# Introdución á programación en BD

# Índice


1. **Programas e rutinas almacenadas ..... 3**
   - 1.1 Creación de programas almacenados ..... 3
   - 1.2 Bloques de programación ..... 3
   - 1.3 Delimitadores de final de sentenza ..... 4
   - 1.4 Parámetros ..... 4
   - 1.5 Execución de programas almacenados ..... 4
2. **Sentenzas básicas para a programación con MySQL ..... 5**
   - 2.1 Sentenzas de declaración e manexo de variables e manipuladores. ..... 5
     - DECLARE ..... 5
       - Declaración de variables ..... 5
       - Declaración de condicións ..... 5
       - Declaración de manipuladores. Manexo de excepcións ..... 6
     - SET ..... 8
     - SELECT ... INTO ..... 8
   - 2.2 Estruturas de control de fluxo ..... 9
     - Sentenza condicional IF ..... 9
     - Sentenza alternativa CASE ..... 10
     - Sentenza repetitiva bucle WHILE ..... 11
     - Sentenza repetitiva bucle REPEAT ..... 11
   - 2.3 Sentenzas preparadas en SQL ..... 12

# 1. Programas e rutinas almacenadas

As ampliacións da linguaxe SQL permiten crear e almacenar no lado do servidor de bases de datos varios tipos de obxectos:

- Procedementos almacenados. Conxunto de sentenzas que permiten automatizar tarefas que poden facer cálculos, ou xerar conxuntos de resultados.
- Funcións definidas polos usuarios (UDF). Conxunto de sentenzas que devolven sempre o resultado dun cálculo, e poden utilizarse en expresións, igual que as funcións propias de SQL.
- Disparadores (*triggers*). Conxunto de sentenzas que se executan de forma automática cando se fai unha determinada operación de manipulación de datos.
- Eventos. Permiten a execución diferida dun conxunto de sentenzas, tendo en conta un calendario establecido.

MySQL introduce a compatibilidade con funcións definidas polo usuario e procedementos almacenados, na versión 5.0; cos disparadores, na versión 5.0.2; e con eventos, na versión 5.1.6. Polo tanto, antes de utilizar estas funcionalidades hai que comprobar a versión coa que se está traballando.

**Programas almacenados:** Este termo fai referencia a todos os tipos de obxectos almacenados no servidor: procedementos, funcións, disparadores, e eventos.

**Rutinas almacenadas:** Este termo só fai referencia aos procedementos almacenados e as funcións. Estes obxectos defínense cunha sintaxe moi similar, e a súa execución non é automática como sucede cos disparadores e os eventos.

**Subrutina:** Este termo emprégase para facer referencia a unha rutina utilizada dentro dun programa almacenado.

## 1.1 Creación de programas almacenados

Para crear un programa almacenado en SQL é necesario escribir un guión (*script*) de sentenzas SQL utilizando un editor, e gardalo nun ficheiro con extensión .sql.

A execución do guión de sentenzas SQL realiza a compilación do programa almacenado e a creación do obxecto correspondente (procedemento, función, disparador, ou evento) no servidor.

As sentenzas para crear, borrar e modificar programas almacenados son:

```
CREATE PROCEDURE        DROP PROCEDURE        ALTER PROCEDURE
CREATE FUNCTION         DROP FUNCTION         ALTER FUNCTION
CREATE TRIGGER          DROP TRIGGER
CREATE EVENT            DROP EVENT            ALTER EVENT
```

## 1.2 Bloques de programación

Os programas almacenados poden conter unha ou máis sentenzas para facer o seu traballo. No caso de conter varias sentenzas, estas pódense agrupar en bloques de programacións, tamén chamados sentenzas compostas, que son un conxunto de sentenzas SQL que resolven un problema concreto, e que empezan cunha sentenza *begin*, e rematan cunha sentenza *end*.

```
BEGIN
   [lista_sentenzas]
END;
```

- A parte *lista_sentenzas* é unha lista dunha ou máis sentenzas SQL.
- Se o bloque ou programa se compón dunha única instrución non é obrigatorio utilizar as sentenzas *begin* e *end*.
- Cada sentenza que forma o bloque ten que rematar en punto e coma (;) que é o carácter delimitador que indica o final dunha sentenza SQL.

