve os 29 pedidos, cando hai tres feitos nas datas do enunciado. O problema está no uso do OR, xa que a consulta comproba a data dos pedidos fila a fila, así que cando se chega a un pedido do 24 de febreiro de 2014 a primeira parte do OR é falsa, pero a segunda é verdadeira, co cal o resultado do WHERE é verdadeiro e devolve o pedido. Para que a consulta sexa correcta habería que usar AND, ou ben NOT IN.

```
--Solución 1 CORRECTA
SELECT numero, data_pedido, cantidade
FROM PEDIDO
WHERE convert(char(10),data_pedido,105) !='24-02-2014' AND
      convert(char(10),data_pedido,105) !='14-04-2014'
order by data_pedido DESC;
--Solución 2 CORRECTA
SELECT numero, data_pedido, cantidade
FROM PEDIDO
WHERE convert(char(10),data_pedido,105) NOT IN('24-02-2014','14-04-2014')
order by data_pedido DESC;
```

- **Consulta de exemplo 13:** Nome completo dos socios nunha columna e nunha segunda columna de nome *Situación* aparecerá 'PAGOU' ou 'NON PAGOU' segundo pagase ou non pagase a cota anual. A solución proposta é a seguinte:

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
--Solución PROPOSTA
SELECT rtrim(ape1+' '+isnull(ape2,''))+','+nome as nome_completo,
       CASE
            when abonada='N' then  'NON PAGOU'
            else 'PAGOU' as "Situación"
FROM SOCIO;
```

Executamos a consulta proposta e analizamos o resultado. Prodúcese un erro de sintaxe cerca da palabra clave AS. Revisamos a sintaxe e atopamos que falta a palabra END que pecha a instrución CASE.

```
--Solución CORRECTA
SELECT rtrim(ape1+' '+isnull(ape2,''))+','+nome as nome_completo,
       CASE
            when abonada='N' then  'NON PAGOU'
            else 'PAGOU'
       END as "Situación"
FROM SOCIO;
```

- **Consulta de exemplo 14:** Nome das cotas e número de socios que pagan cada unha delas, só para os socios que xa a pagaron. A solución proposta é a seguinte:

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT c.nome, count(numero) AS "Número de socios"
FROM SOCIO s, COTA c
WHERE s.abonada='S' AND
      s.cod_cota=c.codigo
GROUP BY cod_cota;
```

Executamos a consulta proposta e analizamos o resultado. Prodúcese o seguinte erro

```
La columna 'COTA.nome' de la lista de selección no es válida,
porque no está contenida en una función de agregado ni en la cláusula GROUP BY.
```

Estamos agrupando polo código da cota pero no SELECT aparece o nome da cota, e todos os campos que estean no SELECT e non sexan parte de funcións de agregado deben estar tamén no GROUP BY. O que hai que facer para corrixila é engadir o nome da cota á cláusula GROUP BY.

```
--Solución CORRECTA
SELECT c.nome, count(numero) AS "Número de socios"
FROM SOCIO s, COTA c
WHERE s.abonada='S' AND
      s.cod_cota=c.codigo
GROUP BY cod_cota, c.nome;
```

8.2. Optimización de consultas

Cando se realiza unha consulta non só debe pensarse en obter a información buscada. A consulta debe estar optimizada para que a información se recupere co menor costo posible de tempo e de recursos da máquina.

Algunhas das regras a ter en conta para optimizar as nosas consultas son:

