

1.1.1 Bases de datos de traballo

As bases de datos *practicar1*, *traballadores*, *utilidades* e *tendaBD*, utilizaranse nalgúns exemplos e tarefas desta actividade. Antes de empezar a probar os exemplos ou realizar as tarefas, hai que executar os scripts de creación no servidor e poñer en uso as bases de datos cando corresponda. Os scripts atópanse no cartafol anexo a esta actividade descrito no apartado '3.3 Material auxiliar'.

1.2 Actividade

1.2.1 Rutinas almacenadas en MySQL

Unha rutina almacenada é un conxunto de sentenzas que poden ser almacenadas no servidor cun nome que se lle asigna no momento da creación. As aplicacións cliente poden executar as rutinas almacenadas, indicando o nome da rutina e pasándolle, opcionalmente, os parámetros necesarios. O termo rutinas almacenadas fai referencia tanto aos procedementos almacenados como ás funcións definidas polo usuario.

Cando se crea unha rutina almacenada, o conxunto de instrucións almacenadas no servidor queda enlazado e optimizado para a súa execución, o que supón unha mellora no rendemento e unha redución do tráfico na rede, aínda que a cambio supón un aumento de carga de traballo no servidor de bases de datos.

As rutinas almacenadas (procedementos e funcións) son soportadas por MySQL a partir da versión 5.0, e segue a sintaxe da norma ANSI SQL:2003 para procedementos almacenados, igual que IBM DB2.

1.2.1.1 Creación de rutinas almacenadas

O proceso de creación dunha rutina almacenada fai que quede asociado á base de datos que estea activa nese momento ou á que se indica explicitamente cun nome cualificado como *nomeBD.nomeRutina*. Isto ten varias implicacións:

- Cando se queiran utilizar o procedemento almacenado ou a función, ten que estar activa a base de datos asociada ou ben hai que cualificar o nome da rutina co nome da base de datos. Exemplo: *utilidades.calculoNota()*, ou *utilidades.letraDNI()*, para facer referencia ás rutinas *calculoNota()* e *letraDNI()* que están na base de datos *utilidades*.
- Nunha base de datos non pode haber dúas rutinas almacenadas que teñan o mesmo nome.
- Cando se borra unha base de datos, se borran todos os procedementos almacenados e todas as funcións asociadas a esa base de datos.

No caso de existir un conxunto de procedementos almacenados e de funcións definidas polo usuario que podan ser de utilidade en máis dunha base de datos, pode ser útil ter creada unha base de datos cunha librería de procedementos almacenados e de funcións definidas polo usuario, en lugar de crear eses procedementos ou funcións en cada unha das bases de datos nas que se vai a utilizar.

Nesta actividade, utilizarase unha base de datos chamada *utilidades* cunha serie de táboas con sistemas de codificación (*provincias*, *países*, ...) e unha librería de rutinas almacenadas de uso xeral que poden ser utilizados dende calquera base de datos mediante nomes cualificados. Exemplo: *utilidades.calculo_cif()*.

A información das rutinas creadas gárdase no diccionario de datos, igual có resto de obxectos das bases de datos. No caso de MySQL, a información sobre as rutinas almacenadas pódese consultar nas táboas *mysql.proc* e *information_schema.routines*.

Sentenzas CREATE PROCEDURE e CREATE FUNCTION

Estas sentenzas permiten crear procedementos almacenados e funcións definidas polo usuario. Sintaxe:

```
CREATE [DEFINER = { usuario | CURRENT_USER }]
  PROCEDURE nome_procedemento ( [parámetro_procedemento [...]] )
  [característica ...] corpo_rutina
CREATE [DEFINER = { usuario | CURRENT_USER }]
  FUNCTION nome_función ( [parámetro_función [...]] )
  RETURNS tipo_dato
  [característica ...] corpo_rutina
```

- As cláusulas DEFINER e SQL SECURITY (forma parte de *característica*) especifican o contexto de seguridade no momento da execución da rutina. Con DEFINER pódese indicar o nome do usuario que vai ser considerado o creador da rutina. Se non se especifica nada, tómasse CURRENT_USER que fai referencia ao usuario actual que está creando a rutina.
- Os parénteses que van despois do nome do procedemento son obrigatorios e conteñen a lista de parámetros. Non se escribe nada dentro deles se o procedemento non ten parámetros.
- *parámetro_procedemento* ten a forma:

[IN | OUT | INOUT] nome_parámetro tipo_dato

- IN, OUT e INOUT é o tipo de parámetro.

Un parámetro de entrada (IN - input) indica que cando se chame ao procedemento almacenado, hai que escribir nesa posición un valor que se corresponda co tipo de dato asociado a ese parámetro. O parámetro pode cambiar de valor durante a execución do procedemento, pero o seu valor non é visible para quen o chama.