## 1.3 Delimitadores de final de sentenza

Un problema do uso de sentenzas compostas é que dentro do bloque é necesario separar as sentenzas co delimitador de final de sentenza, que de forma predeterminada é o carácter punto e coma. Isto produce un conflito cando se crean programas almacenados xa que no caso de que o servidor reciba o primeiro carácter punto e coma considera que a sentenza de creación do programa almacenado remata, e non tería en conta as seguintes instrucións que compoñen o programa almacenado.

Para solucionar este conflito hai que utilizar a sentenza DELIMITER que permite cambiar o carácter delimitador de final sentenza por una carácter ou combinación de carácteres que non se utilice na creación do programa almacenado (por exemplo //). Unha vez creado o programa pódese utilizar esta mesma instrución para volver a definir o punto e coma como delimitador de final de sentenza. Sintaxe:

```
DELIMITER carácter_final_sentenza
```

Exemplo de guión para crear un procedemento almacenado:

```
-- Hola mundo, en SQL
delimiter //                    # Cambia o carácter delimitador de fin de sentenza
create procedure holaMundo()    # Dálle nome ao procedemento
begin                           # Inicia o bloque de programación
  select 'hola mundo';          # Sentenzas do bloque de programación
end;                            # Finaliza o bloque de programación
//                              # Finaliza a sentenza create procedure
delimiter ;                     # Deixa o delimitador de fin de sentenza como estaba
```

## 1.4 Parámetros

Algunhas rutinas almacenadas necesitan que se lles proporcione algún dato para poder facer o seu traballo; estes datos chámanse parámetros de entrada. Ademais pode ocorrer que a rutina devolva algún valor unha vez rematada á execución; estes datos que devolve a rutina chámanse parámetros de saída. Para diferencialos das variables e dos nomes de columnas pode ser útil poñerlles nomes que empecen por p. Exemplos: *pDNI*, ou *pNome*.

## 1.5 Execución de programas almacenados

Unha das diferenzas máis importantes nos distintos tipos de programas almacenados é a forma en que se executan:

- **Procedemento almacenado**. Execútase facendo unha chamada (***call***) ao servidor indicando o nome do procedemento e pasándolle, opcionalmente, os parámetros necesarios.
- **Función definida polo usuario**. Utilízase igual que as funcións que xa existen en SQL. Non se executa cunha chamada explícita como os procedementos, se non que se utiliza como parte dunha expresión nas sentenzas SQL.

- **Disparador**. Está asociado sempre a unha operación de manipulación de datos (inserción, modificación ou borrado de filas) sobre unha táboa, e execútase de forma automática cando se realiza esa operación.
- **Evento**. Na creación do evento **indícase en que momento se ten que executar**, e esa información queda almacenada no servidor; cando chega ese momento, o servidor o executa de forma automática.

Exemplo de execución dun procedemento almacenado:

```
call holaMundo();
```

Resultado da execución:


# 2. Sentenzas básicas para a programación con MySQL

MySQL só permite utilizar a maioría das sentenzas básicas de programación na creación de programas almacenados e non o permite fóra deste contexto. Outros SXBDR non teñen esta limitación.

## 2.1 Sentenzas de declaración e manexo de variables e manipuladores.

### DECLARE

Permiten declarar variables, condicións de erro e manipuladores de erros. As sentenzas de declaración teñen que ir sempre ao inicio do bloque de programación, despois da sentenza *begin*, e antes de escribir calquera outra sentenza. **As sentenzas de declaración hai que escribilas neste orden: primeiro as variables, despois as condicións, e por último os manipuladores**.

#### Declaración de variables

Permite definir variables locais do programa. A sintaxe é:

```
DECLARE nome_variable [,…] tipo_dato [DEFAULT valor]
```

- O tipo de dato pode ser calquera dos utilizados na definición de columnas nas táboas.
- A cláusula DEFAULT permite asignarlle un valor á variable no momento da creación.

