

# Consultas Simples

---

<b>1.</b>	<b>Introducción.....</b>	<b>3</b>
1.1	Bases de datos de traballo .....	3
1.1.1	Base de datos tendaBD .....	3
1.1.2	Base de datos practicas5 .....	6
1.1.3	Base de datos traballadores .....	7
<b>2.</b>	<b>Manipulación de datos con SQL .....</b>	<b>8</b>
2.1	Sentenza SELECT para consulta de datos .....	8
2.1.1	Lista de selección .....	9
2.1.2	Expresións.....	11
	Constantes .....	11
	Variables de usuario.....	14
	Operadores.....	14
	Operadores aritméticos .....	14
	Operadores relacionais .....	15
	Operadores lóxicos .....	15
	Precedencia dos operadores .....	15
2.2	Consultas simples.....	16
2.2.1	Cláusula FROM .....	16
2.2.2	Cláusula WHERE.....	17
	Predicado BETWEEN.....	18
	Predicado IN .....	18
	Predicado LIKE .....	19
	Predicado IS [ NOT ] NULL.....	19
2.2.3	Cláusula ORDER BY .....	20
2.2.4	Cláusula LIMIT.....	22
2.3	Funcións incorporadas en MySQL .....	23
2.3.1	Funcións de data e hora.....	23
	Información sobre data e hora do servidor .....	23
	Aritmética de datas.....	23
	Formato de saída para datos tipo data e hora .....	25
2.3.2	Funcións de cadeas de caracteres .....	26
2.3.3	Funcións numéricas.....	28
2.3.4	Funcións de agrupamento ou de columna .....	30
	Valores NULL e as funcións de agrupamento:.....	32
2.3.5	Outras funcións.....	32
	Funcións de control de fluxo .....	32
	Funcións de información do sistema .....	33
	Funcións de cifrado.....	33
	Funcións de conversión de tipos.....	34

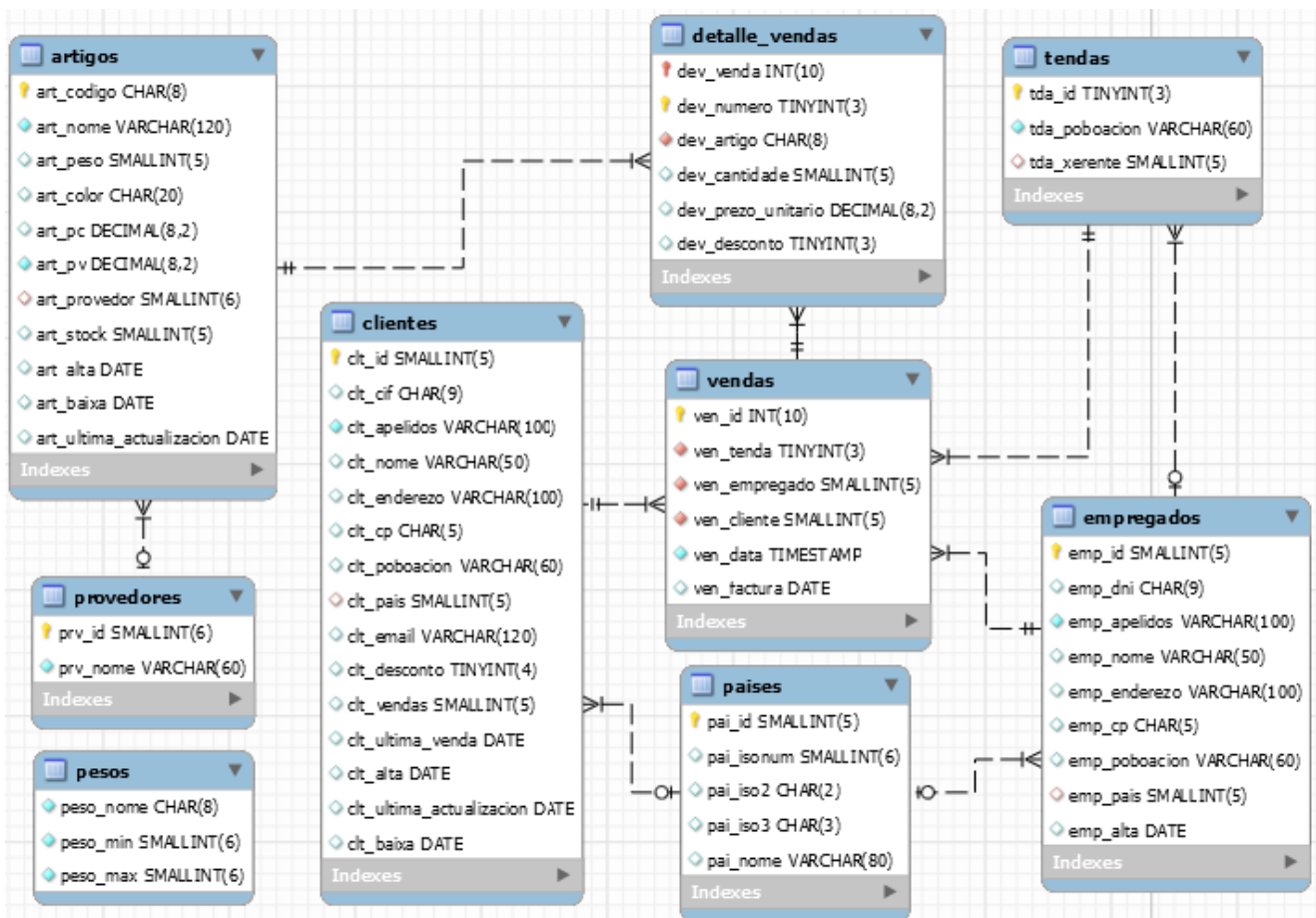
# 1. Introducción

## 1.1 Bases de datos de traballo

As bases de datos *tendaBD*, *traballadores* e *practicar5* utilizarémolos para os exemplos e algunhas tarefas deste tema. Antes de empezar a probar os exemplos ou realizar as tarefas, hai que executar os scripts de creación no servidor e poñer en uso a base de datos correspondente.

### 1.1.1 Base de datos tendaBD

A base de datos *tendaBD* serve para controlar as vendas dunha cadea de tendas. Gárdanse nela os datos das vendas que se realizan, das tendas nas que se fan as vendas, dos artigos vendidos, e dos clientes. As táboas desta base de datos móstranse no seguinte diagrama e descríbense a continuación.



### ■ Táboa *empregados*

Nome columna	Tipo	Null	Clave	Observacións
emp_id	smallint unsigned	Non	Primaria	Identificador do empregado. Numéranse de 1 en diante de forma automática.
emp_dni	char(9)			DNI do empregado.
emp_apelidos	varchar(100)	Non	Índice	Apelidos do empregado.
emp_nome	varchar(50)			Nome do empregado.
emp_enderezo	varchar(100)			Enderezo do empregado.
emp_cp	char(5)			Código postal do empregado.
emp_poboacion	varchar(60)			Poboación do empregado.
emp_pais	smallint unsigned		Foránea	Código do país segundo a táboa de países.
emp_alta	date			Data na que se deu de alta o empregado.

### ■ Táboa *pesos*

Nome columna	Tipo	Null	Clave	Observacións
peso_nome	char(8)	Non		Nome que describe o tipo de peso.
peso_min	smallint	Non		Peso mínimo para ese nome.
peso_max	smallint	Non		Peso máximo para ese nome.

### ■ Táboa *clientes*

Nome columna	Tipo	Null	Clave	Observacións
clt_id	smallint unsigned	Non	Primaria	Identificador do cliente. Numeraranse de 1 en diante de forma automática.
clt_cif	char(9)		Única	
clt_apelidos	varchar(100)	Non	Índice	Apelidos ou razón social do cliente.
clt_nome	varchar(50)			Nome ou tipo de sociedade (SL, SA, ...) do cliente.
clt_enderezo	varchar(100)			
clt_cp	char(5)			Código postal do cliente.
clt_poboacion	varchar(60)			
clt_pais	smallint unsigned		Foránea	Código do país segundo a táboa de países.
clt_email	varchar(120)			
clt_desconto	tinyint			Porcentaxe de desconto aplicable ao cliente.
clt_vendas	smallint unsigned			Número de vendas feitas ao cliente.
clt_ultima_venta	date			Data da última venda feita ao cliente.
clt_alta	date	Non		Data na que se deu de alta ao cliente.
clt_ultima_actualizacion	date			Data da última vez que se fixeron cambios nos datos do cliente.
clt_baixa	date			Data na que se deu de baixa ao cliente.

### ■ Táboa *artigos*

Nome columna	Tipo	Null	Clave	Observacións
art_codigo	char(8)	Non	Primaria	Toma valores entre 1 e 200.000.
art_nome	varchar(120)	Non	Índice	Nome ou descrición do artigo.

art_peso	smallint unsigned			Peso en gramos. Valor numérico enteiro.
art_color	char(20)			Cor do artigo
art_pc	decimal(8,2)			Prezo de compra do artigo.
art_pv	decimal(8,2)	Non		Prezo de venda do artigo.
art_proveedor	smallint		Foránea	Identificador do proveedor.
art_stock	smallint unsigned			Número de unidades do artigo dispoñibles no almacén.
art_alta	date	Non		Data na que se deu de alta o artigo.
art_baixa	date			Data na que se deu de baixa o artigo.
art_ultima_actualizacion	date			Data da última vez que se fixeron cambios nos datos do artigo.

#### ■ Táboa *países*

Nome columna	Tipo	Null	Clave	Observacións
pai_id	smallint unsigned	Non	Primaria	Identificador do país. Numeraranse de 1 en diante de forma automática.
pai_isonum	smallint			Número de país segundo a norma ISO 3166-1:2013. <sup>1</sup>
pai_iso2	char(2)			Código de país de 2 caracteres segundo a norma ISO 3166-1:2013.
pai_iso3	char(3)			Código de país de 3 caracteres segundo a norma ISO 3166-1:2013.
pai_nome	varchar(80)			Nome do país.

#### ■ Táboa *provedores*

Nome columna	Tipo	Null	Clave	Observacións
prv_id	smallint	Non	Primaria	Identificador do proveedor.
prv_nome	varchar(60)	Non		Nome do proveedor.

