Bases de datos

Unidade 7: Rutinas

1. A02. Rutinas almacenadas

1.1 Introdución

1.1.1 Obxectivos

Os obxectivos desta actividade son:

- Crear e manter rutinas almacenadas (procedementos almacenados e funcións de usuario). Utilizar parámetros.
- Utilizar funcións MySQL e rutinas almacenadas.
- Documentar e emendar erros nas rutinas almacenadas.

1.1.2 Software

Utilizarase a plataforma WAMP (Windows-Apache-MySQL-PHP) WampServer 2.5 (última versión estable en outubro 2015), que inclúe MySQL Community Edition 5.6.17 como SXBDR (Sistema Xestor de Bases de Datos Relacional). As razóns de utilización deste software son que:

- É software libre, polo que o alumnado poderá descargalo de forma gratuíta e utilizalo legalmente na súa casa.
- É unha forma sinxela de facer a instalación do software necesario para desenvolver aplicacións web.

 Páxina oficial de  WampServer: http://www.wampserver.com

 Páxina oficial de  MySQL: https://www.mysql.com/

Utilizarase MySQL Workbench 6.3 como ferramenta cliente gráfica xa que é a recomendada por MySQL en outubro de 2015, aínda que tamén poderían utilizarse outras como phpMyAdmin, EMS MyManager, ou MySQL Query Browser.

O resultado das probas dos guións SQL desta actividade, ilustrarase normalmente cunha imaxe capturada da zona de manipulación de datos de Workbench. Para completar esa información, por exemplo, cando non é posible mostrar o resultado completo, mostra-ranse outras zonas de Workbench, como por exemplo a zona de saída (*output*).

 En https://www.mysql.com/products/workbench/ pode obterse información detallada sobre a ferramenta MySQL Workbench e descargar o software.

 En http://dev.mysql.com/doc/index-gui.html pode descargarse o manual de MySQL Workbench.

 O material anexo a esta actividade inclúe unha Guía básica de MySQL Workbench

1.1.3 Bases de datos de traballo

As bases de datos *practicas1*, *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 scritps 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.calculoNnota(),* 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 procedemento 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 dicionario de datos, igual có resto de ob-

xectos 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, tómase 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.empregado
```

```
        where sexo = convert(pSexo using utf8) collate utf8_spanish_ci;
 end
//
delimiter ;
```

O procedemento creado conta os empregados que hai na táboa *practicas1.empregado* 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 columanas 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 carácteres de lonxitude 20, e retorna unha cadea de carácteres 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 asa 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:

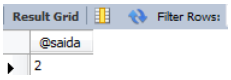
```
CALL   nome_procedemento([parámetro [, ...]])
```

Exemplos

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

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

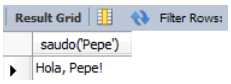
Resultado:



- Exemplo de uso da función *saudo*.

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

Resultado:



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 carácteres), ou un nome de condición, que ten que estar declarada previamente. Os dous primeiros carácteres 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.

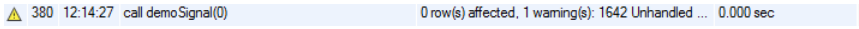
Exemplo de utilización:

```
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 ;
```

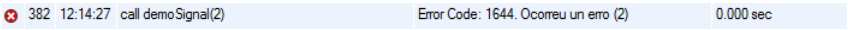
En función do valor que se pase como parámetro, o procedemento devolve unha condición de erro:

- 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.

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



- 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.

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.

1.3 Tarefas

As tarefas propostas son as seguintes:

- Tarefa 1. Crear e executar procedementos almacenados.
- Tarefa 2. Crear e utilizar funcións definidas polo usuario.
- Tarefa 3. Modificar procedementos almacenados e funcións.

1.3.1 Tarefa 1. Crear e executar procedementos almacenados

A tarefa consiste en escribir os guións de sentenzas SQL necesarios para crear procedementos almacenados atendendo a varios supostos, documentando os guións, e executando os procedementos creados.

- Tarefa 1.1. Crear un procedemento almacenado na base de datos *traballadores* que actualice a columna *depEmpregados* da táboa *departamento*, para todos os departamentos, contando o número de empregados que traballan nese departamento tendo en conta a información da columna *empDepartamento* da táboa *empregado*.
- Tarefa 1.2. Crear un procedemento almacenado co nome *vertaboas* na base de datos *utilidades*, que utilice a información contida na base de datos *information_schema* para mostrar información das táboas que hai nas bases de datos. O procedemento recibe como parámetro de entrada unha cadea e texto que pode ser o nome dunha base de datos, ou ben, o carácter asterisco ('*').
  - Cando se lle pasa o carácter '*' debe mostrar todas as táboas do servidor. Para cada táboa nos interesan as columnas: *table_schema, table_name, table_type, engine, table_rows* da táboa *information_schema .tables*, ordenando o resultado polo nome do esquema e o nome da táboa.
  - Cando se lle pasa o nome dunha base de datos, hai que comprobar que a base de datos existe na táboa *schemata*. No caso de existir, se mostrarán todas as táboas desa base de datos. Para cada táboa nos interesan as columnas: *table_name, table_type, engine, table_rows* da táboa *information_schema .tables*, ordenando o resultado polo nome da táboa. No caso de non existir a base de datos, se mostrará unha mensaxe de erro: ' A base de datos *xxxxx* non existe no servidor '.
- Tarefa 1.3. Crear un procedemento almacenado que nos permita inserir datos de proba na táboa *vendas* na base de datos *tendaBD*.
  - O número de filas a inserir se lle pasa como un parámetro.
  - En cada fila, os datos para as columnas *ven_cliente*, *ven_tenda* e *ven_empregado* obtéñense buscando unha fila de maneira aleatoria nas táboas *clientes*, *tendas* e *empregados* respectivamente e collendo o código que corresponde.
  - A columna *ven_data* colle a data do sistema.
  - Nas columnas *ven_id* e *ven_factura* non se cargan datos. Na primeira porque é de tipo autoincremental e xa a calcula o servidor, e a segunda porque non se cubre ata que se facture a venda.
- Tarefa 1.4. Crear un procedemento almacenado na base de datos *tendaBD* que permita controlar os intentos de acceso errados dos usuarios da base de datos. Os parámetros de entrada son o *login* e *password* do usuario.

A táboa *usuario* garda información dos usuarios que poden acceder á base de datos, e terá o seguinte esquema:

Nome columna	Tipo	Null	Clave	Observacións
login	varchar(16)	N	P	Nome de usuario
password	char(40)	N		Contrasinal do usuario

A táboa de *log_erro_conexion* rexistra os intentos de acceso errados, e terá o seguinte esquema:

Nome columna	Tipo	Null	Clave	Observacións
id	integer	N	P	Código autoincremental
login	varchar(16)	N		Nome de usuario
password	char(40)	N		Contrasinal do usuario
data_hora	timestamp	N		Data e hora do intento de acceso

O procedemento debe comprobar se existe na táboa *usuario* algún usuario co *login* e *password* que se pasan como parámetro. No caso de non existir, gárdase na táboa de rexistro *log_erro_conexion* a información correspondente ao intento de acceso errado. No caso de que o usuario faga máis de 5 intentos errados nos últimos 3 minutos, bloquearase a súa conta cambiándolle o contrasinal, poñendo unha contrasinal fixa establecida polo administrador, como por exemplo: 'H347B52(((]ERR'.

O procedemento utilizará un parámetro de saída para poder comprobar se o intento de acceso tivo éxito ou non. O parámetro ten o valor 0 se o *login* e o *password* corresponden a un usuario que existe na táboa; o valor 1 se o usuario non existe; o valor 2 no caso de bloqueo da conta por superar o número de intentos permitidos.

Solución

- Tarefa 1.1.
  - Código do procedemento

```
/*
u7a2tarefa0101.sql

-----------------------------------------------------------------------------------
NOME RUTINA: traballadores.sp_actualizar_depEmpregados (procedemento)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:     - Contar número de empregados que hai en cada departamento
                          - Actualizar a columna depEmpregados da táboa departamento
                         co número de empregados que traballan no departamento.
PARAMETROS REQUERIDOS:    - Non precisa parámetros
RESULTADOS PRODUCIDOS:    - Columna depEmpregados actualizada

