

Bases de datos

Unidade 7: Programación de BD

1. A01. Programación de bases de datos

1.1 Introducción

1.1.1 Obxectivos

Os obxectivos desta actividade son:

- Coñecer as posibilidades de programación de bases de datos.
- Manexar as ferramentas e sentenzas SQL utilizadas para elaborar programas e rutinas almacenadas no lado do servidor de bases de datos.

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 utilízalo 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, mostráranse outras zonas de Workbench, como por exemplo a zona de saída (*output*) con información do estado da execución do guión e o número de filas que devolve.



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

A bases de datos *practicais* utilizarase para algúns exemplos e tarefas desta actividade. Antes de empezar a probar os exemplos ou realizar as tarefas, hai que executar o script de creación no servidor e poñer en uso a base de datos. O script atópase no cartafol anexo a esta actividade descrito no apartado '3.3 Material auxiliar'.

1.2 Actividade

1.2.1 Programación de bases de datos

Unha práctica habitual no manexo dos datos almacenados nas bases de datos é ter deseñadas aplicacións cunha interface fácil de manexar para ser utilizadas por usuarios que non teñan coñecementos suficientes de SQL. O deseño das aplicacións e da base de datos pódese facer en paralelo.

Por exemplo, un sitio Web que permite ver información das actividades culturais dunha cidade e que ten unha base de datos para almacenar a información relativa aos usuarios que poden acceder aos contidos restrinxidos da web, ás salas de exposición, teatros, e cines da cidade, así como a información do programa de actividades. Os usuarios manexan a información a través de formularios HTML, cun botón de envío que cando se pulsa fai unha petición ao servidor da base de datos, para executar unha sentencia INSERT, UPDATE, DELETE ou SELECT. As aplicacións que utilizan os usuarios estarían escritas en linguaxes de programación como PHP, AJAX, Visual C++, ou Java.

Case todos os SGBDR incorporan utilidades que permiten ampliar a linguaxe SQL para poder crear e almacenar no servidor pequenos programas que utilizan instrucións propias da programación procedimental (manexo de variables, instrucións condicionais, bucles, ...). Desta maneira, o administrador da base de datos pode crear pequenos programas para automatizar algunhas tarefas sobre as bases de datos, que poden ser utilizados polos usuarios, ou polos programadores de aplicacións que os chaman dende os seu programas. Exemplos:

- MySQL incorpora un conxunto de sentenzas coas funcións propias das linguaxes procedimentais (manexo de variables, instrucións condicionais, bucles, manexo de excepcións, ...).
- Oracle incorpora a linguaxe PL/SQL para a xestión dos datos, que é unha extensión procedimental da linguaxe SQL.
- MS SQL Server incorpora a linguaxe Transact SQL.
- PostgreSQL incorpora a linguaxe PostgreSQL.

En todo caso, estas extensión de SQL non están deseñadas para a creación de aplicacións sobre a base de datos; iso faise con outras ferramentas como Oracle Developer, PHP Generator for MySQL, ou linguaxes de programación externos como PHP, AJAX, Java, ou Visual C++.

1.2.1.1 Programas almacenados 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.

1.2.1.2 Vantaxes de utilizar programas almacenados no servidor de bases de datos

- **Seguridade:** Os procedementos ocultan os nomes das táboas a usuarios que non teñan privilexios para manipular os datos. Estes usuarios simplemente chaman aos procedementos sen coñecer a estrutura da base de datos. Os bancos, por exemplo, utilizan os procedementos almacenados para todas as operacións estándar, e non deixan o acceso ás táboas en mans dos programadores das aplicacións. Deste xeito asegúranse de que as operacións son feitas e rexistradas correctamente. Nunha situación deste tipo, as aplicacións non acceden directamente ás táboas, se non que o fan executando os procedementos almacenados, que son os encargados de manexar os datos na base de datos.
- **Estándares de código:** Un equipo de programadores pode compartir programas almacenados no servidor de bases de datos. No caso de que cada programador creara o seu propio código para realizar unha mesma tarefa, poderían existir problemas de integridade, ademais da perda de tempo.
- **Velocidade:** Os programas almacenados almacénanse no lado do servidor, de maneira que o código necesario para definilos envíase á rede só cando se crea o programa e non cando se executa, o que reduce a carga na rede, aínda que aumenta a carga no servidor de bases de datos.
- **Control de erros:** Proporcionan un mecanismo para control de erros xestionado polo administrador da base de datos. Permiten ao administrador levar un rexistro de erros e de acceso aos datos, distinto do que proporcionan os arquivos de rexistro (log) do servidor.