#### ■ Táboa *tendas*

Nome columna	Tipo	Null	Clave	Observacións
tda_id	tinyint unsigned	Non	Primaria	Identificador da tenda. Numéranse do 1 en diante de forma automática.
tda_poboacion	varchar(60)	Non		Poboación na que está situada a tenda.
tda_xerente	smallint unsigned		Foránea	Identificador do empregado que é xerente da tenda.

#### ■ Táboa *vendas*

Nome columna	Tipo	Null	Clave	Observacións
ven_id	int unsigned	Non	Primaria	Identificador da venda. Numeraranse de 1 en diante de forma automática.
ven_tenda	tinyint unsigned	Non	Foránea	Identificador da tenda na que se fixo a venda.
ven_empregado	smallint unsigned	Non	Foránea	Identificador do empregado que fixo a venda.
ven_cliente	smallint unsigned	Non	Foránea	Identificador do cliente ao que se fixo a venda.
ven_data	date	Non		Data e hora na que se fixo a venda.

---

<sup>1</sup> Máis información sobre a norma ISO 3166-1:2013 en [https://es.wikipedia.org/wiki/ISO\\_3166-1](https://es.wikipedia.org/wiki/ISO_3166-1)

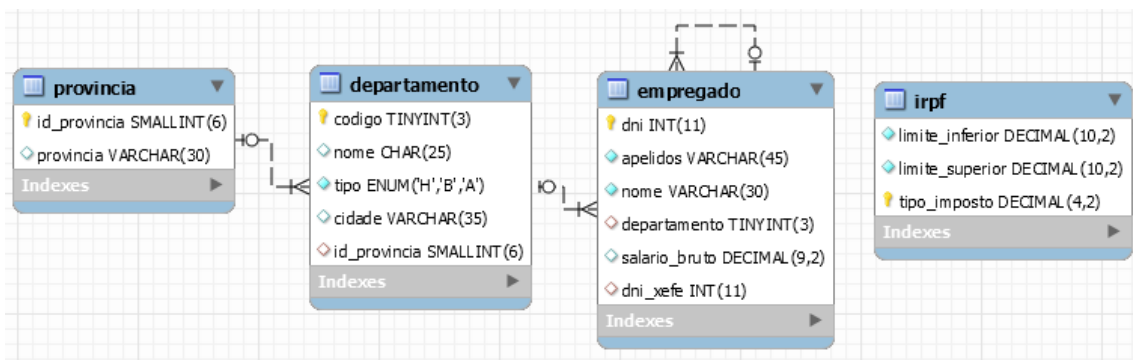
ven_factura	date			Data da factura na que se inclúe esta venda.
-------------	------	--	--	--

#### ■ Táboa *detalle\_vendas*

Nome columna	Tipo	Null	Clave		Observacións
dev_venta	int unsigned	Non	Primaria	Foránea	Identificador da venda á que corresponde a liña de detalle.
dev_numero	tinyint unsigned	Non			Número da liña de detalle dentro da venda.
dev_artigo	char(8)	Non	Foránea		Identificador do artigo vendido.
dev_cantidade	smallint unsigned	Non			Número de unidades vendidas.
dev_prezo_unitario	decimal(8,2) unsigned	Non			Prezo por cada unidade vendida.
dev_desconto	tinyint unsigned	Non			Porcentaxe de desconto aplicado.

### 1.1.2 Base de datos practicas5

A base de datos *practicas5* está creada para realizar os exemplos de consultas nesta unidade. Está formada por un grupo de táboas, relacionadas entre si, tal e como se mostra na seguinte imaxe e se describe a continuación.



- Táboa *empleado*. A columna *departamento* é unha clave foránea que contén o código do departamento no que traballa o empregado, e fai referencia á columna *codigo* da táboa *departamento*. Os valores que toma a columna *departamento* teñen que coincidir cos que toma a columna *codigo* da táboa *departamento*, ou ser NULL no caso que o empregado non teña asignado ningún departamento. A columna *dni\_xefe* é outra clave foránea que contén o dni doutro empregado que sería o seu xefe, ou o valor NULL no caso que non tivera xefe.
- Táboa *departamento*. A columna *id\_provincia* é unha clave foránea que fai referencia á columna *id\_provincia* da táboa *provincia*.
- Táboa *irpf*. Contén a porcentaxe de imposto que hai que aplicarlle a cada empregado, en función do seu salario bruto, dependendo dos límites entre os que se atope. Esta táboa podería conter unha información similar a esta:

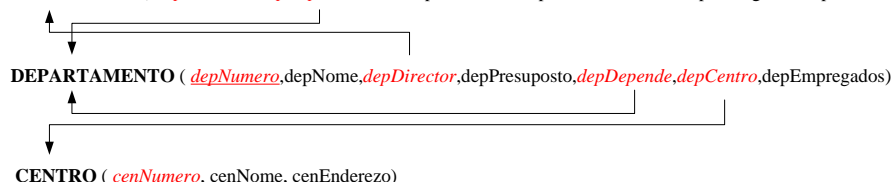
Result Grid			Filter Rows:
	limite_inferior	limite_superior	tipo_impuesto
▶	0.00	17707.00	15.75
	17707.00	33007.00	21.00
	33007.00	53407.00	27.00
	53407.00	120000.00	30.00
	120000.00	175000.00	35.00
	175000.00	300000.00	42.00

Salario bruto entre 0 e 17107 euros corresponde un 15.75% de imposto

### 1.1.3 Base de datos traballadores

A base de datos *traballadores* serve para levar control dos empregados, departamentos e centros dunha empresa. Está formada por un grupo de táboas, relacionadas entre si, tal e como se mostra no seguinte grafo relacional e se describe a continuación. As táboas son MyIsam (non transaccionais) e por tanto non teñen definidas claves foráneas.

EMPREGADO ( *empNumero*, *empDepartamento*, empExtension, empDataNacemento, empDatIngreso, empSalario, empComision, empFillos, empNome)



#### ■ Táboa centro

Nome columna	Tipo	Null	Clave	Observacións
cenNumero	int	Non	Primaria	Número co que se identifica.
cenNome	char(30)		Índice	Nome.
cenEnderezo	char(30)			Enderezo.

#### ■ Táboa empregado

Nome columna	Tipo	Null	Clave	Observacións
empNumero	int	Non	Primaria	Número co que se identifica.
empDepartamento	int	Non	Índice	Número do departamento no que traballa.
empExtension	smallint	Non		Extensión telefónica para o empregado. Pode compartirse entre empregados de diferentes departamentos.
empDataNacemento	date			Data de nacemento.
empDataIngreso	date			Data de ingreso na empresa.
empSalario	decimal(6,2)			Salario mensual en euros.
empComision	decimal(6,2)			Comisión mensual.
empFillos	smallint			Número de fillos.
empNome	char(20)	Non	Índice	Nome do empregado coa forma: primeiro apelido, nome.

#### ■ Táboa departamento

Nome columna	Tipo	Null	Clave	Observacións
depNumero	int	Non	Primaria	Número co que se identifica.
depNome	char(20)		Índice	Nome.

depDirector	int	Non	Índice	Número do empregado director do departamento.
deptipoDirector	char(1)			Tipo de directo: P (en propiedade, é dicir, titular), F (en funcións).
depPresuposto	decimal(9,2)			Cantidade en euros de presuposto anual.
depDepende	int		Índice	Número do departamento do que depende.
depCentro	int		Índice	Número do centro ao que pertence.
depEmpregados	smallint unsigned			Número de empregados que traballan no departamento.

## 2. Manipulación de datos con SQL

SQL corresponde ao acrónimo de *Structured Query Language* (Linguaxe Estruturado de Consultas). Aínda que nun principio foi creado para realizar consultas, utilízase para controlar todas as funcións que subministra un SXBDR aos seus usuarios, incluíndo todas as funcións propias das linguaxes deseñadas para o manexo de bases de datos: Linguaxe de Definición de Datos, Linguaxe de Manipulación de Datos e Linguaxe de Control de Datos.

A linguaxe de manipulación de datos ou LMD (en inglés *Data Management Language* ou *DML*), permite realizar as operacións necesarias para manexar os datos almacenados nunha base de datos. Estas operacións son: inserir filas de datos (sentenza INSERT), modificar o contido das filas de datos (sentenza UPDATE), borrar filas de datos (sentenza DELETE), e consultar os datos contidos nas táboas da base de datos (sentenza SELECT).

### 2.1 Sentenza SELECT para consulta de datos

**A sentenza SELECT permite realizar consultas e é a instrución máis potente e complexa de todas as instrucións SQL.** A pesar da gran cantidade de opcións que ofrece, é posible empezar formulando consultas simples e ir construíndo consultas máis complexas a medida que se teña maior coñecemento da linguaxe.

A sentenza SELECT recupera datos das táboas dunha base de datos e devolve o resultado da consulta en forma de táboa, coas filas e columnas seleccionadas. Resumo da sintaxe da sentenza:

```
SELECT [ALL | DISTINCT | DISTINCTROW] lista_de_selección
[FROM lista_referencias_táboas
[WHERE expresión_condicional]
[GROUP BY {lista_columnas | expresión | posición}
[HAVING expresión_condicional]
[ORDER BY {columna | expresión | posición} [ASC | DESC], ...]
[LIMIT intervalo]
[INTO OUTFILE 'nome_arquivo']
```

- As opcións DISTINCT e ALL especifican se hai que mostrar as filas duplicadas ou non. A opción ALL é o valor predeterminado se non se indica ningunha das opcións, e significa que se mostran todas as filas, incluídas as duplicadas. A opción DISTINCT utilízase cando se queren eliminar as filas duplicadas do conxunto de resultados. A opción DISTINCTROW é un sinónimo de DISTINCT.
- A lista de selección (*lista\_de\_selección*) é a relación dos datos que se queren obter na consulta, separados por comas.
- A cláusula FROM utilízase para escribir a relación de táboas utilizadas na consulta e as relacións que hai entre elas.
- A cláusula WHERE utilízase para escribir as condicións que teñen que cumprir as filas para que se mostren no conxunto de resultados.