-----------------------------------------------------------------------------------
*/
use traballadores;
delimiter //
create procedure sp_actualizar_depEmpregados()
begin
   update departamento
     set depEmpregados = (select count(*)
                          from empregado
                          where empDepartamento = depNumero);
end
//
delimiter ;
```

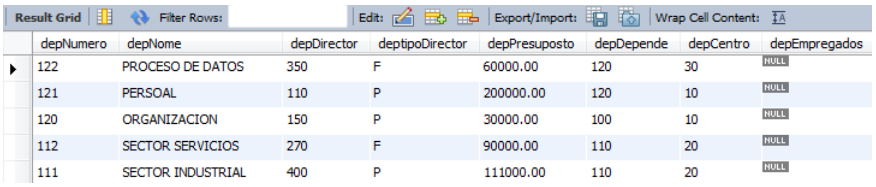
- Execución e comprobación do funcionamento do procedemento

```
call sp_actualizar_depEmpregados();
```

Non produce ningunha saída en pantalla; unicamente informa na zona de saída de MySQL Workbench do número de filas modificadas. Para comprobar o correcto funcionamento do procedemento almacenado hai que consultar o contido da columna *depEmpregados* da táboa *departamento* e contrastar os valores dalgún departamento cos datos da táboa *empregado*.

- Consulta antes de executar o procedemento:

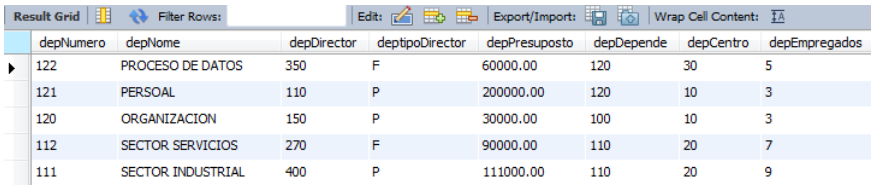
```
select * from departamento;
```



depNumero	depNome	depDirector	deptipoDirector	depPresuposto	depDepende	depCentro	depEmpregados
122	PROCESO DE DATOS	350	F	60000.00	120	30	NULL
121	PERSOAL	110	P	200000.00	120	10	NULL
120	ORGANIZACION	150	P	30000.00	100	10	NULL
112	SECTOR SERVICIOS	270	F	90000.00	110	20	NULL
111	SECTOR INDUSTRIAL	400	P	111000.00	110	20	NULL

- Consulta despois de executar o procedemento:

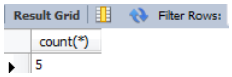
```
select * from departamento;
```



depNumero	depNome	depDirector	deptipoDirector	depPresuposto	depDepende	depCentro	depEmpregados
122	PROCESO DE DATOS	350	F	60000.00	120	30	5
121	PERSOAL	110	P	200000.00	120	10	3
120	ORGANIZACION	150	P	30000.00	100	10	3
112	SECTOR SERVICIOS	270	F	90000.00	110	20	7
111	SECTOR INDUSTRIAL	400	P	111000.00	110	20	9

- Consulta cantos empregados hai no departamento 122, na táboa empregado:

```
select count(*) from empregado where empDepartamento = 122;
```



count(*)
5

- Tarefa 1.2.
  - Código do procedemento

```
/*
u7a2tarefa0102.sql

-------------------------------------------------------------------------------
NOME RUTINA: utilidades.vertaboas (procedemento)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:     - Mostrar información resumida das táboas que hai nun
                            servidor, ou dunha base de datos concreta, tendo en conta a
                            información almacenada nas columnas table_schema, table_name
                            table_name, table_type, engine, table_rows da táboa tables
                            da base de datos information_schema. Os datos deben saír
                            ordenados polo nome da base de datos, e o nome da táboa.
PARAMETROS REQUERIDOS:    - IN: pBaseDatos se lle poden pasar como valores válidos
                            o carácter * que significa que se quere ver información das
                            táboas de todas as bases de datos, ou o nome dunha base de
                            datos no caso de querer ver información das táboas dunha
                            base de datos concreta. Calquera outro valor produce unha
```