#### Declaración de condicións

Permite asignar un nome a unha condición de erro relacionada cun determinado código de erro, ou cun estado SQL (*sqlstate*). Utilízase para condicións de erro que precisan dun tratamento especial. Os nomes de condición pódense utilizar na declaración de manipuladores.

A sintaxe é:

```
DECLARE nome_condición  CONDITION  FOR  valor_condición
```

Onde *valor_condición* pode ser:

```
SQLSTATE [VALUE] codigo_sqlstate  |  código_erro_mysql
```

Os códigos de erro e valores de estado (SQLSTATE) xunto coa explicación do seu significado, pódense consultar no anexo correspondente do manual de referencia de MySQL (Apéndice B, sección 3, no Manual de referencia de MySQL 5.6). Algún exemplo dos códigos de erro e valores de estado máis utilizados:

| Código erro MySQL | sqlstate | Descrición / Mensaxe |
|---|---|---|
| 1329 | 02000 | Sen datos / 0 row(s) affected, 1 warning(s): 1329 No data - zero rows fetched, selected, or processed |
| 1062 | 23000 | Clave duplicada / ERROR 1062:Entrada duplicada '27111444' para a clave 'PRIMARY' |
| 1136 | 21S01 | Nº de columnas e de valores non coincide en INSERT / ERROR 1136. O número de columnas non se corresponde co número na liña 1 |
| 1146 | 42S02 | Non existe a táboa / ERROR 1146. Táboa 'test.empregado' non existe |

Os códigos SQLSTATE son independentes do SXBDR co que se traballe. Por tanto, en Oracle, SQL Server, MySQL, etc. hai os mesmos códigos de erro SQLSTATE, o que fai que o código sexa máis portable. Por contra, os códigos de erro de MySQL son propios, e non teñen nada que ver cos códigos de erro doutros SXBDR.

Exemplo de declaración de condición:

```
--  Facendo referencia ao código de estado (entre comiñas)
declare claveDuplicada condition  for sqlstate '23000';
--  Tamén se podería facer referencia ao código de erro MySQL (sen comiñas)
declare claveDuplicada2 condition  for 1062;
```

A comprobación da sintaxe das ordes de declaración de condición, pódese facer creando un procedemento almacenado cun único bloque de programación:

```
drop procedure if exists condicion;
delimiter //
create procedure condicion()
begin
--  Facendo referencia ao código de estado (entre comiñas)
declare claveDuplicada condition  for sqlstate '23000';
--  Tamén se podería facer referencia ao código de erro MySQL (sen comiñas)
declare claveDuplicada2 condition  for 1062;
end;
//
delimiter ;
```

## Declaración de manipuladores. Manexo de excepcións

Indica as accións que hai que executar no caso que se produza un erro determinado na execución dun programa almacenado. Sintaxe:

```
DECLARE acción_manipulador HANDLER FOR valor_condición [,…] lista_sentenzas
```

Algunhas notas sobre a sintaxe:

- O valor para *acción_manipulador* pode ser:

  ```
  CONTINUE | EXIT
  ```

  – A opción EXIT, utilízase para erros graves, e produce a interrupción da execución do bloque de programación no momento que se produce o erro.

  – A opción CONTINUE, utilízase para erros leves que permiten que a execución poida continuar e que no propio código se decidan as accións a tomar no caso de que se produza o erro.

- O *valor_condición* especifica a condición ou clase de condicións que activan o manipulador, e pode ser:

```
SQLSTATE [VALUE] codigo_sqlstate | código_erro_mysql | nome_condición | SQLWARNING | NOT FOUND| SQLEXCEPTION
```

- Os códigos de erro de MySQL, os valores de estado (*sqlstate*), e a explicación do seu significado, pódense consultar no correspondente manual de referencia de MySQL https://dev.mysql.com/doc/mysql-errors/8.0/en/server-error-reference.html
- O *nome_condición* é un nome creado anteriormente nunha declaración de condición.
- SQLWARNING é unha abreviación para todos os códigos de estado que empezan por 01.
- NOT FOUND é unha abreviación para todos so códigos de estado que empezan por 02. Utilízanse no manexo de cursores.
- SQLEXCEPTION é unha abreviación para todos os códigos de estado que non son tratados por SQLWARNING e NOT FOUND.