- **Uso de DISTINCT.** Hai que usala só se é necesario, porque a sentenza DISTINCT xera unha grande cantidade de traballo e consume moitos recursos do servidor.
- **Uso de alias para as táboas.** Poñer o nome da táboa a que pertence cada campo (ou o alias da mesma) antes do nome do campo non é necesario se os nomes dos campos non aparecen en diferentes táboas. É importante porque facilita o entendemento do código da consulta e concretamente, e centrándonos na optimización, permite que o xestor aforre o tempo de ter que buscar cada campo a que táboa pertence.
- **Uso de *.**O uso de * impide a utilización de índices de forma eficiente para optimizar as buscas.
- **Uso de ORDER BY:** Débese usar ORDER BY nas consultas se é absolutamente indispensable. Se facemos consultas ordenadas normalmente por unha columna determinada debería estar indexada.
- **Verificar a existencia dunha ou varias filas:** É máis eficiente para comprobar se un rexistro existe na BD utilizar EXISTS que *count(*)*.
- **Uso de GROUP BY:** Aínda que é correcto, non se debe usar se non existe unha función colectiva ou de agregado (sum, count, avg, min, max), xa que o único que conseguimos é o mesmo que con DISTINCT, que é máis rápido que agrupar.
- **Subconsulta fronte a join:** Se a mesma consulta se pode realizar empregando un *join* ou unha subconsulta, será máis rápido usar o *join*.
- **Uso de LIKE:** O máis pesado e común dos argumentos de LIKE é o comodín % e sempre que se poida hai que tentar evitalo.
- **Orde das condicións no WHERE:** Se na cláusula WHERE existen outras condición ademais das condicións de *join*, sempre deberanse poñer en primeiro lugar as condicións que non sexan de combinación ou *join*. Conséguese deste xeito que o xestor cando fai o *join* teña un menos número de filas que combinar, xa que se eliminaron as que non cumprían as condicións anteriores da cláusula WHERE.
- **UNION ALL fronte a UNION:** Se sabemos que non se van repetir os valores no resultado dunha unión, é preferible usar UNION ALL porque UNION realiza internamente un DISTICT sobre os datos, que penaliza o rendemento.
- **Uso de NOT IN:** No seu lugar é mellor opción usar algunha das seguintes de menor a maior rendemento:
  - EXISTS ou NOT EXISTS
  - IN

- LEFT OUTER JOIN e comprobar cunha condición NULL.

- **IN fronte BETWEEN:** Sempre que sexa posible é mellor opción empregar BETWEEN e non IN.

  Por exemplo, no caso de ter unha sentenza WHERE sobre un campo enteiro de nome *campo1* como a seguinte, WHERE campo1 IN(7,8,9,10), pódese perfectamente substituír por WHERE campo1 BETWEEN 7 AND 10, porque os valores de IN forman un rango de valores.

A continuación proporanse enunciados de consultas complexas sen erros de sintaxe pero con problemas de rendemento. As tarefas consisten en detectar o problema de rendemento e optimizar a consulta, no editor de consultas do SSMS.

- **Consulta de exemplo 15:** A consulta a optimizar é a seguinte.

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT DISTINCT nif
FROM SOCIO
WHERE year(data_nac)>1950;
```

O problema de optimización está no DISTINCT xa que non o precisamos porque non poden existir dous nifs iguais. Para optimizala eliminamos a palabra DISTINCT da consulta.

```
--Consulta de exemplo 15 OPTIMIZADA
SELECT nif
FROM SOCIO
WHERE year(data_nac)>1950;
```

- **Consulta de exemplo 16:** A consulta a optimizar é a seguinte.

```
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT cidade, cota_de_vendas
FROM SUCURSAL, EMPREGADO
WHERE  identificador=id_sucursal_traballa AND
       rexion='LESTE';
```

Neste caso atopamos dous problemas. Non está indicado a que táboa pertence cada campo (con alias ou co nome da táboa diante de cada campo). Por outro lado a orde das condicións no WHERE non é a óptima xa que débese filtrar sempre antes de facer o *join*.

```
--Consulta de exemplo 16 OPTIMIZADA
SELECT s.cidade, e.cota_de_vendas
FROM SUCURSAL s, EMPREGADO e
WHERE  s.rexion='LESTE' AND
       s.identificador=e.id_sucursal_traballa;
```

- **Consulta de exemplo 17:** A consulta a optimizar é a seguinte.