```
                        mensaxe de erro. O tipo de dato do parámetro ten que ser
                        o mesmo que a columna table_schema para poder comparalas.
RESULTADOS PRODUCIDOS:  - Mostrar en pantalla a información solicitada
-------------------------------------------------------------------------------
*/
delimiter //
-- creación do procedemento usando un nome cualificado
create procedure utilidades.vertaboas(pBaseDatos varchar(64) character set utf8)
begin
  declare existe bit default 0;
  if pBaseDatos='*' then
    select concat(upper(table_schema),'.',lower(table_name)) as `táboa`,
           lower(table_type) as tipo,
           lower(engine) as motor,
           table_rows as filas,
           create_time as data_creacion
    from information_schema.tables
    order by `táboa`;
  else
    select count(*) into existe
      from information_schema.SCHEMATA
      where SCHEMA_NAME=pBaseDatos;
    if existe = 1 then
        select table_name as `táboa`,
           lower(table_type) as tipo,
           lower(engine) as motor,
           table_rows as filas,
           create_time as data_creacion
        from information_schema.tables
        where table_schema=pBaseDatos
       order by `táboa`;
    else
        select concat('A base de datos  "',pBaseDatos,'"  non existe') as Error;
   end if;
  end if;
end
//
delimiter ;
```

- Execución e comprobación do funcionamento do procedemento. Fanse dúas probas pasando dous valores válidos para o parámetro, e unha terceira proba cun valor que non é válido, para comprobar que se mostra a mensaxe de erro.

```
call utilidades.vertaboas('*');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

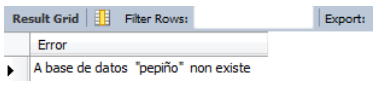
táboa	tipo	motor	filas	data_creacion
CINE.director	base table	innodb	0	2015-10-13 16:46:19
CINE.pelicula	base table	innodb	0	2015-10-13 20:22:39
ELECCIONMODULOS.grupo	base table	innodb	0	2015-11-17 12:57:53
ELECCIONMODULOS.imparte	base table	innodb	0	2015-11-17 12:57:54
ELECCIONMODULOS.modulo	base table	innodb	0	2015-11-17 12:57:53

```
call utilidades.vertaboas('traballadores');
```

Result Grid | Filter Rows: | Export: | Wrap Cell

táboa	tipo	motor	filas	data_creacion
centro	base table	myisam	3	2015-11-23 12:32:32
departamento	base table	myisam	9	2015-11-23 12:32:33
empregado	base table	myisam	39	2015-11-23 12:32:33

```
call utilidades.vertaboas('pepiño');
```



- Tarefa 1.3.
  - Código do procedemento

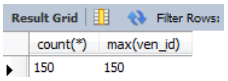
```
/*
u7a2tarefa0103.sql

-------------------------------------------------------------------------------
NOME RUTINA: sp_inserir_vendas_proba (procedemento)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:     - Inserir datos de proba na táboa vendas da base de datos
                           tendaBD. Os datos para as columnas ven_cliente, ven_tenda e
                          ven_empregado obtéñense buscando unha fila de maneira
                          aleatoria nas táboas clientes, tendas e empregados, e collendo
                          o código que corresponde. A columna ven_data colle a data do
                          sistema. Para as columnas ven_id e ven_factura non se cargan
                          datos.
PARAMETROS REQUERIDOS:    - IN: pFilas - Indica o número de filas a inserir.
RESULTADOS PRODUCIDOS:    - Non produce saída en pantalla. Insire na táboa vendas
                           filas con datos de proba. O nº de filas que se insiren
                           pásase como un parámetro de entrada.
-------------------------------------------------------------------------------
*/
use tendaBD;
drop procedure if exists sp_inserir_vendas_proba;
delimiter //
create procedure sp_inserir_vendas_proba(pFilas integer)
begin
   declare vCliente, vEmpregado  smallint unsigned;
   declare vTenda tinyint unsigned;
   declare vCcontador tinyint unsigned default 0;
   while vCcontador < pFilas do
          /*seleccionar un empregado aleatoriamente*/
          select emp_id into vEmpregado
                 from empregados
                 order by rand()
                 limit 1;
          /*seleccionar un cliente aleatoriamente*/
          select clt_id into vCliente
                 from clientes
                 order by rand()
                 limit 1;
          /*seleccionar una tenda aleatoriamente*/
          select tda_id into vTenda
                 from tendas
                 order by rand()
                 limit 1;
          /*inserir unha fila na táboa de vendas*/
          insert into vendas (ven_tenda,ven_empregado,ven_cliente,ven_data)
                 values (vTenda, vEmpregado, vCliente, now());
          /*contar a fila nserida*/
          set vCcontador = vCcontador + 1;
   end while;
end

//
delimiter ;
```