- A cláusula GROUP BY permite agrupar as filas do conxunto de resultados.
- A cláusula HAVING utilízase para escribir as condicións que teñen que cumprir os grupos resultantes de agrupar as filas do conxunto de resultados coa cláusula GROUP BY.
- A cláusula LIMIT utilízase para restrinxir o número de filas retornadas pola consulta.
- A cláusula INTO OUTFILE utilízase para enviar o conxunto de resultados a un arquivo.

Nas especificacións da sintaxe indícase a orde en que se teñen que escribir as cláusulas opcionais. No caso de escribir as cláusulas nun orden diferente do que figura na sintaxe, móstrase unha mensaxe de erro.

As sentenzas SELECT, igual que o resto de sentenzas de SQL rematan nun punto e coma, aínda que MySQL permite escribir as consultas sen o punto e coma final por tratarse dunha única sentenza. É unha boa práctica utilizar o punto e coma sempre, porque este é o xeito habitual de indicar o fin dunha instrución en programación.

### 2.1.1 Lista de selección

A *lista\_de\_selección* da sentenza SELECT, indica os datos que se queren obter coa consulta, separados por comas. Cada dato representa unha columna da táboa que contén o conxunto de resultados, e **pode conter calquera das funcións e operadores que soporta MySQL**. A sentenza permite recuperar datos de cálculos ou de información do servidor, sen facer referencia a ningunha táboa, como por exemplo:

```
select 2+2, version(), user(), curdate();
```

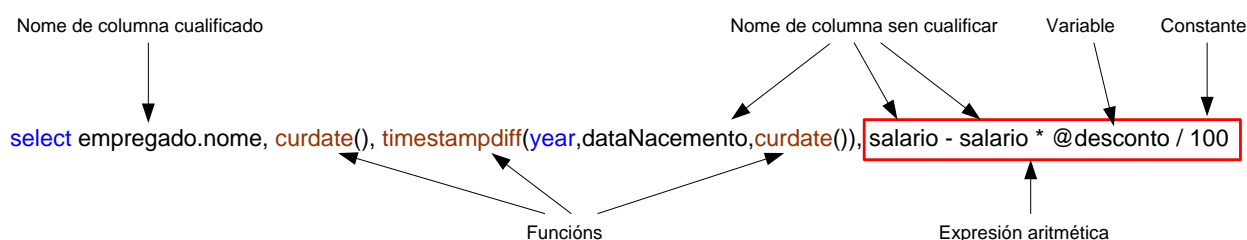
Result Grid				
Filter Rows:				
	2+2	version()	user()	curdate()
▶	4	5.6.17	root@localhost	2015-11-28

A lista de selección ten que cumprir a seguinte sintaxe:

```
expresión [ [ AS ] alias_columna ] [, expresión [ [ AS ] alias_columna ]] ...
```

- A *expresión* pode ser o nome dunha columna, unha expresión aritmética, unha expresión de data, unha constante, unha variable de usuario, unha función, ou unha combinación de todas esas cousas. Cando se fai referencia a un nome de columna, pode suceder que ese nome exista en máis dunha das táboas relacionadas na cláusula FROM; nese caso para diferenciarlas e que non existan nomes ambiguos na lista de selección, é obrigatorio utilizar nomes cualificados co formato *nome\_táboa.nome\_columna*.

Exemplo:



- Os *alias\_columna* son nomes que se lles dá ás columnas da táboa de resultados e son especialmente útiles no caso de columnas que conteñen unha expresión. Se non se asigna un alias, o nome da columna coincide coa expresión. Os alias poden ser utilizados nas cláusulas ORDER BY, GROUP BY e HAVING, pero non poden ser utilizados na cláusula WHERE xa que nesta cláusula establécense condicións para filtrar filas das táboas relacionadas na cláusula FROM, e os alias corresponden a nomes de columnas da táboa de resultados que no momento de resolver a cláusula WHERE aínda está en formación.

Como calquera nome de columna, os alias deben cumprir as regras do servidor referidas aos caracteres e tamaño permitidos, aínda que MySQL permite definir alias que non cumpran estas regras sempre que vaian pechados entre comiñas. Estes últimos só se deben utilizar para que o contido da columna se entenda mellor, pero poden dar problemas cando se utilizan noutras cláusulas.

Exemplos:

```
select 2+2,
       version() as 'Versión do servidor',
       user() as Usuario_conectado,
       curdate() as dataActual;
```

A utilización de alias permite facilitar a lectura dos resultados, como se pode ver na saída que produce a consulta anterior.

Result Grid		Filter Rows:		Export:	
	2+2	Versión do servidor	Usuario_conectado	dataActual	
▶	4	5.6.17	root@localhost	2015-11-28	

Como a palabra AS é opcional, pode ocorrer un sutil problema no caso de esquecerse da coma entre dúas expresións da lista de selección: MySQL interpreta a segunda expresión como un nome de alias.

Exemplo:

```
select apellidos nome, cidade from empregado;
```

Nesta consulta MySQL interpreta que *nome* é un alias de *apellidos*. Por esta razón é unha boa práctica poñer os alias de columnas empregando a palabra AS diante.

- Na lista de selección pódese empregar o símbolo \* para representar todas as columnas das táboas seleccionadas, ou empregar o formato *nome\_táboa.\** para facer referencia a todas as columnas dunha táboa.

Exemplo:

```
select * from departamento; /* Mostra todas as columnas da táboa departamento*/
```

Result Grid					
Filter Rows:					
	codigo	nome	tipo	cidade	id_provincia
▶	1	Central	H	Lugo	27
	2	Oficina1	H	Monforte	27
	3	Oficina2	B	Ferrol	15
	4	Oficina3	H	Vigo	36
	5	Oficina4	A	Ourense	32
	6	Oficina5	A	Villalba	27
	7	Oficina6	H	Ourense	32
	8	Oficina7	H	Lugo	27
	9	Oficina8	A	Coruña	15
	10	Oficina9	B	Villalba	28
	* NULL	NULL	NULL	NULL	NULL

departamento 4 x

Apply Revert

Output

Action Output

	Time	Action	Message	Duration / Fetch
✓	1 18:10:27	select * from departamento	10 row(s) returned	0.000 sec / 0.000 sec

Cando as consultas están embebidas no código dun programa nunca se debe utilizar o símbolo \*, xa que pode ocorrer que se modifique o esquema da táboa e isto afecte ao resultado que devolve a consulta. Por exemplo, a consulta anterior devolve 5 columnas; modifícase o esquema da táboa e engádese unha nova columna; despois do cambio, a consulta devolve 6 columnas; isto pode producir un erro na aplicación se é que o código da mesma só está preparado para manexar 5 columnas.

O \* é útil para facer probas de condicións, mostrando todas as columnas e seleccionando só as filas que cumpran a condición.

## 2.1.2 Expresións

Os elementos que poden conter as expresións son: nomes de columnas, constantes ou literais, variables de usuario, funcións e operadores.

### Constantes

As constantes ou valores literais representan valores fixos. As máis utilizadas en MySQL son as de tipo cadeas de caracteres, números, data e hora, booleanas ou lóxicas, e o valor NULL.

- Os valores constantes de tipo cadea de caracteres represéntanse por un conxunto de caracteres pechados entre comiñas simples ou dobres. A configuración por defecto do servidor MySQL, fai que non se diferencie entre maiúsculas e minúsculas, excepto que a cadea se defina como binaria (BINARY).

Exemplo:

```
select 'cadea con comiñas simples',
       "cadea con comiñas dobres",
       'cadea que inclúe un carácter " (comiñas dobres)';
```

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	cadea con comiñas simples	cadea con comillas dobres	cadea que inclue un carácter " (comiñas dobres)		
▶	cadea con comiñas simples	cadea con comillas dobres	cadea que inclue un carácter " (comiñas dobres)		

- Se o servidor ten habilitado o modo ANSI\_QUOTES, as cadeas só poden delimitarse con comiñas simples. Nese caso, unha cadea delimitada por comiñas dobres será interpretada como un identificador. Na instalación por defecto de MySQL este modo está deshabilitado.
- Dentro das comiñas pódense utilizar algunhas secuencias de caracteres para representar caracteres especiais. Estas secuencias empezan cun carácter de escape, representado pola barra invertida (\). As secuencias de escape que reconece MySQL pódense consultar no manual de referencia. Nesta táboa móstranse algúns dos máis empregados:

\n	Carácter de salto de liña (LF). Código ASCII: 10
\r	Carácter de retorno de carro. (CR). Código ASCII: 13
\t	Carácter de tabulación
\\	Carácter de barra invertida (\)

Exemplos:

```
select 'exemplo \r\n cambio de liña';
select 'A barra invertida non se mostra, \ pero pódese "escapar" \\ para mostrala';
```

Resultado da execución:

exemplo cambio de liña	A barra invertida non se mostra, \ pero pódese "escapar" \ para mostrala
---------------------------	--

- Os valores constantes de tipo numérico enteiro represéntanse como unha secuencia de díxitos; os de tipo decimal utilizan o carácter punto (.) como separador decimal. Calquera número pode ir precedido do signo menos (-) para indicar un valor negativo, ou do signo máis para indicar un valor positivo (+). Se non se pon ningún signo, considérase positivo. Exemplos:

```
select 258 as enteiro, 3658.25 as 'decimal', -25 as negativo;
```

Result Grid				Filter Rows:	Export:
	entero	decimal	negativo		
	258	3658.25	-25		

- Os valores constantes de tipo DATE (data), TIME (hora) e DATETIME (data e hora) representan valores pechados entre comiñas simples ou dobres cun determinado formato.
  - No caso das datas, o formato establecido é *'aaaa-mm-dd'* ou *'aa-mm-dd'*, onde *aaaa* e *aa* representan o ano con catro díxitos ou dous díxitos, *mm* os dous díxitos que corresponden ao mes e *dd* os dous díxitos que corresponden ao día. MySQL admite calquera carácter como delimitador de ano, mes e día; e incluso admite que se omitan os delimitadores.

Exemplos:

```
select '2015-12-25','15-12-25','20151225'
```

- No caso das horas, o formato establecido é *'hh:mm:ss'* onde *hh* representa os dous díxitos que corresponden á hora, *mm* os dous díxitos que corresponden aos minutos e *ss* os dous díxitos que corresponden aos segundos.