A principal desvantaxe da utilización de programas almacenados en SQL é que as linguaxes procedimentais de distintos fabricantes non son compatibles entre elas.

1.2.1.3 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

1.2.1.4 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 programación, 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*. Un bloque pode estar contido noutro bloque. Sintaxe da creación de bloques de programación :

```
[etiqueta_inicio:] BEGIN
    [lista_sentenzas]
END [etiqueta_fin]
```

- 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.
- As etiquetas de inicio e de final, que son optativas, pódense utilizar para facer referencia a ese bloque dende calquera parte do programa almacenado.

1.2.1.5 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 un carácter ou combinación de caracteres 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_senza
```

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

```

1.2.1.6 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 diferenciarlos das variables e dos nomes de columnas pode ser útil poñerlles nomes que empecen por p. Exemplos: *pDNI*, ou *pNome*.

1.2.1.7 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:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
hola mundo			
▶ hola mundo			

1.2.1.8 Ferramentas para a edición e execución de guións

Para a edición e execución de guións necesítase o código coas sentenzas SQL, un cliente conectado ao servidor que permita enviar o código e un servidor que execute ese código.

A escritura de guións de sentenzas SQL pode facerse con calquera editor de texto plano. As aplicacións cliente cunha interface gráfica (GUI) incorporan un editor especializado na escritura de guións e posibilita ademais de maneira sinxela e rápida a execución dos guións mediante un sistema de menús ou combinación de teclas.

As aplicacións cliente máis empregadas para conectase a un servidor MySQL e envialle as sentenzas SQL para que se executen, son as seguintes:

- MySQL Workbench

Cliente gráfico que de forma fácil e intuitiva permite establecer conexión cun servidor, mostrar información sobre a conexión, o servidor, bases de datos e táboas; escribir sentenzas SQL mediante un editor; enviar sentenzas ao servidor para que sexan executadas; dispón de axuda nas tarefas de edición e execución de sentenzas. Para máis información sobre o seu funcionamento, débese utilizar o manual de referencia ou ben a guía básica de funcionamento de MySQL Workbench incluído no material auxiliar da actividade.

- Consola modo texto `mysql.exe`

Cliente en modo texto que permite establecer a conexión co servidor e escribir e executar sentenzas SQL.

- No caso de ter instalado Wampserver, hai que abrir o menú de Wampserver, e seleccionar 'MySQL console' no submenú de MySQL. Ábrese unha consola en modo texto que solicita introducir o contrasinal do usuario *root*. Unha vez introducida o contrasinal, móstrase o prompt *mysql>* que indica que está establecida a conexión co servidor e se poden empezar a escribir sentenzas para que o servidor as execute.
- No caso de non ter instalado Wampserver, hai que abrir unha consola de ordes do sistema, e executar a utilidade cliente `mysql.exe` que está no directorio bin que se atopa no directorio de instalación de MySQL (en Windows é normalmente, `c:\Program Files\MySQL\MySQL version`).

- Aplicación web phpMyAdmin:

Cliente web que de forma gráfica permite establecer a conexión co servidor e enviarlle sentenzas SQL para que sexan executadas.

- No caso de ter instalado Wampserver, hai que abrir o menú e seleccionar a opción phpMyAdmin.
- No caso de non ter instalado Wampserver, hai que abrir un navegador e teclear a url coa identificación do servidor web no que está instalado phpMyAdmin. Exemplos de url: `http://localhost/phpMyAdmin` (no caso que phpMyAdmin estea instalado nun servidor web no equipo local), `http://proveedor_web.com/phpMyAdmin` (no caso que phpMyAmin estea instalado no servidor proveedor_web.com).

1.2.2 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.

A proba das sentenzas básicas de programación con MySQL nesta actividade, farase creando un procedemento almacenado sinxelo cun único bloque de programación, similar ao que se mostra no apartado "Creación de programas almacenados".

1.2.2.1 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.empleado' 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] código_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 anexo correspondente do manual de referencia de MySQL (Apéndice B, sección 3, no Manual de referencia de MySQL 5.6). No apartado sobre declaración de condicións, pódese ver un pequeno resumo.
 - 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 os códigos de estado que empezan por 02. Utilízanse no manexo de cursores que se ve máis adiante. Tamén se pode utilizar nunha asignación mediante SELECT ... INTO, para indicar que a consulta non devolveu ningunha fila.
 - 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:

Result Grid			Filter Rows: <input type="text"/>	Export: <input type="button" value=""/>
	numero_intento	vFinal		
	Primer intento	0		

Result Grid		Filter Rows:	Export
numero_intento	vFinal		
Segundo intento	1		



Tarefa1. Declarar variables, condicións, e manipuladores.