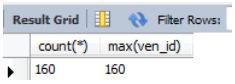
- Execución e comprobación do funcionamento do procedemento. Pódese executar o procedemento pasándolle como parámetro o número de filas que se van a inserir e despois execútase unha consulta con SELECT para ver os datos inseridos. Se non se desexan conservar estas filas engadidas e son as únicas feitas na data actual, pódense borrar cunha sentenza DELETE.
  - Consulta do número de filas antes de executar o procedemento almacenado, e id da última venda.

```
select count(*), max(ven_id) from tendaBD.vendas;
```



  - Execución do procedemento almacenado e consulta para saber se foron inseridas as filas.

```
call tendaBD.sp_inserir_vendas_proba(10);
select count(*), max(ven_id) from tendaBD.vendas;
```



  - Borrado das filas inseridas na proba.

```
delete from vendas where ven_id between 151 and 160;
```

- Tarefa 1.4.
  - Código do procedemento

```
/*
u7a2tarefa0104.sql

-------------------------------------------------------------------------------
NOME RUTINA: sp_erro_login (procedemento)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:     - Controlar intentos de acceso dos usuarios, comprobando se
                          o usuario que intenta acceder está na táboa de usuarios.
                          No caso de non existir o usuario na táboa se rexistra a
                          información do intento de acceso nunha táboa de rexistro.
                          No caso de que un usuario faga máis de 5 intentos errados nos
                          últimos 3 minutos se lle bloqueará a súa conta cambiándolle o
                          password por un valor fixo establecido polo administrador

PARAMETROS REQUERIDOS:    - IN: pLogin: login do usuario.
                          - IN: pPassword: contrasinal do usuario
                          - OUT: pMensaxe: devolve o valor 0 se o login e  password
                          corresponden a un usuario que existe na táboa de usuarios, o
                          valor 1 se o usuario non existe, e o valor 2 no caso en que
                          se lle cambie á password ao usuario por superar o número de
                          intentos permitidos.
RESULTADOS PRODUCIDOS:    - Non mostra nada en pantalla, pero devolve os valores 0,1,2
                          no parámetro de saída.

-------------------------------------------------------------------------------
*/
use tendaBD;
drop procedure if exists sp_erro_login;
delimiter //
create procedure sp_erro_login (pUsuario char(16), pPalabra char(40), out pMensaxe
```

```
tinyint(1))
   begin
     declare vIntentos int;    /*contador de intentos errados nos tres últimos minutos*/
     declare vUsuarioValido boolean default 0; /*vale 1 cando usuario existe na táboa*/
     /*Comprobación da existencia do usuario co login e password pasados como parámetro */
     select count(*) into vUsuarioValido
      from usuario
      where login=pUsuario and password=pPalabra;
     /*No caso de que non sexa correcta a conta de usuario rexístrase o intento errado en
log_erro_conexion e cóntanse o número de intentos errados nos últimos 3 minutos*/
     if vUsuarioValido = 0  then   /*No caso de non existir o usuario*/
           insert into log_erro_conexion (login, password) values (pUsuario, pPalabra);
           select count(*) into vIntentos   /*Contar intentos nos últimos 3 minutos*/
                    from log_erro_conexion
                    where login = pUsuario and timestampdiff(minute,data_hora,now()) <=3;
           if vIntentos <= 5 then
                  set pMensaxe = 1;
           else
                  set pMensaxe = 2;
                  update usuario set password = 'H347B52(((]ERR' where login = pUsuario;
           end if;
      else     /*No caso de existir o usuario*/
           set pMensaxe = 0;
     end if;
   end
   //
   delimiter ;
```

- Execución e comprobación do funcionamento do procedemento. O primeiro para facer a comprobación é crear as táboas no caso de non existir, e dar de alta algún usuario. Para facer as probas se van a inserir usuarios co seu *password* sen cifrar, aínda que na práctica real a *password* dos usuarios debería gardarse cifrada utilizando para elo funcións que xa incorpora MySQL, como MD5, SHA1, ou SHA2.

```
use tendaBD;
create table if not exists usuario (
login varchar(16),
password char(40),
primary key (login)
)engine = innodb;
create table if not exists log_erro_conexion (
id integer unsigned auto_increment not null,
login varchar(16),
password char(40),
data_hora timestamp default now(),
primary key (id)
)engine = innodb;
insert into usuario values ('pepe','pepe');
insert into usuario values ('pepa','pepa');
```