Exemplos:

```
select '12:00:30','12:00:30','120030'
```

- Os valores constantes de tipo lóxico en MySQL son TRUE (verdadeiro) e FALSE (falso) e avalíanse como 1 e 0 respectivamente. Os nomes destas constantes pódense escribir en calquera combinación de maiúsculas e minúsculas.

Exemplos:

```
select TRUE, true, FALSE, false, 3>2, 5=6;
```

Result Grid							Filter Rows:
	TRUE	TRUE	FALSE	FALSE	3>2	5=6	
	1	1	0	0	1	0	

- O valor NULL significa 'non hai dato' ou 'valor descoñecido'. A palabra pódese escribir en calquera combinación de maiúsculas e minúsculas. Hai que ter en conta que o valor NULL non é o mesmo que o valor 0 para tipos numéricos ou á cadea valeira para tipos cadea de caracteres.

Exemplos:

```
select NULL, null, @variable;
```

Result Grid				Filter Rows:
	NULL	NULL	@variable	
	NULL	NULL	NULL	

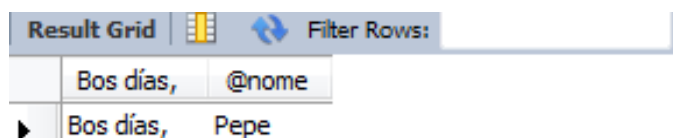
## Variables de usuario

As variables de usuario permiten almacenar un valor e facer referencia a el máis tarde; isto fai posible pasar valores dunha sentenza a outra. As variables de usuario son propias da conexión na que se crean e non poden ser vistas por outros clientes. A memoria utilizada polas variables de usuario é liberada no momento en que se pecha a conexión. Os nomes das variables de usuario levan como primeiro carácter un símbolo @.

Para asignar valores ás variables de usuario utilízase a sentenza SET, ou ben pódense asignar na propia sentenza SELECT co operador := .

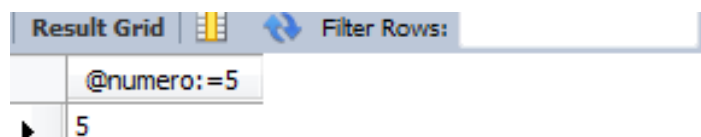
Exemplos:

```
set @nome = 'Pepe';  
select 'Bos días, ', @nome;
```



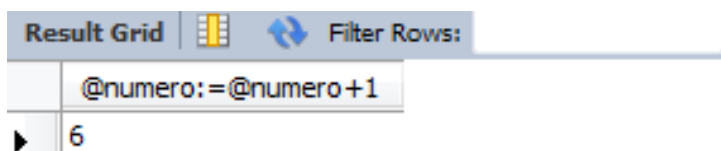
	Bos días,	@nome
▶	Bos días,	Pepe

```
select @numero:=5;
```



	@numero:=5
▶	5

```
select @numero:=@numero+1;
```



	@numero:=@numero+1
▶	6

## Operadores

MySQL dispón de multitude de operadores diferentes para cada un dos tipos de dato. Os operadores permiten construír expresións que se utilizan na lista de selección, ou nas cláusulas das sentenzas de manipulación de datos, incluída a sentenza SELECT.

### Operadores aritméticos

Permiten facer operacións aritméticas con datos de tipo numérico. Se algún dos operandos toma o valor NULL o resultado da operación sempre é o valor NULL.

*	Multiplicación
/ DIV	División e División enteira
% MOD	Resto da división enteira
+	Suma
-	Resta

## Operadores relacionais

Permiten facer comparacións entre expresións devolvendo o valor 1 (*true = verdadeiro*) ou 0 (*false = falso*). Se un dos valores a comparar é o valor NULL, o resultado é NULL, excepto cando se utiliza o operador  $\leq$ , ou o operador IS NULL.

=	Igual
!= <>	Distinto
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
<=>	Igual, pero devolve 1 en lugar de NULL se ambos operandos son NULL, ou 0 en lugar de NULL se un deles é NULL
IS	Comparación co valor NULL. Explicase con máis detalle no apartado: Cláusula WHERE
LIKE	Comparación con patrón de busca. Explicase con máis detalle no apartado: Cláusula WHERE
REGEXP	Comparación con patrón de busca. Explicase con máis detalle no apartado: Cláusula WHERE
IN	Comparación cun conxunto de valores. Explicase con máis detalle no apartado: Cláusula WHERE
BETWEEN	Comparación cun intervalo de valores. Explicase con máis detalle no apartado: Cláusula WHERE

## Operadores lóxicos

Permiten formar condicións compostas por máis dunha condición, ou negar unha condición.

NOT	NEGACIÓN. Devolve o valor verdadeiro cando non é verdadeira a condición
AND &&	E. Devolve o valor verdadeiro cando son verdadeiras as dúas condicións
XOR	OU LÓXICO. Devolve o valor verdadeiro cando é verdadeira unha das dúas condicións, pero non as dúas
OR	OU. Devolve o valor verdadeiro cando é verdadeira algunha das dúas condicións, ou se son verdadeiras as dúas

Para comparar filas de valores, a expresión  $(a, b) = (x, y)$  é equivalente á expresión  $(a = x)$  and  $(b = y)$ .

## Precedencia dos operadores



As expresións avalíanse tendo en conta a precedencia dos operadores que a forman. A precedencia está establecida polo SXBDR, aínda que é posible cambiala introducindo parénteses nas expresións. O primeiro que se vai a avaliar é a expresión que está contida entre parénteses e dentro dos parénteses utilízase a precedencia preestablecida. A precedencia dos operadores máis empregados móstrase na seguinte táboa, na que os operadores que están dentro da mesma liña teñen a mesma precedencia e por tanto execútanse a medida que se aparecen na expresión dende esquerda á dereita.

1	*, /, DIV, %, MOD
2	-, +
3	= (comparación), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN

4	BETWEEN, CASE, WHEN, THEN, ELSE
5	NOT
6	AND, &&
7	XOR
8	OR,

Exemplos:

```
select 1+2*3, (1+2)*3, 5/2, 5 div 2, 5 mod 2, 5%2; # aritméticos
```

Result Grid				Filter Rows:			Export
	1+2*3	(1+2)*3	5/2	5 div 2	5 mod 2	5%2	
▶	7	9	2.5000	2	1	1	

```
select 3=2, 3<=>2, null = null, null <=>null, 3 = null, 3 <=> null;
```

Result Grid		Filter Rows:		Export:		W
	3=2	3<=>2	null = null	null <=>null	3 = null	3 <=> null
▶	0	0	NULL	1	NULL	0

```
select 3>=2, 2>3, 2<>3, 3!=3, 3 in (2,3,4);
```

Result Grid		Filter Rows:			
	3>=2	2>3	2<>3	3!=3	3 in (2,3,4)
▶	1	0	1	0	1

```
select not (3>=2), 3>=2 and 7>8, 3>=2 or 7>8, 3>=2 xor 7>8, 3>=2 or 7<8, 3>=2 xor 7<8
```

Result Grid		Filter Rows:		Export:	Wrap Cell Content:	
	not (3>=2)	3>=2 and 7>8	3>=2 or 7>8	3>=2 xor 7>8	3>=2 or 7<8	3>=2 xor 7<8
	0	0	1	1	1	0

## 2.2 Consultas simples

Son consultas que utilizan unicamente as cláusulas FROM, WHERE, ORDER BY, LIMIT e INTO OUTFILE, aínda que non teñen porque utilízalas todas. Neste tema realízanse consultas deste tipo tomando os datos dunha soa táboa.

### 2.2.1 Cláusula FROM

Esta cláusula utilízase para escribir a relación das táboas nas que están os datos que se utilizan na consulta. Cada elemento da *lista\_referencias\_táboas* segue a seguinte sintaxe:

```
referencia_táboa [[AS] alias_táboa]
[[USE INDEX (key_list)]
| [IGNORE INDEX (key_list)]
| [FORCE INDEX (key_list)]]
```

- A *referencia\_táboa* pode ser o nome dunha táboa ou unha expresión (posta entre parénteses) que dá como resultado unha táboa. O alias é un nome que pode ser empregado en lugar do nome da táboa. Se a referencia de táboa é unha expresión, hai que especificar obrigatoriamente un alias.



Exemplos:

- Utilizando o nome dunha táboa:

```
select dni,apelidos, nome
from empregado;
```

- Utilizando unha expresión que dá como resultado unha táboa:

```
select codigo, nome, cidade
from (select * from departamento where cidade = 'Lugo') as DepartamentosLugo;
```

Na segunda consulta, o resultado de executar a sentenza SELECT que vai pechada entre paréntese ten forma de táboa e asignáselle o nome *DepartamentosLugo*. A consulta principal colle os datos desa táboa.

O nome da táboa pode ter o formato *nome\_base\_datos.nome\_táboa* para indicar de forma explícita a base de datos á que pertence a táboa.

- As opcións USE INDEX, IGNORE INDEX, FORCE INDEX permiten dar pistas ao optimizador de consultas acerca de cómo escoller os índices para facer a consulta. Pode ser moi útil en consultas que utilizan máis dunha táboa.

Máis adiante veremos a sintaxe da cláusula FROM cando hai que coller datos de máis dunha táboa.

## 2.2.2 Cláusula WHERE

Esta cláusula permite seleccionar as filas, mediante condicións do tipo:

```
expresión operador_relacional expresión
expresión [NOT] BETWEEN expresión1 AND expresión2
expresión [NOT] IN (lista_de_valores)
nome_columna [NOT] LIKE 'patrón_de_busca'
nome_columna [NOT] REGEXP [BINARY] 'expresión_regular'
nome_columna IS [NOT] NULL
```

As condicións utilizadas na cláusula poden ser compostas, combinando máis dunha condición simple cos operadores lóxicos:

- AND ou && : Para que a condición composta devolva o valor verdadeiro todas as condicións teñen que ser verdadeiras.
- OR ou || : Para que a condición composta devolva o valor verdadeiro é suficiente con que sexa verdadeira algunha das condicións.
- XOR : Para que a condición composta devolva o valor verdadeiro ten que ser verdadeira algunha das condicións pero non as dúas.