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 diferenciarlas 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 asígnaselles 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:

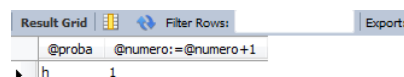
```
SELECT nome_columna [,...] INTO nome_variable [,... ]
```

Utilízanse cursores para manexar o resultado dunha consulta que devolve máis dunha fila. O manexo de cursores verase máis adiante.

Exemplo de asignación de valores coa sentenza SELECT:

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

Resultado da execución:



@proba	@numero:=@numero+1
h	1



Tarefa2. Asignar valores a variables.

1.2.2.2 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_senzas1
  [ELSEIF condición2 THEN lista_senzas3] ...
  [ELSE lista_senzas2]
END IF
```

O resultado é:

- Se a *condición1* é verdadeira, execútanse as sentenzas contidas en *lista_senzas1*.
- Se a *condición1* é falsa,
 - e existe a parte *else*, execútanse as sentenzas contidas en *lista_senzas2*.
 - e existe a parte *elseif*, avalíase a *condición2*: se é verdadeira, execútanse a sentenza ou sentenzas contidas en *lista_senzas3*.
 - 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 muestre 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:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Número par que non é múltiplo de 10			



Tarefa 3. Utilizar a sentença *if*.

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 sentença fóra da creación de programas almacenados. Exemplo de uso dentro dunha consulta:

```

-- Exemplo de uso nunha consulta con sentença 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:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
muller			
► muller			



Tarefa 4. Utilizar a sentenza *case*.

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:

Result Grid	Filter Rows:
vNumero vSuma	
► 101 5050	



Tarefa 5. Utilizar a sentenza *while*.

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:

Result Grid		Filter Rows:
	vNumero	vSuma
▶	101	5050



Tarefa 6. Utilizar a sentenza *repeat*.

1.2.2.3 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 insercións.

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 borrarla. 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 caracteres ? como marcadores da posición dos parámetros que van a recibir os valores no momento da execución. Os caracteres ? 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
```

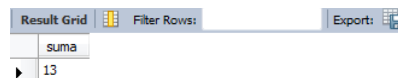
O alcance para unha sentenza preparada é a sesión na que se crea, e non está dispoñible para outras sesións. Cando se utiliza sentenzas preparadas nun programa almacenado, no caso de que non se borren dentro do programa, seguen existindo aínda que finalice a execución do programa e poden ser executadas 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:

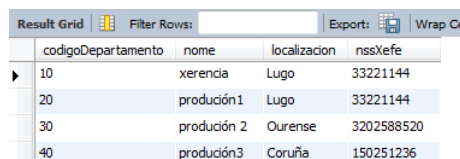


suma
13

- 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	producción1	Lugo	33221144
30	producción 2	Ourense	3202588520
40	producción3	Coruña	150251236



Tarefa 7. Utilizar sentenzas preparadas.

1.3 Tarefas

As tarefas propostas son as seguintes:

- Tarefa 1. Declarar variables, condicións, e manipuladores.
- Tarefa 2. Asignar valores a variables.
- Tarefa 3. Utilizar a sentenza *if*.
- Tarefa 4. Utilizar a sentenza *case*.
- Tarefa 5. Utilizar a sentenza *while*.
- Tarefa 6. Utilizar a sentenza *repeat*.
- Tarefa 7. Utilizar sentenzas preparadas.

1.3.1 Tarefa 1. Declarar variables, condicións, e manipuladores

A tarefa consiste en crear un guión coas seguintes sentenzas e comprobar con Workbench que non se cometeron erros de sintaxe:

- Declarar unha variable chamada *vNumero* que é de tipo numérica, enteira, e asignarlle o valor 0 cando se crea.
- Declarar unha condición á que se lle vai a asignar o nome *claveDuplicada* para o caso que se produza o código de erro de MySQL 1062.
- Declarar unha variable chamada *vFinal* de tipo bit e asignarlle o valor 0, e declarar un manipulador que lle asigne o valor 1 a esa variable cando se produza unha condición de

erro NOT FOUND. Cando se produce o erro, debería continuar a execución do programa almacenado.

Solución