Execútase o procedemento e mírase cal é o valor que devolve o parámetro de saída en cada execución. Primeiro pásanse como parámetros un *login* e un *pasword* dun usuario que exista, e compróbase que devolva o valor 0, e despois execútase 6 veces o procedemento pasando sempre o mesmo *login*, pero cunha *password* errónea. Despois dos seis intentos compróbase que o *password* do usuario foi modificado.

```
call sp_erro_login('pepe','pepe',@proba);
select @proba;
call sp_erro_login('pepe','aa',@proba);
select @proba;
call sp_erro_login('pepe','ee',@proba);
```

```
select @proba;
call sp_erro_login('pepe','el',@proba);
select @met;
call sp_erro_login('pepe','es',@met);
select @proba;
select * from usuario;
call sp_erro_login('pepe','ex',@proba);
select @proba;
call sp_erro_login('pepe','EX',@proba);
select @proba;
select * from log_erro_conexion;
select * from usuario;
```

1.3.2 Tarefa 2. Crear e utilizar funcións definidas polo usuario

A tarefa consiste en escribir os guións de sentenzas SQL necesarios para crear funcións atendendo a varios supostos, e facer as probas de funcionamento utilizando as funcións creadas nunha consulta coa sentenza SELECT.

- Tarefa 2.1. Crear unha función na base de datos *utilidades* á que se lle pasa como parámetro o número do mes, e devolva o nome do mes en galego.
- Tarefa 2.2. Crear unha función na base de datos *utilidades* á que se lle pase como parámetro a nota numérica (dous enteiros e dous decimais) dun alumno, e devolva a nota en letra tendo en conta a seguinte táboa:

Nota numérica		Nota en letra
>= 0	< 5	suspenso
>= 5	< 6	aprobado
>= 6	< 7	ben
>= 7	< 9	notable
>= 9	<= 10	sobresaínte
Outro valor		erro na nota

- Tarefa 2.3. Crear unha función na base de datos *utilidades* que pasándolle como parámetro as 8 cifras correspondentes ao número do DNI, devolva a letra que lle corresponde.

  A letra do DNI obtense mediante un algoritmo coñecido como módulo 23. O algoritmo consiste en dividir o número correspondente ao DNI entre 23 e obter o resto da división enteira. O resultado é un número comprendido entre o 0 e o 22. A cada un destes números se lles fai corresponder unha letra tendo en conta a seguinte táboa:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T	**R**	**W**	**A**	**G**	**M**	**Y**	**F**	**P**	**D**	**X**	**B**	**N**	**J**	**Z**	**S**	**Q**	**V**	**H**	**L**	**C**	**K**	**E**

  Non se utilizan as letras: I, Ñ, O, e U. As letras I e O se descartan para evitar confusións con outros carácteres, como 1, l o 0.

- Tarefa 2.4. Crear unha función na base de datos *utilidades* que pasándolle como parámetro os 8 primeiros carácteres correspondentes ao Número de Identidade de Estranxeiro (NIE), devolva a letra que lle corresponde.

  Utilízase o mesmo algoritmo que para o DNI, coa diferenza de que o NIE pode empezar por unha letra, polo que hai que engadirlle as seguintes restricións:

  - No caso de que o NIE empece pola letra X, se calcula desprezando a X, e utilizando os 7 díxitos restantes.
  - No caso de que o NIE empece pola letra Y, se substitúe a letra Y polo número 1.

- No caso de que o NIE empece pola letra Z, se substitúe a letra Z polo número 2.

- Tarefa 2.5. Crear unha función na base de datos *utilidades* á que se lle pasa como parámetro os 20 díxitos dunha conta bancaria española, e retorne como saída o IBAN que lle corresponde.

TA1. Cálculo do IBAN. Contén a explicación de como se calcula o IBAN para as contas bancarias de España, noutros países as contas bancarias poden ter ata 34 díxitos, e poden incluír letras.