Exemplo de condición composta: seleccionar os *apelidos*, *nome*, *salario\_bruto* e *departamento* dos empregados do departamento 1 que teñan un salario superior a 40000 euros.

```
select apelidos, nome, salario_bruto, departamento
from empregado
where departamento = 1 and salario_bruto > 40000;
```

	apelidos	nome	salario_bruto	departamento
▶	Fernandez Lopez	Jose Luis	160000.00	1
	Fernandez Diaz	Julian	85400.00	1

## Predicado BETWEEN

Permite comparar o valor da expresión situada á esquerda da palabra BETWEEN cos valores comprendidos no intervalo definido pola expresión1 e a expresión2, ambas incluídas. Sintaxe:

```
expresión [NOT] BETWEEN expresión1 AND expresión2
```

Exemplo: seleccionar *apelidos*, *nome* e *salario\_bruto* dos empregados que teñan un salario bruto entre 50000 e 70000 euros, ambos incluídos.

```
select apelidos, nome, salario_bruto
from empregado
where salario_bruto between 50000 and 70000;
```

Result Grid				Filter Rows:
	apelidos	nome	salario_bruto	
▶	Iglesias Dominguez	Adolfo	52500.00	
	Bernardez Macia	Luisa	65200.00	
	Porto Novo	Begoña	52000.00	
	Canedo Tellez	Angeles	58500.00	
	Cendan Villa	Lorenzo	65350.00	

## Predicado IN

Permite comparar o valor da expresión situado á esquerda da palabra IN coa lista de valores pechados entre parénteses. Sintaxe:

```
expresión [NOT] IN (lista_de_valores)
```

Exemplos:

- Seleccionar o *nome*, *cidade* e *id\_provincia* de todos os departamento situados en provincias galegas, e dicir, que a columna *id\_provincia* tome os valores 15, 27, 32 ou 36.

```
select * from practicas5.departamento
where id_provincia in (15,27,32,36);
```

Result Grid						Filter Rows:	Edit:
	codigo	nome	tipo	cidade	id_provincia		
▶	1	Central	H	Lugo	27		
	2	Oficina1	H	Monforte	27		
	3	Oficina2	B	Ferrol	15		
	4	Oficina3	H	Vigo	36		
	5	Oficina4	A	Ourense	32		
	6	Oficina5	A	Villalba	27		
	7	Oficina6	H	Ourense	32		
	8	Oficina7	H	Lugo	27		
	9	Oficina8	A	Coruña	15		

- Seleccionar o *nome*, *cidade* e *id\_provincia* de todos os departamentos situados en provincias que non sexan galegas, e dicir, que a columna *id\_provincia* tome valores distintos a 15, 27, 32 ou 36.

```
select * from practicas5.departamento
where id_provincia not in (15,27,32,36);
```

Result Grid					Filter Rows:
	codigo	nome	cidade	id_provincia	
▶	10	Oficina9	Villalba	28	

## Predicado LIKE

Permite realizar unha comparación de semellanza entre o valor dunha expresión e o dunha cadea de caracteres que representa un patrón de busca. Sintaxe:

```
nome_columna [NOT] LIKE 'patrón_de_busca'
```

No patrón pódense empregar os seguintes caracteres 'comodín':

% (porcentaxe)	Substitúe a un grupo de caracteres. Exemplos: 'A%' representa un grupo de caracteres que empeza por 'A' '%ez' representa un grupo de caracteres que acaba por 'ez' '%a%' representa un grupo de caracteres que contén a letra 'a'
_ (guión baixo)	Substitúe a un carácter. Exemplo: '1__a' representa un grupo de caracteres que empeza por lo número 1, seguido de dous caracteres calquera e remata coa letra 'a'

Exemplos:

- Seleccionar *apelidos*, *nome* e *salario\_bruto* de todos os empregados que teñen uns *apelidos* que empezan por 'A'.

```
select nome, apelidos
from empregado
where apelidos like 'A%';
```

Result Grid			Filter Rows:
	nome	apelidos	
▶	Dorinda	Abelleira Carrion	
	Luis	Aguiar Lopez	

- Seleccionar o *nome* e *apelidos* de todos os empregados que teñen un *nome* formado por 6 letras, e acaba en 'AN'.

```
select nome, apelidos
from empregado
where nome like '_____an';
```

Result Grid			Filter Rows:
	nome	apelidos	
▶	Julian	Fernandez Diaz	
	Adrian	Garcia Perez	

## Predicado IS [ NOT ] NULL

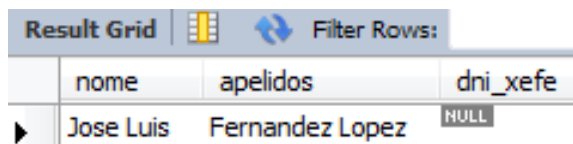
Serve para comprobar se o contido dunha columna é un valor nulo ou non. En SQL, o valor nulo representa un valor descoñecido, diferente de 0 e dunha cadea baleira. Sintaxe:

```
nome_columna IS [NOT] NULL
```

Exemplos:

- Seleccionar *nome*, *apelidos* e *dni\_xefe* dos empregados que non teñen xefe, ou non se coñece quen é o seu xefe, é dicir, os que teñen o valor NULL na columna *dni\_xefe*.

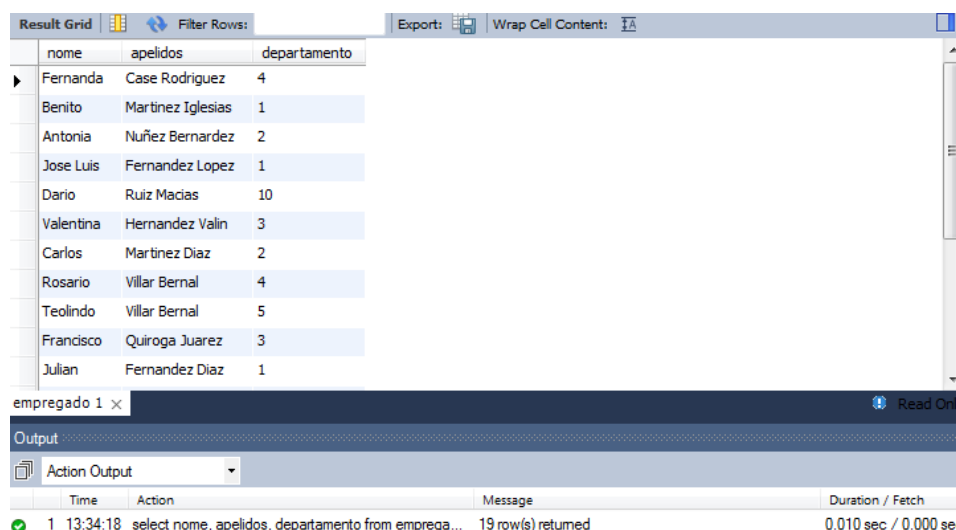
```
select nome, apelidos, dni_xefe
from empregado
where dni_xefe is null;
```



	nome	apelidos	dni_xefe
▶	Jose Luis	Fernandez Lopez	NULL

- Seleccionar *nome*, *apelidos* e *departamento* dos empregados que teñen asignado un departamento, e dicir, que teñen un valor distinto de NULL na columna *departamento*.

```
select nome, apelidos, departamento
from empregado
where departamento is not null;
```



	nome	apelidos	departamento
▶	Fernanda	Case Rodriguez	4
	Benito	Martinez Iglesias	1
	Antonia	Nuñez Bernardes	2
	Jose Luis	Fernandez Lopez	1
	Dario	Ruiz Macias	10
	Valentina	Hernandez Valin	3
	Carlos	Martinez Diaz	2
	Rosario	Villar Bernal	4
	Teolindo	Villar Bernal	5
	Francisco	Quiroga Juarez	3
	Julian	Fernandez Diaz	1

empregado 1 x

Output

Action Output

	Time	Action	Message	Duration / Fetch
✓	1	13:34:18	select nome, apelidos, departamento from empra...	19 row(s) returned 0.010 sec / 0.000 sec

## 2.2.3 Cláusula ORDER BY

Esta cláusula permite ordenar os resultados da consulta tendo en conta o contido dunha ou máis columnas ou expresións. Sintaxe:

```
[ORDER BY {columna | expresión | posición} [ASC | DESC], ...]
```

- Os nomes de columna aos que se fai referencia nesta cláusula son os da táboa do conxunto de resultados da consulta, polo tanto, se hai alias asociados ás columnas, poden ser utilizados. Tamén se poden utilizar expresións, ou o número de orde da columna na lista de selección. Non é recomendable utilizar o número de orde da columna porque no caso de facer cambios na lista de selección, cambiando a orde das columnas, ou engadindo ou eliminando algunha columna, habería que cambiar tamén a cláusula ORDER BY para que non varíe o resultado.

- As opcións [ASC | DESC] indican a orde na que se mostran os resultados (ASC = ascendente, DESC = descendente). Se non se indica nada, tómasse ASC como valor por defecto.