Un parámetro de saída (OUT - output) indica que nesa posición hai que poñer unha variable de usuario que almacenará o resultado que devolverá o parámetro, para poder manexalo posteriormente. O seu valor inicial é null.

Un parámetro de entrada-saída (INOUT) indica que nesa posición hai que poñer unha variable de usuario que ten un valor inicial que se pasa como entrada cando se chama ao procedemento, e no que almacenará o resultado que devolverá o parámetro despois de executarse o procedemento.

No caso de chamar a un procedemento dende outro procedemento ou función, tamén se poden utilizar variables locais para os parámetros OUT e INOUT, en lugar de variables de usuario.

O procedemento almacenado pode devolver resultados mediante os parámetros OUT e INOUT.

Se non se indica IN, OUT ou INOUT, considérase que é de entrada (IN).

- O nome do parámetro pódese escribir en minúsculas ou maiúsculas indistintamente.
- O tipo de dato asociado do parámetro pode ser calquera tipo de dato válido en MySQL.

- *parámetro_función* ten a forma:

nome_parámetro tipo_dato

O tipo de dato pode ser calquera tipo de dato válido en MySQL.

- A cláusula RETURNS só se pode utilizar na creación de funcións, onde é obrigatoria. Indica o tipo de dato que retorna a función. No corpo da función ten que existir unha sentenza RETURN para indicar o valor que retorna a función. Sintaxe da sentenza RETURN:

RETURN expresión

O tipo de dato da expresión ten que ser o mesmo có que se especifica na cláusula

RETURNS.

Unha función só pode retornar un valor. No caso de necesitar unha rutina que devolva máis dun valor, utilizarase un procedemento almacenado con máis dun parámetro de saída (OUT).

- *característica* ten a forma:

```
LANGUAGE SQL | [NOT] DETERMINISTIC | SQL SECURITY { DEFINER | INVOKER } |  
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA } | COMMENT 'string'
```

- LANGUAGE SQL indica a linguaxe na que están escritas as sentenzas do corpo da rutina. Actualmente non se ten en conta porque só se permite o uso de sentenzas SQL, aínda que está previsto poder utilizar, nun futuro, sentenzas doutras linguaxes, como por exemplo PHP.
 - Unha rutina considérase de tipo DETERMINISTIC se sempre devolve o mesmo valor de saída para os mesmos valores dos parámetros de entrada. Unha rutina que contén unha función NOW() ou unha función RAND() sería de tipo NOT DETERMINISTIC. Se non se especifica nada, o valor que toma o servidor por defecto é NOT DETERMINISTIC.
 - SQL_SECURITY permite indicar se na execución da rutina almacenada se utilizan privilexios do creador da rutina (DEFINER - valor por defecto) ou do usuario que a chama (INVOKER).
 - Hai unha serie de características que fan referencia á natureza dos datos manexados pola rutina: CONTAINS SQL indica que a rutina non contén sentenzas que len ou escriben datos. NO SQL indica que a rutina non contén sentenzas SQL. READS SQL indica que contén sentenzas que len datos, por exemplo select, pero non sentenzas que escriben datos. MODIFIES SQL DATA indica que a rutina contén sentenzas que poden escribir datos, por exemplo insert ou update. O valor por defecto é CONTAINS SQL.
 - COMMENT permite introducir un comentario que se verá ao executar a sentenza SHOW CREATE PROCEDURE ou SHOW CREATE FUNCTION.
- *corpo_rutina* pode ser unha instrución ou un conxunto de instrucións SQL en forma de bloque de programación empezando por BEGIN e rematando en END. O corpo da rutina pode incluír sentenzas de declaración de variables, de asignación de valores, de control de fluxo, de manipulación de datos e a maioría das sentenzas da linguaxe SQL, e ademais, pode facer chamadas a outras rutinas almacenadas (procedementos e funcións definidas polo usuario).

Exemplos

- Exemplo de creación dun procedemento almacenado

```
use practicas1;  
drop procedure if exists contarEmpregados ;  
delimiter //  
create procedure contarEmpregados (pSexo char(1), out pContador smallint)  
begin  
    select count(*) into pContador  
        from practicas1.empleado  
        where sexo = convert(pSexo using utf8) collate utf8_spanish_ci;  
end  
//  
delimiter ;
```

O procedemento creado conta os empregados que hai na táboa *practicas1.empleado* que teñan na columna *sexo* o valor que se pasa como primeiro parámetro, e devolve o resultado nun parámetro de saída que aparece na segunda posición da lista de parámetros. A función *convert* converte o parametro ao sistema de colación adecuado, xa que a colación por defecto para utf8 é utf8_general_ci e o sistema de colación que utilizan as columnas da táboa é utf8_spanish_ci. Aos nomes dos parámetros se lles puxo ao inicio

unha letra p para diferencialos dos nomes de columnas, aínda que isto so é unha recomendación.