- A parte *lista_sentenzas* pode conter unha ou máis sentenzas SQL que se van a executar no caso de producirse a condición de erro asociada ao manipulador. No caso de ter máis dunha sentenza hai que utilizar BEGIN e END.

A acción asociada a un manipulador está en función da clase de condición que ten asociada. No caso de SQLEXCEPTION debería ser EXIT, e no caso de SQLWARNING e NOT FOUND debería ser CONTINUE.

A declaración de manipuladores ten que ir sempre nun bloque de programación, situada despois da declaración de variables e de condicións, e antes de empezar calquera outra sentenza distinta das sentenzas de declaración.

Exemplo de declaración dun manipulador que utiliza o nome de condición *claveDuplicada*, como o creado no exemplo anterior, que cando se produce o erro asociado a esa condición, asígnalle o valor 1 á variable *vFinal*:

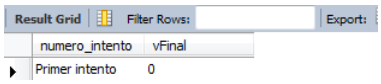
```
declare vFinal bit default 0;
declare claveDuplicada condition  for sqlstate '23000';
declare continue handler for claveDuplicada set vFinal = 1;
```

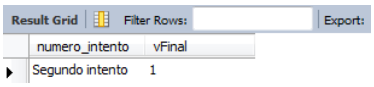
A comprobación da sintaxe das ordes de declaración e o funcionamento dos manipuladores, pódese facer creando un procedemento almacenado cun único bloque de programación no que se declara a condición, o manipulador, e unha variable tipo interruptor chamada *vFinal* que cando se crea toma o valor 0, e cando se produce a condición de erro asígnaselle o valor 1 :

```
drop procedure if exists manipuladorDemo;
delimiter //
create procedure manipuladorDemo()
begin
declare vFinal bit default 0;
declare claveDuplicada condition  for sqlstate '23000';
declare continue handler for claveDuplicada set vFinal = 1;
insert into utilidades.provincia values ('27','Lugo');
select 'Primer intento ',vFinal;
insert into utilidades.provincia values ('27','Lugo');
select 'Segundo intento ',vFinal;
end;
//
delimiter ;
call manipuladorDemo();
```

Se non existe ningunha provincia co código 27 a variable *vFinal* toma o valor 0 e no caso de que xa exista unha provincia con ese código toma o valor 1.

Resultado da execución:




## SET

Permite asignar valores ás variables que se manexan no procedemento. A sintaxe é:

```
SET nome_variable = expresión   [,nome_variable = expresión] …
```

Tipos de variables que se poden utilizar nos programas almacenados:

- **Variables locais**. Decláranse dentro dun bloque de código SQL coa sentenza *declare*. O seu valor só se pode ver dentro do bloque e elimínase cando remata o bloque. O seu nome non leva ningún carácter especial ao inicio, aínda que para diferencialas dos nomes de columnas recoméndase que empecen por unha letra 'v'. Exemplo:

  ```
  set vNumero = 0.
  ```

- **Variables de usuario**. Poden crearse, verse e modificarse dentro ou fóra do procedemento e son visibles mentres non se pecha a sesión do usuario. Para crealas basta con asignarlle un valor coa orde *set*, e non é necesario declaralas previamente. O seu nome leva diante o símbolo @. Exemplo:

  ```
  set @mes = 3.
  ```

  A este tipo de variables tamén se lle poden asignar valores utilizando o operador := nunha sentenza SELECT . Exemplo:

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

- **Variables de sistema**. Son variables que manexa o SXBD para configurar o servidor. Créanse e asignáselles un valor no momento de iniciar o servidor. Pódese cambiar o seu valor a nivel *global* ou *session*. Cando se asigna o valor a nivel *global*, ese será o valor que teña a variable para todas as sesións que se inicien a partir dese momento, e cando se asigna a nivel *session* o valor só cambia para a sesión actual. O seu nome leva diante dous símbolos @. Fóra dun bloque de programación non é necesario poñer diante os dous símbolos @ aos nomes das variables de sistemas; faise dentro dos programas almacenados para distinguilas das variables locais e de usuario. Exemplo:

  ```
  set session @@foreign_key_checks = 0.
  ```

## SELECT ... INTO

É outra maneira de asignar valores ás variables, tomando como entrada o resultado dunha sentenza *select*. Permite almacenar nunha variable os valores das columnas do resultado dunha consulta que só devolve unha fila. A sintaxe é:

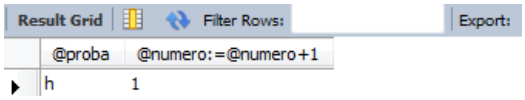
```
SELECT nome_columna [,...] INTO nome_variable [,... ]
```

Utilízanse cursores para manexar o resultado dunha consulta que devolve máis dunha fila.