```
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT *
FROM PRODUTO;
```

Non se debe empregar nunca * no SELECT dunha consulta para recuperar todos os campos dunha táboa. Débese especificar os nomes de tódolos campos.

```
--Consulta de exemplo 17 OPTIMIZADA
SELECT cod_fabricante, identificador, descricion, prezo, existencias
FROM PRODUTO;
```

- **Consulta de exemplo 18:** A consulta a optimizar é a seguinte.

```
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT cod_fabricante, identificador, descricion
FROM PRODUTO
GROUP BY cod_fabricante, identificador, descricion;
```

A cláusula GROUP BY non está asociada nesta consulta a ningunha función colectiva ou de agregado como *sum*, *count*, *avg*, *max* ou *min*. Co seu uso nesta sentenza SQL só se consegue ordenar o resultado, do mesmo xeito que se faría coa cláusula ORDER BY e de xeito máis rápido.

```
--Consulta de exemplo 18 OPTIMIZADA
SELECT cod_fabricante, identificador, descricion, prezo, existencias
FROM PRODUTO
ORDER BY cod_fabricante, identificador, descricion;
```

- **Consulta de exemplo 19:** A consulta a optimizar é a seguinte.

```
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT nome
FROM CLIENTE
WHERE 1<= (SELECT count(*)
           FROM PEDIDO
           WHERE num_empregado=108);
```

Esta consulta pretende devolver os nomes dos clientes só no caso de que o empregado de código 108 fixese algún pedido. O uso da función *count(*)* para verificar a existencia dun rexistro debe ser substituído polo uso de EXISTS para mellorar a eficiencia da consulta.

```
--Consulta de exemplo 19 OPTIMIZADA
SELECT nome
FROM CLIENTE
WHERE EXISTS (SELECT numero
              FROM PEDIDO
              WHERE num_empregado=108);
```

- **Consulta de exemplo 20:** A consulta a optimizar é a seguinte.

```
USE SOCIEDADE_CULTURAL;
GO
SELECT nif, nome, ape1, ape2
FROM EMPREGADO
UNION
SELECT nif, nome, ape1, ape2
FROM SOCIO;
```

Se partimos do suposto de que un empregado non pode ser socio da sociedade cultural e deportiva, sabemos que as dúas consultas non van ter filas en común, polo que non van existir filas repetidas no resultado final. Neste caso débese substituír UNION por UNION ALL para evitar que o servidor internamente antes de amosar o resultado execute un DISTINCT cando non é necesario.

```
--Consulta de exemplo 20 OPTIMIZADA
USE SOCIEDADE_CULTURAL;
GO
SELECT nif, nome, ape1, ape2
FROM EMPREGADO
UNION ALL
SELECT nif, nome, ape1, ape2
FROM SOCIO;
```

- **Consulta de exemplo 21:** A consulta a optimizar é a seguinte.

```
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT numero, nome, ape1, ape2
FROM EMPREGADO
WHERE numero IN (104,105,106,107);
```

O campo *numero* da táboa EMPREGADO é enteiro. Obsérvase que na cláusula WHERE os números do parénteses do IN constitúen un rango, do 104 ao 107, polo que é posible utilizar o predicado BETWEEN no canto de IN.

```
--Consulta de exemplo 21 OPTIMIZADA
USE EMPRESA;
GO
SELECT numero, nome, ape1, ape2
FROM EMPREGADO
WHERE numero BETWEEN 104 AND 107;
```

- **Consulta de exemplo 22:** A consulta a optimizar é a seguinte.

```
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT DISTINCT cod_cota
FROM SOCIO
WHERE cod_cota IN (SELECT codigo
                   FROM COTA
                   WHERE importe BETWEEN 50 and 100);
```

Esta consulta devolve a mesma información, de xeito máis eficiente, realizando un *inner join* entre SOCIO e COTA.