Exemplos:

- Seleccionar *nome*, e *apelidos* de todos os empregados ordenando o resultado alfabeticamente polos apelidos, e no caso de haber empregados que teñan os mesmos *apelidos*, ordénanse alfabeticamente polo *nome*.

```
select nome, apelidos
from empregado
order by apelidos, nome;
```

The screenshot shows a database interface with a 'Result Grid' and an 'Output' window. The 'Result Grid' displays the following data:

nome	apelidos
Valentina	Hernandez Valin
Adolfo	Iglesias Dominguez
Carlos	Martinez Diaz
Benito	Martinez Iglesias
Antonia	Nuñez Bernardez
Begoña	Porto Novo
Francisco	Quiroga Juarez
Dario	Ruiz Macias
Maria	Sanchez Rodriguez
Rosario	Villar Bernal
Teolindo	Villar Bernal

The 'Output' window shows the following log entry:

Time	Action	Message	Duration / Fetch
2 13:55:29	select nome, apelidos from empregado order by a...	20 row(s) returned	0.000 sec / 0.000 sec

- Mostrar o nome, apelidos e o 20% do *salario\_bruto*, que representa a retención que se lle vai a facer, de todos os empregados ordenando o resultado pola retención, de tal maneira que se mostren en primeiro lugar os que teñen unha retención máis alta.

```
select nome, apelidos, salario_bruto * 0.20 as retencion
from empregado
order by retencion desc;
```

The screenshot shows a database interface with a 'Result Grid' and an 'Output' window. The 'Result Grid' displays the following data:

nome	apelidos	retencion
Jose Luis	Fernandez Lopez	32000.0000
Julian	Fernandez Diaz	17080.0000
Lorenzo	Cendan Villa	13070.0000
Luisa	Bernardez Macia	13040.0000
Angeles	Canedo Tellez	11700.0000
Adolfo	Iglesias Dominguez	10500.0000
Begoña	Porto Novo	10400.0000
Antonia	Nuñez Bernardez	8400.0000
Francisco	Quiroga Juarez	7300.0000
Fernanda	Case Rodriguez	7104.0000
Maria	Sanchez Rodriguez	7080.0000

The 'Output' window shows the following log entry:

Time	Action	Message	Duration / Fetch
1 13:57:51	select nome, apelidos, salario_bruto * 0.20 as reten...	20 row(s) returned	0.010 sec / 0.000 sec

Calquera das seguintes solucións tamén serían válidas, aínda que a anterior é a máis recomendable:

```
-- utilizando unha expresión
select nome, apellidos, salario_bruto * 0.20 as retencion
from empregado
order by salario_bruto * 0.20 desc;

-- utilizando o número de orden da columna dentro da lista de selección
select nome, apellidos, salario_bruto * 0.20 as retencion
from empregado
order by 3 desc;
```

## 2.2.4 Cláusula LIMIT

Esta cláusula permite indicar o número de filas que ten que devolver a consulta e de forma optativa dende que posición se van a mostrar. Pode ter un ou dous argumentos numéricos, que deben ser enteiros positivos (incluíndo cero). No caso de utilizar un só argumento, este indica o número de filas que hai que mostrar; e no caso de utilizar dous argumentos, o primeiro indica o número de fila a partir da que hai que mostrar o conxunto de resultados, e o segundo representa o nº de filas.

Exemplos:

- Seleccionar o *nome*, *apellido* e *retención* dos 5 empregados coas retencións máis altas. A retención representa o 20% do *salario\_bruto*.

```
select nome, apellidos, salario_bruto * 0.20 as retencion
from empregado
order by retencion desc
limit 5;
```



	nome	apellidos	retencion
▶	Jose Luis	Fernandez Lopez	32000.0000
	Julian	Fernandez Diaz	17080.0000
	Lorenzo	Cendan Villa	13070.0000
	Luisa	Bernardez Macia	13040.0000
	Angeles	Canedo Tellez	11700.0000

A consulta mostra as 5 primeiras filas da táboa de resultados, despois de ordenar pola columna *retencion*.

- Seleccionar o *nome*, *apellido* e *retención* dos 3 empregados seguintes aos mostrados no exemplo anterior, tendo en conta que están ordenados polas retencións, de maior a menor. A retención representa o 20% do *salario\_bruto*.

```
select nome, apellidos, salario_bruto * 0.20 as retencion
from empregado
order by retencion desc
limit 5,3;
```

Result Grid			
	nome	apelidos	retencion
▶	Adolfo	Iglesias Dominguez	10500.0000
	Begoña	Porto Novo	10400.0000
	Antonia	Nuñez Bernardez	8400.0000

A consulta mostra 3 filas da táboa de resultados, empezando pola 6ª, despois de ordenar pola columna *retencion*.

## 2.3 Funcións incorporadas en MySQL

Cando se necesita facer algunha operación complexa, débese de consultar o manual para saber se xa existe algunha función implícita no servidor para resolvela antes de poñerse a crear unha función nova de usuario. MySQL incorpora unha gran cantidade de funcións que poden ser utilizadas nas expresións contidas nas consultas. O manual de referencia de MySQL posúe información moi detallada das funcións; aquí móstrase un resumo das máis utilizadas.

### 2.3.1 Funcións de data e hora

Permiten obter información sobre a data e hora do sistema en distintos formatos, resolver os problemas de aritmética de datas (cálculos aritméticos feitos con datos tipo data e hora) e cambiar o formato de saída.

#### Información sobre data e hora do servidor

- `CURRENT_DATE()` ou `CURDATE()`  
Devolve a data actual do sistema, en formato 'aaaa-mm-dd'
- `CURRENT_TIME()` ou `CURTIME()`  
Devolve a hora actual do sistema, en formato 'hh:mm:ss'
- `NOW()`  
Devolve a data e hora do sistema, en formato 'aaaa-mm-dd hh:mm:ss'

Exemplo considerando que agora sean as 11:14:10 do 13/12/2015:

```
select current_date(), curdate(), curtime(), now();
```

Result Grid				
	current_date()	curdate()	curtime()	now()
▶	2015-12-13	2015-12-13	11:14:10	2015-12-13 11:14:10

#### Aritmética de datas

Para facer operacións aritméticas con datas en MySQL, hai que utilizar as funcións que incorpora para tal fin. Algunhas destas funcións poden manexar intervalos de tempo facendo referencia ás palabras reservadas: MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, ou YEAR.

- **DATE\_ADD**

Suma (*add*) a unha data que se pasa como primeiro parámetro o intervalo de tempo especificado como segundo parámetro. Sintaxe:

```
DATE_ADD(data, INTERVAL expresión unidade_intervalo)
```

- **ADDDATE**

Suma (*add*), a unha data que se pasa como primeiro parámetro, un número de días ou un intervalo de tempo, que se pasan como segundo parámetro. Sintaxe:

```
ADDDATE(data, {número_días | INTERVAL expresión unidade_intervalo })
```

Cando se utiliza coa forma INTERVAL no segundo parámetro, dá o mesmo resultado cá función DATE\_ADD.

- **DATE\_SUB**

Resta (*subtract*) a unha data que se pasa como primeiro parámetro o intervalo de tempo especificado como segundo parámetro. Sintaxe:

```
DATE_ADD(data, INTERVAL expresión unidade_intervalo)
```

- **SUBDATE**

Resta (*subtract*), a unha data que se pasa como primeiro parámetro, un número de días ou un intervalo de tempo especificado como segundo parámetro. Sintaxe:

```
SUBDATE(data, {número_días|INTERVAL expresión unidade_intervalo})
```

Cando se utiliza coa forma INTERVAL no segundo parámetro, dá o mesmo resultado cá función DATE\_SUB.

- **DATEDIFF**

Devolve o número de días transcorridos entre dúas datas que se pasan como parámetros. Sintaxe:

```
DATEDIFF(data_maior, data_menor)
```

- **TIMESTAMPDIFF**

Devolve o número de intervalos de tempo (MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, ou YEAR) que se pasa como primeiro parámetro, transcorridos entre dúas expresións tipo data e hora, ou tipo data. Sintaxe:

```
TIMESTAMPDIFF(unidade_intervalo, data_hora_inicio, data_hora_fin)
```

Exemplos:

```
select adddate(curdate(), interval 5 day), date_add(curdate(), interval 5 day);
```

```
select adddate(curdate(), 5), subdate('2012-11-30', interval 2 year);
```

```
select datediff(curdate(), '2012-12-30'), timestampdiff(year, '2012-12-30',curdate());
```



## Formato de saída para datos tipo data e hora

### ■ DATE\_FORMAT

Aplica un formato de saída a unha expresión tipo data, ou tipo data e hora. Sintaxe:

```
DATE_FORMAT(data, 'cadea_de_formato')
```

A cadea de formato representa a forma en que vai a mostrar a data. Dentro da cadea pódese poñer calquera carácter que será mostrado tal e como aparece na cadea de formato, e ademais, pode levar o carácter % que engade un significado especial a algúns caracteres. Algunhas combinacións de caracteres especiais permitidas son:

%d	Día do mes, con dous díxitos (entre 01 e 31)
%a	Abreviatura do nome do día da semana (en inglés: Sun...Sat)
%W	Nome do día da semana (en inglés: Sunday...Saturday)
%w	Día da semana en cifras (0 = domingo ... 6 = sábado)
%m	Mes, con dous díxitos (entre 01 e 12)
%b	Abreviatura do nome do mes (en inglés: Jan...Dec)
%M	Nome do mes (en inglés: January...December)
%y	Año, con dous díxitos
%Y	Año, con catro díxitos
%h	Hora, con dous díxitos (entre 01 e 12)
%H	Hora, con dous díxitos (entre 00 e 23)
%i	Minutos, con dous díxitos (entre 00 e 59)
%s	Segundos, con dous díxitos (entre 00 e 59)
%r	Hora con formato 12 horas (hh:mm:ss), seguido de AM ou PM
%p	AM ou PM
%%	Mostra o literal %

As sete últimas combinacións de caracteres desta lista tamén poden ser utilizados coa función TIME\_FORMAT.