Solución

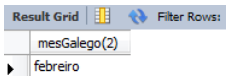
- Tarefa 2.1.
  - Código da función

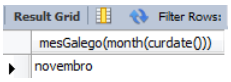
```
/*
u7a2tarefa0201.sql

_____________________________________________________________________________
NOME RUTINA: mesGalego (función)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:    - Obter o nome do mes en galego partindo do número do mes
PARAMETROS REQUERIDOS:   - Número do mes
RESULTADOS PRODUCIDOS:   - Nome do mes en galego
_____________________________________________________________________________
*/
use utilidades;
drop function if exists mesGalego ;
delimiter //
create function mesGalego(pMes tinyint(2)) returns char(10)
  deterministic
  begin
    declare vMesLetra char(10) default null;
    case pMes
        when 1 then set vMesLetra="xaneiro";
        when 2 then set vMesLetra="febreiro";
        when 3 then set vMesLetra="marzo";
        when 4 then set vMesLetra="abril";
        when 5 then set vMesLetra="maio";
        when 6 then set vMesLetra="xuño";
        when 7 then set vMesLetra="xullo";
        when 8 then set vMesLetra="agosto";
        when 9 then set vMesLetra="setembro";
        when 10 then set vMesLetra="outubro";
        when 11 then set vMesLetra="novembro";
        when 12 then set vMesLetra="decembro";
    end case;
    return vMesLetra;
  end
//
delimiter ;
```

  - Proba do funcionamento da función

```
select mesGalego(2); #febreiro
select mesGalego(month(curdate())); #mes da data actual
```





- Tarefa 2.2.
  - Código da función

```
/*
u7a2tarefa0202.sql

-------------------------------------------------------------------------------
NOME RUTINA: notaLetra (función)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:     - Obter a nota en forma de texto partindo dunha nota numérica
PARAMETROS REQUERIDOS:    - Cifra de dous enteiros e dous decimais correpondente á nota
RESULTADOS PRODUCIDOS:    - Cadea de 20 carácteres coa nota en forma de texto

-------------------------------------------------------------------------------
*/
use utilidades;
delimiter //
drop function if exists notaLetra //
create function notaLetra(pNota decimal(4,2))  returns char(20)
deterministic
begin
    declare vTexto char(20);
        if pNota >= 0 and pNota < 5 then set vTexto = 'suspenso';
         elseif pNota >= 5 and pNota < 6 then set vTexto = 'aprobado';
        elseif pNota >= 6 and pNota < 7 then set vTexto = 'ben';
        elseif pNota >= 7 and pNota < 9 then set vTexto = 'notable';
        elseif pNota >= 9 and pNota <= 10 then set vTexto = 'sobresaínte';
        else set vTexto = 'Erro na nota';
    end if;
    return vTexto;
end //
delimiter ;
```

  - Proba do funcionamento da función

```
select notaLetra(0);     #suspenso
select notaLetra(1);     #suspenso
select notaLetra(5);     #aprobado
select notaLetra(6.9);   #ben
select notaLetra(8.5);   #notable
select notaLetra(10);    #sobresaínte
select notaLetra(11);    #Erro na nota
```

- Tarefa 2.3.
  - Código da función

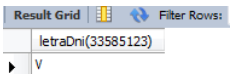
```
/*
u7a2tarefa0203.sql

-------------------------------------------------------------------------------
NOME RUTINA: letraDni (función)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:     - Obter a letra correspondente a un DNI a partir do algoritmo
                            coñecido como módulo 23
PARAMETROS REQUERIDOS:    - Número enteiro, correspondente ao número dun DNI
RESULTADOS PRODUCIDOS:    - Cadea de 1 carácter correspondente a letra do DNI

-------------------------------------------------------------------------------
*/
use utilidades;
create function letraDni (pDni integer) returns char(1)
 deterministic
 return substring('TRWAGMYFPDXBNJZSQVHLCKE', pDni % 23 + 1, 1);
```