```
--Consulta de exemplo 22 OPTIMIZADA
SELECT DISTINCT s.cod_cota
FROM SOCIO s, COTA c
WHERE c.importe BETWEEN 50 and 100 AND
      s.cod_cota=c.codigo;
```

8.3. Tarefa de consultas complexas

Tarefa 1. Unha vez copiados, executados e probados os exemplos das explicacións, realizarase o código T-SQL adecuado para obter a información que se pide en cada unha das consultas propostas na BD EMPRESA. Aínda que non se indique, nos resultados todas as columnas terán un nome que identifique correctamente a información que amosan.

Todas as consultas deberán facerse empregando unha consulta optimizada, aínda que o resultado se poida obter doutros xeitos.

- Consultas propostas nas BBDD **EMPRESA** e **SOCIEDADE_CULTURAL**:

  - **Proposta 1.** DB SOCIEDADE_CULTURAL. Nif e nome completo nunha columna (ape1 ape2, nome) de cada socio, só para os socios que deben algunha actividade. Nunha segunda columna aparecerá o importe total que debe en actividades. A columna do nome chamarase *nome_completo* e a do importe debido *cantidade_debe*. O resultado aparecerá por orde alfabética de apelidos e nome dos socios.

  - **Proposta 2.** BD EMPRESA. Número de pedido, descrición e prezo do produto, unidades vendidas e importe de todos os pedidos da BD ordenados de maior a menor importe. No caso de coincidir os importes deberá ordenarse alfabeticamente pola descrición do produto.

  - **Proposta 3.** BD EMPRESA. Número de pedido, descrición e prezo do produto, unidades vendidas e importe dos pedidos da BD con importe superior a 1000€, ordenados de maior a menor importe. No caso de coincidir os importes deberá ordenarse alfabeticamente pola descrición do produto.

  - **Proposta 4.** BD EMPRESA. Número de pedido, nome do cliente e data de pedido dos pedidos recibidos nos días en que se contrataron empregados. No resultado deben aparecer primeiro os pedidos máis recentes.

  - **Proposta 5.** DB EMPRESA. Nome completo dos empregados co nome do empregado que teñen por xefe. Na primeira columna de nome *empregado* aparecerá o nome completo de cada empregado co formato *ape1 ape2, nome*, teña ou non teña xefe. Na segunda columna de nome *xefe* aparecerá o nome completo do xefe dese empregado co mesmo formato que o campo *empregado*. No caso de non ter xefe na segunda columna aparecerá a frase 'XEFE POR DETERMINAR'.

    Non se amosan os empregados que dirixen a sucursal onde traballa cada un

deles, senón o xefe directo. (Ver campo EMPREGADO.num_empregado_xefe).

- **Proposta 6.** DB SOCIEDADE_CULTURAL. Gasto en actividades por socio. Na primeira columna aparecerá o nif do socio e na segunda o gasto, pagado ou non, que leva feito en actividades. Os socios que non participaron en actividades non aparecerán no resultado.

- **Proposta 7.** DB SOCIEDADE_CULTURAL. Nome e apelidos, (en tres columnas de nomes *apelido1*, *apelido2* e *nome_propio*) das persoas que forman parte da nosa sociedade cultural independentemente da súa labor na sociedade. Nunha cuarta columna *cargo* se a persoa é empregado aparecerá a frase 'É EMPREGADO' e noutro caso 'NON É EMPREGADO'. O resultado aparecerá ordenado alfabeticamente por apelidos e nome.

- **Proposta 8.** BD EMPRESA. Empregando unha consulta composta amosa o identificador das sucursais que non teñen empregados traballando nelas.

- **Proposta 9.** BD EMPRESA. Nunha columna *nome_abreviado* amosa os tres primeiros caracteres en minúsculas do primeiro apelido de cada empregado.

- **Proposta 10.** DB SOCIEDADE_CULTURAL. Nome e apelidos, (en tres columnas de nomes *apelido1*, *apelido2* e *nome_propio*) dos socios que cumpren anos no mes actual.

Tarefa 2. Dadas unha serie de consultas SQL pouco eficientes volverase a escribir o código T-SQL coas correccións necesarias para que a consulta mellore o seu rendemento. Os cambios deberanse xustificar.

- Consultas propostas nas BBDD **EMPRESA** e **SOCIEDADE_CULTURAL**:

  - **Consulta a optimizar 1.**