Exemplo de asignación de valores coa sentenza SELECT:

```
set @proba = 'x';
select @numero:=0;
select sexo into  @proba from practicas1.empregado where dni='33258458K';
select @proba,@numero:=@numero+1;
```

Resultado da execución:




## 2.2 Estruturas de control de fluxo

SQL permite utilizar, como calquera linguaxe de programación, unha serie de sentenzas de control de fluxo para poder escribir rutinas que teñan certa complexidade.

### Sentenza condicional IF

Permite seleccionar qué sentenzas executar en función dunha condición.

```
IF condición1 THEN lista_sentenzas1
   [ELSEIF condición2 THEN lista_sentenzas3]  …
   [ELSE lista_sentenzas2]
END IF
```

O resultado é:

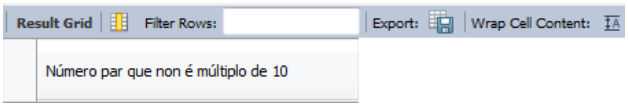
- Se a *condición1* é verdadeira, execútanse as sentenzas contidas en *lista_sentenzas1*.
- Se a *condición1* é falsa,
  - e existe a parte *else*, execútanse as sentenzas contidas en *lista_sentenzas2*.
  - e existe a parte *elseif*, avalíase a *condición2*: se é verdadeira, execútanse a sentenza ou sentenzas contidas en *lista_sentenzas3*.
  - e non existe *else* nin *elseif*, execútase a sentenza que apareza despois de END IF.

Está permitido o uso de sentenza *if* anidadas, e dicir, unha sentenza *if* pode conter outras sentenzas *if*. Neste caso cada sentenza *if* ten que levar a correspondente *end if*.

Exemplo que mostra se a cantidade almacenada na variable *vNumero*, é par e ademais múltiplo de 10, ou par e ademais non múltiplo de 10, ou é impar.

```
drop procedure if exists evaluaNumero;
delimiter //
/* Escribir un bloque de programación que mostre se a cantidade almacenada na variable vNumero, é par e ademais múltiplo de 10, ou par e ademais non múltiplo de 10 ou é impar.*/
create procedure evaluaNumero()
begin
declare vNumero integer;
set vNumero = 92;
if vNumero % 2 = 0 then
   if vNumero % 10 = 0 then select 'Número par e múltiplo de 10';
   else select 'Número par que non é múltiplo de 10';
   end if;
else select 'Número impar';
end if;
end ;
//
delimiter ;
call evaluaNumero();
```

Resultado da execución:




## Sentenza alternativa CASE

Permite executar unha lista de sentenzas, dependendo do valor que toma unha expresión.

```
CASE  expresión
   [WHEN valor  THEN   lista_sentenzas1] …
   [ELSE  lista_sentenzas2]
END CASE
```

Avalía o resultado que devolve a expresión e cando toma o valor que vai despois da cláusula *when*, executa as instrucións contida en *lista_sentenzas1*. Pódense poñer as cláusulas *when* que se queiran, e ademais, pódese utilizar a cláusula *else* para que se execute a *lista_sentenzas2* no caso de que expresión tome un valor diferente dos relacionados nas cláusulas *when*.