  - Proba do funcionamento da función

```
select letraDni(33585123);
```

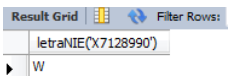


- Tarefa 2.4.
  - Código da función

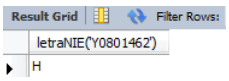
```
/*
u7a2tarefa0204.sql
_____________________________________________________________________________
NOME RUTINA: letraNIE (función)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:    - Obter a letra correspondente a un NIE a partir do algoritmo
                         coñecido como módulo 23, engadindo as restricións:
                         a) Se a primeira letra é unha X se despreza a primeira letra
                         b) Se a primeira letra é unha Y se substitúe polo número 1
                         c) Se a primeira letra é unha Z se substitúe polo número 2
                         d) Se empeza por calquera outro carácter devolve un cero
PARAMETROS REQUERIDOS:   - Cadea de 8 carácteres, correspondentes a un NIE
RESULTADOS PRODUCIDOS:   - Cadea de 1 carácter correspondente a letra do NIE ou un 0
_____________________________________________________________________________
*/
use utilidades;
drop function if exists letraNIE;
delimiter //
create function letraNIE(pNIE char(8)) returns char(1) deterministic
  begin
   declare vBase integer;
    case left(pNIE,1)
        when 'X' then set vBase = right(pNIE,7);
        when 'Y' then set vBase = concat('1',right(pNIE,7));
        when 'Z' then set vBase = concat('2',right(pNIE,7));
        else  return '0';
    end case;
        return substring('TRWAGMYFPDXBNJZSQVHLCKE', vBase % 23 + 1, 1);
  end
//
delimiter ;
```

  - Proba do funcionamento da función

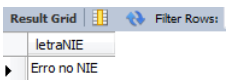
```
select letraNIE('X7128990');
```




```
select letraNIE('Y0801462');
```




```
select if(letraNIE('30801462')=0,'Erro no NIE',letraNIE('30801462')) as letraNIE;
```



- Tarefa 2.5.
  - Código da función

```
/*
u7a2tarefa0205.sql

_____________________________________________________________________________

NOME RUTINA: calcularIBAN (función)
DATA CREACIÓN: 9/11/2015
AUTOR: Grupo licenza 2015
TAREFA A AUTOMATIZAR:     - Obter o código internacional de conta bancaria (IBAN) para
                            contas en España. O algoritmo para o cálculo descríbese no
                            documento 'Cálculo de IBAN.pdf'
PARAMETROS REQUERIDOS:    - Cadea de 20 carácteres que identifican unha conta bancaria
                          - Cadea de dous carácteres co código do país: ES - España
RESULTADOS PRODUCIDOS:    - Cadea de 25 carácter correspondente ao IBAN da conta

_____________________________________________________________________________

*/
use utilidades;
drop function if exists calcularIBAN;
delimiter //
create function calcularIBAN(pCCC char(20), pPais char(2)) returns char(25) charset la-
tin1
    deterministic
begin
   declare vBase decimal(30,0);
   declare vControl tinyint(2) zerofill;
   set vBase = concat(pCCC,locate(left(pPais,1),'ABCDEFGHIJKLMNOPQRSTUVWXYZ')+9,
                      locate(right(pPais,1),'ABCDEFGHIJKLMNOPQRSTUVWXYZ')+9,'00');
   set vControl = 98 - vBase % 97;
   return concat(pPais,vControl,' ',pCCC);
  end
//
delimiter ;
```

  - Proba do funcionamento da función

```
select calcularIBAN('00120345030000067890','ES');
```

Result Grid	Filter Rows:
calcularIBAN('00120345030000067890','ES')	
ES07 00120345030000067890	

```
select calcularIBAN('900000012101234567 89','ES');
```

Result Grid	Filter Rows:
calcularIBAN('900000012101234567 89','ES')	
ES06 900000012101234567 89	

```
select calcularIBAN('21000418450200051332','ES');
```

Result Grid	Filter Rows:
calcularIBAN('21000418450200051332','ES')	
ES91 21000418450200051332	

1.3.3 Tarefa 3. Modificar procedementos almacenados e funcións

A tarefa consiste en facer modificacións nos seguintes procedementos almacenados e funcións:

- Tarefa 3.1. Cambiar as seguintes características do procedemento almacenado *utilidades.vertaboas()* creado na tarefa 1.2.:

- Se teñan en conta os privilexios do usuario que o executa (INVOKER).
- Engadir como comentario o texto: 'Mostra a información das táboas das bases de datos que se pasan como parámetro'.

- Tarefa 3.2. Borrar a función letraNIE, creada na tarefa 2.4.

Solución

Para facer os cambios pódese facer referencia á rutina cualificándoa co nome da base de datos, ou ben activar a base de datos. Nas solucións propostas móstranse as dúas opcións. A comprobación dos cambios pódese facer consultando o dicionario de datos. No caso de MySQL, as táboas *mysql.proc* ou *information_schema.routines*.

- Tarefa 3.1

```
alter procedure utilidades.vertaboas
 comment 'Mostra información das táboas das bases de datos que se pasan como parámetro'
 sql security invoker;
```

- Tarefa 3.2

```
use utilidades;
drop function if exists letraNIE;
```