```
/*** CONSULTA A OPTIMIZAR 1 ***/
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT DISTINCT sum(p.cantidade) as Unidades_totais
FROM CLIENTE c, PEDIDO p
WHERE c.numero=p.num_cliente AND
      c.nome='APPS INFOR, SL';
```

  - **Consulta a optimizar 2.**

```
/*** CONSULTA A OPTIMIZAR 2 ***/
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT nome
FROM ACTIVIDADE
WHERE identificador IN (SELECT id_actividade
                        FROM PROFE_CURSA_ACTI);
```

  - **Consulta a optimizar 3.**

```
/*** CONSULTA A OPTIMIZAR 3 ***/
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT nif, nome, ape1, ape2
FROM SOCIO, SOCIO_REALIZA_ACTI
WHERE numero=num_socio;
```

  - **Consulta a optimizar 4.**

```
/*** CONSULTA A OPTIMIZAR 4 ***/
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT numero, nome
FROM CLIENTE
WHERE numero IN (1109,1106,1108,1107);
```

- **Consulta a optimizar 5.**

```
/*** CONSULTA A OPTIMIZAR 5 ***/
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT *
FROM CLIENTE;
```

- **Consulta a optimizar 6.**

```
/*** CONSULTA A OPTIMIZAR 6 ***/
--Seleccionamos a BD
USE SOCIEDADE_CULTURAL;
GO
SELECT nif, nome, ape1, ape2
FROM SOCIO
GROUP BY nome, ape1, ape2, nif;
```

- **Consulta a optimizar 7.**