Utilízanse as sentenzas BEGIN e END para identificar as sentenzas que forman o corpo do procedemento, aínda que non sería obrigatorio porque o corpo do procedemento só contén unha sentenza.

A sentenza DELIMITER cambia o delimitador de fin de sentenza mentres se crea o procedemento, e despois da creación vólvese a deixar o valor normal que é o carácter punto e coma (;).

- Exemplo de creación dunha función

```
use practicas1;
create function saudo(pEntrada char(20)) returns char(27)
deterministic
return concat('Hola, ',pEntrada,'!');
```

A función recibe como parámetro de entrada unha cadea de caracteres de lonxitude 20, e retorna unha cadea de caracteres de lonxitude 27 formada pola concatenación do texto literal 'Hola, ' e a cadea recibida como parámetro de entrada. Hai que poñerlle a característica DETERMINISTIC, como a case todas as funcións, pois o valor por defecto é NOT DETERMINISTIC.

Como o corpo da función ten só unha liña, non é obrigatorio utilizar as sentenzas END e BEGIN, e tampouco fai falta cambiar o delimitador de final de sentenza porque o remate da creación da función coincide co remate da sentenza que forma o corpo da función.

A función pódese utilizar como parte dunha expresión nunha sentenza SQL igual que as funcións que xa ten definidas MySQL.

1.2.1.2 Utilización de rutinas almacenadas

Para chamar a un procedemento almacenado hai que utilizar a sentenza CALL, mentres que unha función utilízase como calquera outra función das que xa ten definidas MySQL, engadíndoa a unha expresión que pode utilizarse dentro dunha sentenza SQL, como por exemplo, SELECT, INSERT, UPDATE ou DELETE. A función devolve sempre un valor no momento de avaliar a expresión.

Sintaxe da sentenza CALL:

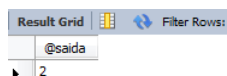
```
CALL nome_procedemento([parámetro [, ...]])
```

Exemplos

- Sentenzas para a execución do procedemento almacenado *contarEmpregados*:

```
call practicas1.contarEmpregados('m', @resultado);
select @resultado as numero_empleados;
```

Resultado:

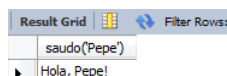


@saida
2

- Exemplo de uso da función *saudo*.

```
select practicas1.saudo('Pepe');
```

Resultado:



saudo(Pepe)
Hola, Pepe!



Tarefa 1: Crear e executar procedementos almacenados.



Tarefa 2: Crear e utilizar funcións definidas polo usuario.

1.2.1.3 Tratamento de erros. Sentenza SIGNAL

A sentenza SIGNAL devolve un erro como resultado da execución dun programa almacenado. Sintaxe:

```
SIGNAL valor_condición_erro
[SET nome_información = valor_información ...]
```

- *valor_condición_erro* indica o valor de erro que se vai a devolver. Pode ser un valor SQLSTATE (unha cadea de 5 caracteres), ou un nome de condición, que ten que estar declarada previamente. Os dous primeiros caracteres de SQLSTATE indican a clase de erro que pode ser un erro grave que produce a finalización da sentenza, unha advertencia (*warning*) ou un erro de tipo '*not found*'. O valor SQLSTATE para unha sentenza SIGNAL non debe empezar nunca por '00' porque eses valores corresponden a operacións rematadas con éxito. Para sinalar un valor SQLSTATE xenérico pódese utilizar o valor '45000' que significa 'excepción non controlada definida polo usuario'.
- A sentenza pode incluír, de maneira opcional, unha cláusula SET cunha lista de informacións asociadas á sinal de erro. Esta lista ten a forma:

```
nome_información = valor_información
```

Algúns dos *nomes_información* máis empregados son:

- MESSAGE_TEXT = 'texto da mensaxe que se devolve'.
- MYSQL_ERRNO = número de erro MySQL que se devolve.

Exemplo de utilización:

```
drop procedure if exists demoSignal;
delimiter //
create procedure demoSignal(pValor tinyint)
begin
    declare especialidad condition for sqlstate '45000';
    if pValor = 0 then
        signal sqlstate '01000';
    elseif pValor = 1 then
        signal sqlstate '45000' set message_text = 'Ocorreu un erro (1)';
    elseif pValor = 2 then
        signal especialidad set message_text = 'Ocorreu un erro (2)';
    else
        signal sqlstate '01000'
        set message_text = 'Ocorreu unha advertencia', mysql_errno = 1000;
        signal sqlstate '45000'
        set message_text = 'Ocorreu unha advertencia', mysql_errno = 1001;
    end if;
end;
//
delimiter ;
```

En función do valor que se pase como parámetro, o procedemento devolve unha condición de erro:

- No caso de que *pValor* tome o valor 0, devólvese unha advertencia (*warning*), porque o valor SQLSTATE empeza por '01'.