### ■ TIME\_FORMAT

Aplica un formato a unha expresión tipo hora. Funciona igual que DATE\_FORMAT, pero só pode levar as combinacións especiais de formato para horas, minutos, segundos e microsegundos. Sintaxe:

```
TIME_FORMAT(hora, "cadea_de_formato")
```

### ■ DAY ou DAYOFMONTH

Devolve o día do mes da data que se pasa como parámetro, en número. Sintaxe:

```
DAY(data)
```

```
DAYOFMONTH(data)
```

- **DAYOFWEEK**

Devolve o número de día da semana da data que se pasa como parámetro. Mostra valores numéricos entre o 1 (para o domingo) e o 7 (para o sábado). Sintaxe:

`DAYOFWEEK(data)`

- **MONTH**

Devolve o mes da data especificada. Sintaxe:

`MONTH(data)`

- **YEAR**

Devolve o ano da data especificada. Sintaxe:

`YEAR(data)`

Exemplos:

```
select date_format(curdate(), '%d/%m/%Y'), time_format(curtime(), '%r');
```

```
select date_format(curdate(), 'Lugo a, %d de %m de %Y'), year(now()), month(now());
```

## 2.3.2 Funcións de cadeas de caracteres

- **ASCII**

Devolve o código ASCII do primeiro carácter dunha cadea. Sintaxe:

`ASCII(cadea)`

- **CHAR**

Devolve os caracteres que se corresponden con cada número, segundo a táboa de códigos ASCII. Sintaxe:

`CHAR(número1 [, número2] ...)`

- **HEX**

Devolve a representación en hexadecimal dunha cadea ou dun valor numérico. Cando o parámetro é unha cadea de caracteres, devolve unha combinación de dous díxitos hexadecimais para cada carácter da cadea. Cando o parámetro é un número, devolve a representación do número en hexadecimal. Sintaxe:

`HEX(cadea_caracteres)`

`HEX(número)`

- **CONCAT**

Concatena unha serie de cadeas e devolve unha cadea, ou o valor NULL se algún argumento é NULL. Sintaxe:

`CONCAT(cadea1, cadea2 [,cadea3] ...)`

- **POSITION**

Busca unha subcadea dentro dunha cadea e devolve a posición na que se atopa a primeira aparición. Sintaxe:

`POSITION(subcadea IN cadea)`

- **LOCATE**

Busca unha subcadea dentro dunha cadea e devolve a posición na que se atopa a pri-

meira aparición. Pódese pasar un terceiro parámetro indicando a posición a partir da cal se fai a busca. Sintaxe:

```
LOCATE(subcadea, cadea [, posición])
```

- **LEFT**

Devolve os n primeiros caracteres da cadea, empezando a ler pola esquerda. Sintaxe:

```
LEFT(cadea, n)
```

- **RIGHT**

Devolve os n primeiros caracteres da cadea, empezando a ler pola dereita. Sintaxe:

```
RIGHT(cadea, n)
```

- **LENGTH**

Devolve a lonxitude da cadea en caracteres. Sintaxe:

```
LENGTH(cadea)
```

- **LOWER**

Pasa a minúsculas todos os caracteres da cadea. Sintaxe:

```
LOWER(cadea)
```

- **UPPER**

Pasa a maiúsculas todos os caracteres da cadea. Sintaxe:

```
UPPER(cadea)
```

- **LTRIM**

Elimina os espazos situados á esquerda na cadea. Sintaxe:

```
LTRIM(cadea)
```

- **RTRIM**

Elimina os espazos situados á dereita na cadea. Sintaxe:

```
RTRIM(cadea)
```

- **TRIM**

Elimina os espazos situados á esquerda e á dereita na cadea. Sintaxe:

```
TRIM(cadea)
```

- **REPEAT**

Repite unha cadea n veces. Sintaxe:

```
REPEAT(cadea, n)
```

- **REPLACE**

Substitúe unha subcadea por outra dentro dunha cadea. Sintaxe:

```
REPLACE(cadea, subcadea_inicial, subcadea_final)
```

- **SPACE**

Devolve unha cadea composta de n espazos en branco. Sintaxe:

```
SPACE(n)
```

- **SUBSTRING**

Devolve unha subcadea de lonxitude especificada, empezando dende a posición elixida. Se non se indica lonxitude, se extrae dende a posición indicada ata o final. Sintaxe:

`SUBSTRING(cadea, posición, [lonxitude])`

Exemplos:

```
select length(trim('   sen espazos ')), length('   con espazos '), length(curdate());

select right(curdate(),2), left(curdate(),4), substring(curdate(),6,2);

select upper('Federico'), lower('PASO A MINÚSCULAS') ;

select concat('bos' , ' días ', @nome:='Manuel'), substring('www.google.es',5,6);

select position('r' in 'Federico'), locate('e','Federico',3);

select ascii('a'),char(97),hex(ascii('a')),hex(97),hex(255), hex('abz');
```

### 2.3.3 Funcións numéricas

- **ABS**

Obtén o valor absoluto dun número. Cando se pasa como parámetro un valor que non é numérico devolve o valor 0. Sintaxe:

`ABS(número)`

- **SIGN**

Devolve o valor -1 se o número que se pasa como parámetro é negativo, 1 se é positivo e 0 se o parámetro non é un número, ou é o número 0. Sintaxe:

`SIGN(número)`

- **SQRT**

Devolve a raíz cadrada do número. Sintaxe:

`SQRT(número)`

- **POWER**

Calcula potencias de número. Devolve o resultado de elevar o *número1* á potencia de *número2*. Sintaxe:

`POWER(número1, número2)`

- **MOD**

Devolve o resto da división enteira de dous números que se pasan como parámetros. Sintaxe:

`MOD(número1, número2)`

Outros operadores para obter o resto da división enteira: `número1%número2`.

- **ROUND**

Redondea un número decimal ao enteiro máis próximo. Pódese utilizar un segundo argumento para indicar o número de decimais cos que debe facer o redondeo. Cando *de-*

*cima*is ten un valor negativo, converte en zeros ese número de díxitos contando dende o punto decimal á esquerda e fai o redondeo no seguinte dígito á esquerda. Sintaxe:

ROUND(número [,decimais])

#### ■ CEILING ou CEIL

Devolve o menor valor enteiro, non menor có número que se pasa como parámetro. Sintaxe:

CEIL(número)

CEILING(número)

#### ■ TRUNCATE

Retorna o número truncado co número de decimais que se pasan como parámetro. Cando *decimais* ten un valor negativo, converte en zeros ese número de díxitos contando dende o punto decimal á esquerda. Sintaxe:

TRUNCATE(número, decimais)

#### ■ FLOOR

Devolve o número enteiro máis grande que sexa maior có número que se pasa como parámetro. Sintaxe:

FLOOR(número)

#### ■ RAND

devolve un valor de coma flotante aleatorio no rango  $0 \leq \text{valor} \leq 1.0$ . Sintaxe:

RAND()

RAND(número)

Cando se pasa un número como parámetro produce unha secuencia repetible sempre que se pase ese mesmo número como parámetro.

Exemplos:

```
select abs(-90), abs(10), abs('texto'),abs(0), sign(0),sign(-5),sign(8),sign('texto');
```

```
select sqrt(4), round(sqrt(250),2),power(2,2),power(3,3), mod(56,3);
```

```
select truncate(236.28,1), round(236.28,1),truncate(255.23,-1), round(255.23,-1);
```

```
select floor(rand() * 50); # xera un número de 0 a 49.
```

```
select dni,rand(5) from empregado;
```

Primeira execución			Segunda execución, e posteriores		
Result Grid	Filter Rows:		Result Grid	Filter Rows:	
dni	rand(5)		dni	rand(5)	
33254916	0.40613597483014313	<div> <div>rand(5)</div> <div>→</div> <div>mesma secuencia</div> </div>	33254916	0.40613597483014313	
12852654	0.8745439358749836		12852654	0.8745439358749836	
33123456	0.15431178561813363		33123456	0.15431178561813363	
33456852	0.1479271511993624		33456852	0.1479271511993624	
15245258	0.276700429876056		15245258	0.276700429876056	
33251256	0.9397207265158377		33251256	0.9397207265158377	
33219853	0.8685023736846655		33219853	0.8685023736846655	
33365846	0.5233497196094596		33365846	0.5233497196094596	

```
select dni,rand() from empregado;
```

Primeira execución			Segunda execución	
dni	rand()		dni	rand()
33254916	0.42295907477192496	rand()	33254916	0.19788184873562478
12852654	0.8812412022438284	distinta secuencia	12852654	0.029142507379075984
33123456	0.13732881763701216		33123456	0.5520670214221506
33456852	0.042920512187220636		33456852	0.6729076157071698
15245258	0.8026161387587116		15245258	0.7083370450030426
33251256	0.8843191879655413		33251256	0.5229624086273484
33219853	0.013747554285216661		33219853	0.4898009388612592
33365846	0.41578023826310434		33365846	0.8801174991578958

## 2.3.4 Funcións de agrupamento ou de columna

Permiten facer cálculos coas columnas da táboa de resultados dunha consulta. Cando se utilizan na lista de selección, non poden ir acompañadas de ningunha outra expresión que non conteña unha función deste tipo. Só poden ir acompañadas de nomes de columnas ou de expresións que conteñan nomes de columnas cando se agrupan filas utilizando a cláusula GROUP BY, que se verá nunha actividade posterior.

### ■ COUNT

Conta o número de liñas resultantes da consulta, ou o número de valores distintos que toma unha expresión (normalmente, unha columna). Sintaxe:

COUNT (\*)

COUNT ([DISTINCT] expresión)

- Cando se utiliza o símbolo asterisco (\*) como parámetro, devolve o número de filas que ten a táboa de resultados.
- Cando se pon como parámetro 'DISTINCT expresión', devolve o número de valores distintos que toma a expresión nas filas da táboa de resultados, sen ter en conta as que toman o valor NULL.
- Cando se pon como parámetro só unha 'expresión', devolve o número de filas da táboa de resultados en que a expresión toma un valor distinto de NULL.

Exemplos:

```
select distinct departamento from empregado;
```

departamento
NULL
1
2
3
4
5
6
7
8
9
10

```
select count(*), count(departamento), count(distinct departamento) from empregado;
```

Result Grid		Filter Rows:		Export:
	count(*)	count(departamento)	count(distinct departamento)	
▶	20	19	10	

O resultados destas consultas informan que hai 20 empregados, 19 empregados cun valor distinto de NULL na columna departamento (hai un empregado que non ten departamento asignado) e hai empregados en 10 departamentos distintos numerados do 1 ao 10.