Está permitido utilizar esta sentenza fóra da creación de programas almacenados. Exemplo de uso dentro dunha consulta:

```
-- Exemplo de uso nunha consulta con sentenza select
select dni, nome,
   case sexo
    when 'h' then 'home'
    when 'm' then 'muller'
    end as sexo
from empregado;
```

Exemplo dentro dun bloque de programación:

```
-- Exemplo de uso dentro dun bloque de programación
delimiter //
create procedure demoCase()
begin
declare vSexo char(1) default null;
set vSexo = 'm';
case vSexo
  when 'h' then select 'home';
  when 'm' then select 'muller';
  else select 'erro';
end case;
end;
//
delimiter ;
call demoCase();
drop procedure demoCase;
```

Resultado da execución:

## Sentenza repetitiva bucle WHILE

Repite un bloque de sentenzas, mentres se cumpra unha condición.

```
[etiqueta_inicio:] WHILE condición  DO
   lista_de_sentenzas
END  WHILE [etiqueta_fin]
```

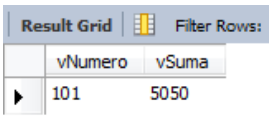
As sentenzas incluídas en *lista_de_sentenzas* vanse a executar un número indeterminado de veces, mentres a condición que vai despois de WHILE sexa verdadeira, e deixarán de executarse cando a condición sexa falsa.

Exemplo que calcula a suma dos 100 primeiros número naturais empregando un bucle *while*.

```
-- suma dos 100 primeiros números naturais
drop procedure if exists suma100;
delimiter //
create procedure suma100()
  begin
  declare vNumero tinyint default 1;
  declare vSuma smallint unsigned default 0;
  bucle: while (vNumero <= 100)  do
    set vSuma = vSuma + vNumero;
   set vNumero = vNumero+1;
  end while bucle;
  select vNumero, vSuma;
  end;
//
delimiter ;
call suma100();
```

O bucle leva un nome de etiqueta para mostrar como é a sintaxe de etiquetas, aínda que neste exemplo non ten ningunha utilidade. O uso de etiquetas ten utilidade cando se necesita facer referencia ao bucle nalgunha parte do programa.

Resultado da execución:




## Sentenza repetitiva bucle REPEAT

Repite unha ou máis sentenzas, e para de executalas cando se cumpre unha condición.

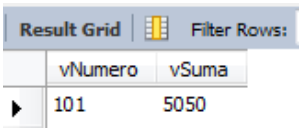
```
[etiqueta_inicio:] REPEAT
   lista_de_sentenzas
UNTIL condición
END REPEAT[etiqueta_fin]
```

As sentenzas incluídas en *lista_de_sentenzas* vanse a executar un número indeterminado de veces, ata que a condición que vai despois de UNTIL sexa verdadeira.

Exemplo que calcula a suma dos 100 primeiros números naturais empregando un bucle *repeat..until*.

```
-- suma dos 100 primeiros números naturais
drop procedure if exists suma100repeat;
delimiter //
create procedure suma100repeat()
  begin
  declare vNumero tinyint default 1;
  declare vSuma smallint unsigned default 0;
  repeat
    set vSuma = vSuma + vNumero;
   set vNumero = vNumero+1;
  until vNumero > 100
  end repeat;
  select vNumero,vSuma;
  end;
//
delimiter ;
call suma100repeat();
```

Resultado da execución:




## 2.3 Sentenzas preparadas en SQL

A utilización de sentenzas preparadas en programas almacenados e nas aplicacións, permiten introducir parámetros na construción das sentenzas, que son substituídos polos valores que se lles asignan no momento da execución.

Isto supón menos sobrecarga para analizar a sentenza cada vez que se executa. Normalmente as aplicacións de bases de datos manexan moitas sentenzas case idénticas, con pequenos cambios en valores constantes ou variables en cláusulas como: *where* para o caso de consultas, *set* no caso de modificacións, ou *values* no caso de inserción.

A Sintaxe SQL para sentenzas preparadas basease na utilización de tres sentenzas:

- A sentenza PREPARE permitir preparar unha sentenza e asignarlle un nome para facer referencia a ela posteriormente para executala ou borrala. Sintaxe:

  ```
  PREPARE nome_sentenza_preparada FROM sentenza_preparada;
  ```

  A parte *sentenza_preparada* é unha cadea de texto ou unha variable de usuario que contén o texto da sentenza. O texto debe representar unha única sentenza. Dentro da sentenza poden utilizarse carácteres *?* como marcadores da posición dos parámetros que van a recibir os valores no momento da execución. Os carácteres *?* non deben delimitarse con comiñas, aínda que representen cadeas de texto.