- No caso de que *pValor* tome o valor 1, devólvese un erro e unha mensaxe . O erro provoca que termine a execución do procedemento e móstrase a mensaxe coa información do erro.

381 12:14:27 call demoSignal(1) Error Code: 1644. Ocorreu un erro (1) 0.000 sec

- No caso de que *pValor* tome o valor 2, devólvese o mesmo erro pero asociado ao nome de condición.

382 12:14:27 call demoSignal(2) Error Code: 1644. Ocorreu un erro (2) 0.000 sec

- No caso de que *pValor* tome calquera outro valor, primeiro devólvese unha advertencia que non supón o final da execución do procedemento e execútase a seguinte sentenza que devolve: un erro, a mensaxe coa información do erro, o código do erro, e ademais, provoca que termine a execución do procedemento.

383 12:14:27 call demoSignal(3) Error Code: 1001. Ocorreu unha advertencia 0.010 sec

1.2.1.4 Documentación de programas almacenados

É moi recomendable documentar o código co obxecto de facilitar as labores de mantemento dos programas almacenados (facer modificacións nos programas almacenados debido á aparición de novas necesidades, novas normas, ...).

A documentación do código pódese incluír nos propios guións, inserindo liñas de comentarios que conteñan informacións referentes ao autor, a data de creación, tarefas que automatizan, parámetros que necesita, e resultados que producen.

Cando se fan cambios no código tamén se deberían inserir liñas de comentarios que conteñan a data da modificación, autor, cambios realizados, e causas polas que se fan os cambios.

Cando se utiliza MySQL Workbench para editar os guións de creación de programas almacenados pódese crear un fragmento de código (*snippet*) e gardalo para utilizalo cada vez que se crea un novo ficheiro coa creación ou modificación de programas almacenados. Para máis información sobre snippets, pódese consultar a Guía básica de MySQL Workbench 6.3 anexionada a esta actividade. Exemplo de contido do snippet *docu_creaRutina*:

```
/*
nome_arquivo.sql

NOME RUTINA:
DATA CREACIÓN:
AUTOR:
TAREFA A AUTOMATIZAR:
PARAMETROS REQUERIDOS:
RESULTADOS PRODUCIDOS:

*/
```

1.2.1.5 Modificación de procedementos almacenados e funcións

Os procedementos e función existentes pódense modificar empregando as sentenzas ALTER e DROP.

Sentenzas ALTER PROCEDURE e ALTER FUNCTION

Esta sentenza permite cambiar as características dunha rutina almacenada (procedemento ou función). Na mesma sentenza pódese cambiar máis dunha característica. Sen embargo, non se permite facer cambios nos parámetros nin no corpo da rutina, para facer estes cambios é necesario borrar a rutina almacenada e volvela a crear. Sintaxe :

```
ALTER {PROCEDURE | FUNCTION} nome_rutina [características ...]
```

As características que se poden modificar son algunhas das que se viron na sintaxe da orden de creación de rutinas almacenadas:

```
LANGUAGE SQL | SQL SECURITY { DEFINER | INVOKER } |  
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA } | COMMENT 'string'
```

- LANGUAGE SQL indica a linguaxe na que están escritas as sentenzas do corpo da rutina. Actualmente non se ten en conta porque só se permite o uso de sentenzas SQL, aínda que está previsto poder utilizar, nun futuro, sentenzas doutras linguaxes, como por exemplo PHP.
- SQL_SECURITY permite indicar se na execución da rutina almacenada se utilizan privilexios do creador da rutina (DEFINER - valor por defecto) ou do usuario que a chama (INVOKER).
- Características que fan referencia á natureza dos datos manexados pola rutina almacenada: CONTAINS SQL indica que a rutina non contén sentenzas que len ou escriben datos. NO SQL indica que a rutina non contén sentenzas SQL. READS SQL indica que a rutina contén sentenzas que len datos, por exemplo *select*, pero non sentenzas que escriben datos. MODIFIES SQL DATA indica que a rutina contén sentenzas que poden escribir datos, por exemplo *insert* ou *update*. O valor por defecto é CONTAINS SQL.
- COMMENT permite introducir un comentario que se verá ao executar a sentenza SHOW CREATE PROCEDURE ou SHOW CREATE FUNCTION.

Sentenzas DROP PROCEDURE e DROP FUNCTION

Estas sentenzas permiten borrar procedementos almacenados. Sintaxe:

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] nome_rutina
```

A opción IF EXISTS comproba primeiro se existe a rutina almacenada, e só fai o borrado no caso de que exista, evitando así que se produza un erro grave que faga que se corte a execución dun guión de sentenzas e dá lugar só a unha mensaxe de advertencia (*warning*).



Tarefa 3. Modificar procedementos almacenados e funcións.