#### ■ SUM

Suma os valores que toma a expresión (normalmente, unha columna) especificada, para todas a filas resultantes da consulta. A opción DISTINCT non ten en conta os valores repetidos da columna. Sintaxe:

`SUM ([DISTINCT] expresión)`

Exemplo: Calcular o que se gasta en salarios. Suma dos salarios brutos.

```
select sum(salario_bruto) from empregado;
```

Result Grid		Filter Rows:	
	sum(salario_bruto)		
▶	921370.00		

#### ■ AVG

Calcula a media dos valores que toma a expresión nas filas da táboa de resultados. A opción DISTINCT non ten en conta os valores repetidos da columna. Sintaxe:

`AVG ([DISTINCT] expresión)`

Exemplo: Calcular a media dos salarios brutos dos empregados.

```
select avg(salario_bruto), round(avg(salario_bruto),2) as media_salario from empregado;
```

Result Grid		Filter Rows:	
	avg(salario_bruto)	media_salario	
▶	46068.500000	46068.50	

#### ■ MAX

Devolve o valor máis alto que toma a expresión nas filas da táboa de resultados. Sintaxe:

`MAX (expresión)`

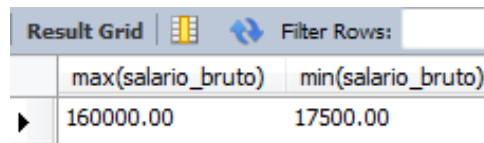
#### ■ MIN

Devolve o valor máis baixo da expresión nas filas da táboa de resultados. Sintaxe:

MIN (expresión)

Exemplo: Seleccionar o salario máis alto e o máis baixo de todos os empregados.

```
select max(salario_bruto), min(salario_bruto) from empleado;
```



	max(salario_bruto)	min(salario_bruto)
▶	160000.00	17500.00

### Valores NULL e as funcións de agrupamento:

O estándar ANSI/ISO establece unhas regras para o manexo de valores NULL nas funcións de columna:

- Se algún dos valores dunha columna é NULL, non se ten en conta ao facer os cálculos nunha función de agrupamento.
- Se o valor de todas as columnas é NULL, as funcións SUM, AVG, MAX e MIN devolven o valor NULL, e a función COUNT o valor cero.
- COUNT(\*) conta filas e non depende da presenza de valores NULL nas columnas. Cando se quere ter en conta os valores NULL dunha columna, pódese utilizar a fórmula COUNT(expresión) que conta as filas nas que expresión non é NULL.

## 2.3.5 Outras funcións

A continuación móstranse de forma moi resumida grupos de funcións que MySQL incorpora que son utilizados con menos frecuencia pero que poden ser de utilidade.

### Funcións de control de fluxo

#### ▪ IF

Examina a *expresión1*; e se é verdadeira (*expresión1* <> 0 e *expresión1* <> NULL) entón retorna *expresión2*; se é falsa, retorna *expresión3*. É posible aníñar condicións, de modo que *expresión2* e *expresión3* poden conter funcións IF. Sintaxe:

```
IF(expresión1, expresión2, expresión3)
```

#### ▪ IFNULL

Cando a expresión1 toma o valor NULL, a función devolve o contido de expresión2, noutro caso devolve o contido de expresión1. Sintaxe:

```
IFNULL(expresión1, expresión2)
```

Exemplos:

```
select if(1>2,2,3), if(1<2,'verdadero','falso'),if(1>2,true,false);
```

```
select apellidos, nome, ifnull(departamento,'Sen departamento')
from empleado
order by departamento;
```



	apelidos	nome	ifnull(departamento,'Sen departamento')
▶	Iglesias Dominguez	Adolfo	Sen departamento
	Fernandez Diaz	Julian	1
	Martinez Iglesias	Benito	1
	Fernandez Lopez	Jose Luis	1
	Nuñez Bernardéz	Antonia	2
	Martinez Diaz	Carlos	2

33 12:13:37 select apelidos, nome, ifnull(departamento,'Sen departament... 20 row(s) returned

0.000 sec / 0.000 sec

## Funcións de información do sistema

### ■ USER

Mostra información do usuario que fixo a conexión (nome e host). Sintaxe:

USER ()

### ■ VERSION

Mostra información da versión do servidor MySQL. Sintaxe:

VERSION ()

Exemplo:

```
select user(), version();
```

	user()	version()
▶	root@localhost	5.6.17

## Funcións de cifrado

As funcións de cifrado ou funcións *'hash'* utilízanse para enmascarar información que se desexa ocultar. Unha función *'hash'* é un algoritmo que transforma un conxunto de datos, como pode ser un ficheiro de texto ou unha cadea de caracteres, nun único valor de lonxitude fixa (*'hash'*). O valor *'hash'* calculado pode ser utilizado para verificar a integridade de copias dun dato orixinal sen necesidade de facilitar o dato orixinal. O proceso de cifrado é practicamente irreversible, polo que un valor *'hash'* pode ser libremente distribuído ou almacenado e só se utiliza para fins de comparación. Traballando con bases de datos, o uso máis corrente é para gardar contrasinais.

### ■ PASSWORD

É a función que usa MySQL para cifrar as contrasinais dos usuarios que acceden ao servidor e que se almacenan na táboa *user* da base de datos *mysql*. Devolve unha cadea de 41 díxitos en hexadecimal. Sintaxe:

PASSWORD (cadea)

### ■ MD5

Calcula unha suma de verificación (checksum) MD5 de 128-bit para unha cadea. Devolve unha cadea de 32 díxitos en hexadecimal. Sintaxe:

MD5 (cadea)

- **SHA1 (Secure Hash Algorithm)**

Calcula unha suma de verificación (checksum) SHA1 de 160-bit para a cadea. O valor se devolve como unha cadea de 40 díxitos en hexadecimal, ou NULL, no caso de que o parámetro que se pasa tivera o valor NULL. Pode considerarse un equivalente a MD5(), criptograficamente máis seguro. Sintaxe:

```
SHA1(cadea)
```

- **SHA2**

É un conxunto de funcións 'hash' criptográficas (SHA-224, SHA-256, SHA-384, SHA-512) deseñadas pola 'Agencia de Seguridad Nacional' (NSA). SHA-2 inclúe un importante número de cambios respecto a súa predecesora, SHA-1, que a fai máis segura. Devolve unha cadea en hexadecimal de lonxitude 56, 64, 96, ou 128 caracteres, dependendo da función utilizada. Sintaxe:

```
SHA2(cadea, lonxitude_resultado)
```

O primeiro parámetro que se pasa é unha cadea de texto plano, e o segundo indica o tipo de función que se vai a aplicar, e pode tomar os valores 224, 256, 384, 512, ou 0 (que é equivalente a 256). Se algún dos parámetros que se lle pasa toma o valor NULL ou o valor que se pasa como segundo parámetro non é un dos valores permitidos, devolve o valor NULL.

Exemplos:

```
select password('abc'), length(password('abc'));
```

```
select md5('abc'), length(md5('abc'));
```

```
select sha1('abc'), length(sha1('abc'));
```

```
select sha2('abc',224),length(sha2('abc',224));
```

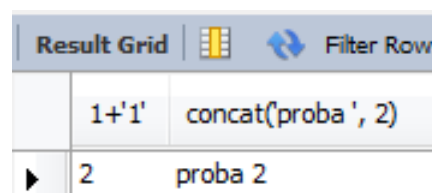
```
select sha2('abc',384),length(sha2('abc',384));
```

## Funcións de conversión de tipos

Unha expresión ou unha comparación, poden ter operandos de distintos tipos que terán que converterse en tipos compatibles para que se resolva. Algunhas conversións fanse de forma implícita, como por exemplo, a conversión automática de números en cadeas, e viceversa, que fai MySQL.

Exemplo:

```
select 1+'1', concat('proba ', 2);
```



	1+'1'	concat('proba ', 2)
▶	2	proba 2

As funcións CAST e CONVERT permiten facer conversións explícitas dun tipo de dato a outro, indicando o tipo de dato no que se quere converter. Con estas funcións soluciónase de forma fácil un problema que pode chegar a ser un quebracabezas, sobre todo nas comparacións. Os tipos de datos que se poden utilizar para a conversión son:

- BINARY[(tamaño)]
- CHAR[(tamaño)]
- DATE
- DATETIME
- DECIMAL[(tamaño[,decimais])]
- SIGNED [INTEGER]
- TIME
- UNSIGNED [INTEGER]

As funcións CAST e CONVERT que incorpora MySQL, utilizan a sintaxe SQL estándar.

#### ■ CAST

Devolve o valor que se pasa como primeiro parámetro convertido ao tipo que se indica na función. Sintaxe:

```
CAST(expresión AS tipo_dato)
```

As funcións de conversión son útiles, por exemplo, para ordenar os resultados por columnas de tipo ENUM por orden alfabético. Normalmente cando se ordena por unha columna ENUM, utilízanse os valores numéricos do orden interno. Facendo unha conversión a tipo CHAR pódese facer a ordenación por orden alfabético.

Exemplo: seleccionar *codigo*, *nome* e *tipo* de todos os departamentos, ordenando o resultado alfabeticamente pola columna *tipo*, que é de tipo ENUM como se pode ver na descrición do esquema da táboa.

Column	Type
◇ codigo	tinyint(3) unsigned
◇ nome	char(25)
◇ tipo	enum('H','B','A')
◇ cidade	varchar(35)
◇ id_provincia	smallint(6)

A primeira solución ordena polo tipo seguindo o número de orde dos valores válidos (H, B, A).

```
select codigo, nome, tipo
from departamento
order by tipo;
```

	codigo	nome	tipo
▶	1	Central	H
	2	Oficina1	H
	8	Oficina7	H
	4	Oficina3	H
	7	Oficina6	H
	10	Oficina9	B
	3	Oficina2	B
	5	Oficina4	A
	9	Oficina8	A
	6	Oficina5	A
*	NULL	NULL	NULL

A segunda solución ordena alfabeticamente polo valor do tipo transformado en carácter (A, B, H).

```
select codigo, nome, tipo
from departamento
order by cast(tipo as char);
```

	codigo	nome	tipo
▶	6	Oficina5	A
	9	Oficina8	A
	5	Oficina4	A
	10	Oficina9	B
	3	Oficina2	B
	4	Oficina3	H
	7	Oficina6	H
	8	Oficina7	H
	2	Oficina1	H
	1	Central	H
*	NULL	NULL	NULL

## ■ CONVERT

Transforma unha expresión nun tipo de dato indicado como parámetro. Sintaxe:

CONVERT(expresión, tipo)

CONVERT(expresión USING nome\_xogo\_carácteres)

- Cando se utiliza a primeira forma da sintaxe, funciona igual que a función CAST, converte a expresión ao tipo de datos que se pasa como segundo parámetro.
- A segunda forma da sintaxe, con USING, utilízase para converter datos entre diferentes xogos de caracteres.

Exemplo: Converter a cadea 'abc' ao xogo de caracteres utf8:

```
select convert('abc' using utf8);
```