  No caso de existir unha sentenza almacenada con ese mesmo nome, elimínase antes de crear a nova sentenza preparada.

- A sentenza EXECUTE permite executar unha sentenza preparada. Sintaxe:

  ```
  EXECUTE nome_sentenza_preparada [USING @nome_variable [,@nome_variable] ...];
  ```

  Se a sentenza preparada contén algún carácter ? como marcador da posición de parámetros, hai que engadir unha cláusula *using* que asigne os valores correspondentes aos parámetros. Teñen que existir tantos valores como marcadores de parámetros.

  Pódese executar unha sentenza preparada varias veces, pasando distintas variables, ou

asignando ás variables valores distintos en cada execución.

- A sentenza DEALLOCATE ou DROP permite eliminar unha sentenza preparada. Sintaxe:

```
{DEALLOCATE | DROP} PREPARE nome_sentenza_preparada;
```

MySQL 5.6 soporta a utilización de sentenzas preparadas no lado do servidor. Non se pode utilizar calquera sentenza SQL en sentenzas preparadas. Para máis información sobre as sentenzas permitidas en sentenzas preparadas, débese consultar o manual de referencia de

MySQL. Relación resumida das sentenzas permitidas:

```
ALTER TABLE
ANALYZE TABLE
CALL
{CREATE | RENAME | DROP} DATABASE
{CREATE | DROP} INDEX
{CREATE | RENAME | DROP} TABLE
{CREATE | RENAME | DROP} USER
{CREATE | DROP} VIEW
DELETE
INSERT
OPTIMIZE TABLE
REPAIR TABLE
SELECT
UPDATE
```

O alcance para unha sentenza preparada é a sesión na que se crea, e non está dispoñible para outras sesións. Cando se utiliza sentenzas preparadas nun programa almacenado, no caso de que non se borren dentro do programa, seguen existindo aínda que finalice a execución do programa e poden ser executadas posteriormente fóra do programa. Por esa razón non se poden utilizar parámetros ou variables locais do programa almacenado na definición da sentenza preparada,  e no seu lugar hai que utilizar sempre variables de usuario.

Exemplos de sentenzas preparadas:

- Crear e executar unha sentenza preparada que mostre a suma de dous números que están gardados nas variables de usuario *n1* e *n2*.

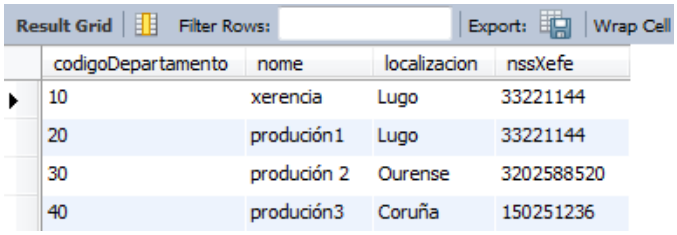
```
-- Mostrar a suma de dous números contidos en variables de usuario
use test;
set @n1=9;
set @n2=4;
prepare suma from 'select ? + ? as suma';
execute suma using @n1,@n2;
deallocate prepare suma;
```

Resultado da execución:

- Crear e executar unha sentenza preparada que mostre todos os datos dunha táboa da base de datos *utilidades*. O nome da táboa da que se van a mostrar os datos está almacenado nunha variable chamada *nomeTaboa*.

```
-- Mostrar datos dunha táboa. Nome da táboa en variable de usuario
use practicas1;
set @nomeTaboa = 'departamento';
set @sentenza = concat('select * from ', @nomeTaboa);
prepare datosTaboa from @sentenza;
execute datosTaboa;
deallocate prepare datosTaboa;
```

Resultado da execución:




| codigoDepartamento | nome | localizacion | nssXefe |
|---|---|---|---|
| 10 | xerencia | Lugo | 33221144 |
| 20 | produción1 | Lugo | 33221144 |
| 30 | produción 2 | Ourense | 3202588520 |
| 40 | produción3 | Coruña | 150251236 |