```
/*** CONSULTA A OPTIMIZAR 7 ***/
--Seleccionamos a BD
USE EMPRESA;
GO
SELECT identificador
FROM PRODUTO
UNION
SELECT codigo
FROM FABRICANTE;
```

Anexo 1. Bases de datos de traballo

As BBDD sobre as que realizamos as consultas de exemplo, propostas e de avaliación, son a BD SOCIEDADE_CULTURAL e a BD EMPRESA. A continuación exporase unha breve descrición das mesmas (*minimundo*) e o diagrama de base de datos xerado por SQL Server de cada unha delas.

BD SOCIEDADE_CULTURAL

Minimundo

Esta BD garda a seguinte información dunha sociedade cultural:

- Empregados (profesores/as ou administrativos/as).
- Actividades.
- Aulas.
- Socios.
- Cotas.

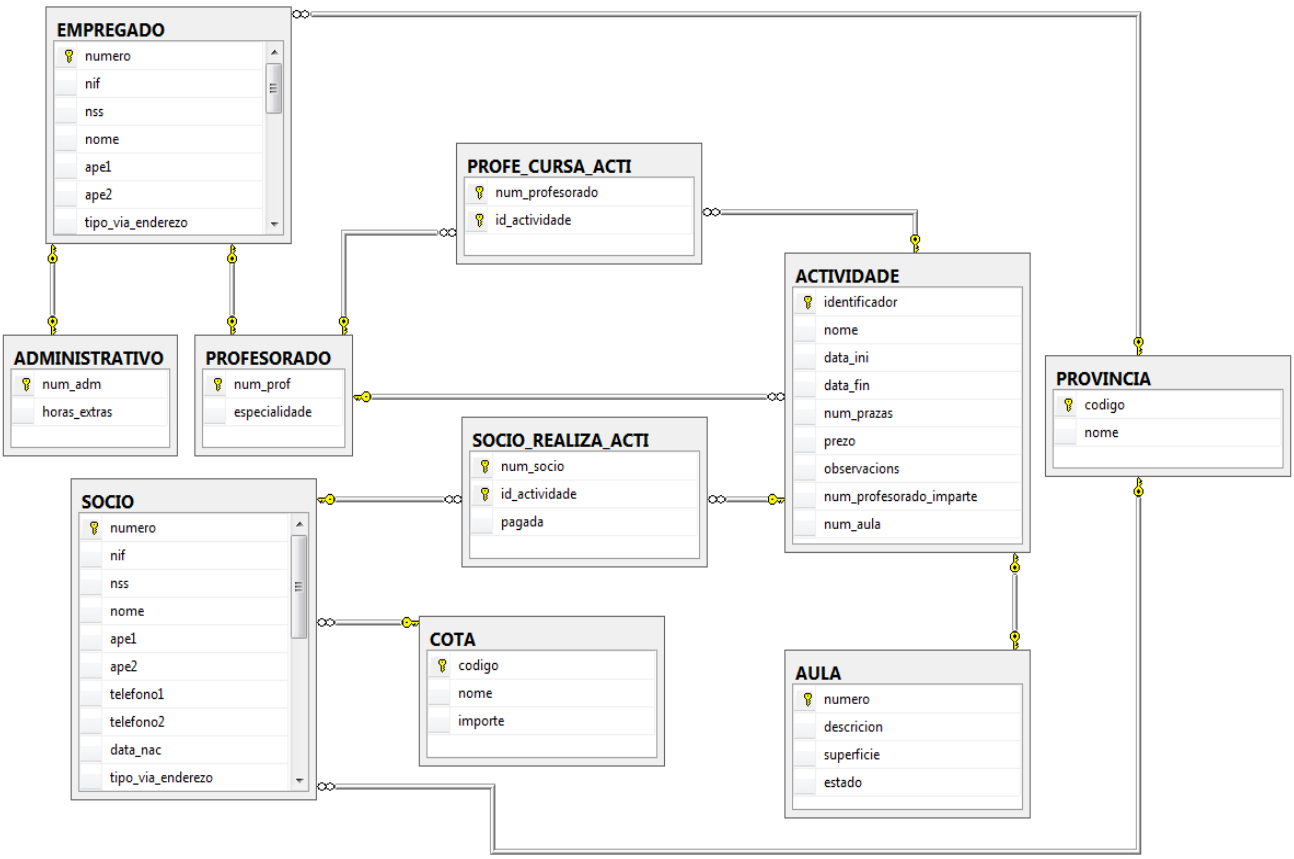
O profesorado está asociado as actividades que imparte. O profesorado e socios poden asistir a actividades. O profesorado non poderá asistir aos cursos que imparte.

Deberase gardar información de se os socios pagaron ou non a cota anual, e tamén se pagaron cada unha das actividades nas que se matricularon.

Cada actividade terá unha aula asociada. Cada aula é exclusiva dunha actividade.

Gardarase tamén a provincia na que residen empregados e socios.

Diagrama de Base de Datos



BD EMPRESA

Minimundo

Esta BD garda a seguinte información dunha empresa:

- Empregados.
- Sucursais.
- Fabricantes.
- Produtos.
- Pedidos.
- Clientes.

Os empregados traballan cada un nunha sucursal da empresa. Algúns empregados son xefes, é dicir, teñen outros empregados ao seu cargo. Ademais dentro dos empregados gárdanse os que son directores de sucursais. Cada sucursal ten un único director, pero o mesmo empregado pode dirixir varias sucursais.

Cada produto da empresa está asociado ao seu fabricante. En cada pedido rexístrase a seguinte información:

- Cliente que solicita o pedido.
- Empregado que se encarga do pedido.
- Produto que o cliente solicita (identificado polo código do fabricante e do produto).
- Data do pedido.
- Número de unidades de produto solicitadas no pedido.

Deberase gardar información de que empregado ten asignado cada cliente. Pode ocorrer que nun momento determinado un cliente non sexa atendido polo vendedor que ten asignado. Por exemplo, se o empregado asignado ao cliente está de vacacións, o cliente deberá facer o pedido que necesite a outro empregado.

Diagrama de Base de Datos