```
/*Declarar unha variable chamada vNumero que é de tipo numérica, enteira, e asignarlle o valor 0 cando se crea.*/
declare vNumero integer default 0;
/* Declarar unha condición á que se lle vai a asignar o nome claveDuplicada para o caso que se produza o código de erro de MySQL 1062.*/
declare claveDuplicada condition for 1062;
/*Declarar unha variable chamada vFinal de tipo bit e asignarlle o valor 0, e declarar un manipulador que lle asigne o valor 1 a esa variable cando se produza unha condición de erro NOT FOUND. Cando se produce o erro debería continuar a execución do programa almacenado.*/
declare vFinal bit default 0;
declare continue handler for not found set vFinal = 1;
```

A sentenza *declare* só se pode utilizar dentro dun bloque de programación dun programa almacenado, así que de querer ver se a sintaxe é correcta coa axuda do editor gráfico de Workbench, podemos incluílas na creación dun procedemento almacenado cun único bloque de programación:

```
delimiter //
create procedure proba()
begin
/*Declarar unha variable chamada vNumero que é de tipo numérica, enteira, e asignarlle o valor 0 cando se crea.*/
declare vNumero integer default 0;
/* Declarar unha condición á que se lle vai a asignar o nome claveDuplicada para o caso que se produza o código de erro de MySQL 1062.*/
declare claveDuplicada condition for 1062;
/*Declarar unha variable chamada vFinal de tipo bit e asignarlle o valor 0, e declarar un manipulador que lle asigne o valor 1 a esa variable cando se produza unha condición de erro NOT FOUND. Cando se produce o erro debería continuar a execución do programa almacenado.*/
declare vFinal bit default 0;
declare continue handler for not found set vFinal = 1;
end;
//
delimiter ;
```

1.3.2 Tarefa 2. Asignar valores a variables

A tarefa consiste en crear un guión coas seguintes sentenzas de asignación e comprobar que funcionan:

- Asignar o valor 'servido' a unha variable local chamada *vEstado*.
- Asignar o valor 26 a unha variable de usuario chamada *idade*.
- Asignar o valor 'OFF' a unha variable de sistema chamada *autocommit*.
- Asignar a unha variable local chamada *vNomeProvincia* o contido da columna do *nomeProvincia* da provincia que ten como código o valor 15, tomando os datos da táboa *provincia* na base de datos *utilidades*.

Solución

```
declare vEstado char(8);
```

```

declare vNomeProvincia varchar(35) default null;
-- Asignar o valor 'servido' a unha variable local chamada vEstado.
set vEstado = 'servido';
-- Asignar o valor 26 a unha variable de usuario chamada idade.
set @idade = 26;
select @idade := 26; # tamén se podería facer con esta sentenza
-- Asignar o valor 'OFF' a unha variable de sistema chamada autocommit.
set @@autocommit = 0;
/*Asignar a unha variable local chamada vNomeProvincia o contido da columna
nomeProvincia da provincia que ten como código o valor 15, tomando os datos
da táboa provincia na base de datos utilidades.
*/
select nome into vNomeEmpregado
from practicas1.empregado
where dni = '44552010K';

```


Para probar que funcionan as ordes de asignación pódese crear un procedemento almacenado cun único bloque de programación:

```

set autocommit = 1;
drop procedure if exists asignarValores;
delimiter //
create procedure asignarValores()
begin
declare vEstado char(8);
declare vNomeEmpregado varchar(80) default null;
-- Asignar o valor 'servido' a unha variable local chamada vEstado.
set vEstado = 'servido';
-- Asignar o valor 26 a unha variable de usuario chamada idade.
set @idade = 26;
select @idade := 26; # tamén se podería facer con esta sentenza
-- Asignar o valor 'OFF' a unha variable de sistema chamada autocommit.
set @@autocommit = 0;
/*Asignar a unha variable local chamada vNomeProvincia o contido da columna
nomeProvincia da provincia que ten como código o valor 15, tomando os datos
da táboa provincia na base de datos utilidades.
*/
select nome into vNomeEmpregado
from practicas1.empregado
where dni = '44552010K';
select vEstado,@idade,@@autocommit,vNomeEmpregado;
end;
//
delimiter ;
call asignarValores();

```

Resultado da execución:



vEstado	@idade	@@autocommit	vNomeEmpregado
servido	26	0	Bernardez Varela, Luis

1.3.3 Tarefa 3. Utilizar a sentenza if

A tarefa consiste en escribir e comprobar o funcionamento dun bloque de programación que permita mostrar o texto 'positivo' no caso de que a variable *vNumero* tome un valor maior que cero, 'negativo' en caso de que tome un valor menor a 0, e 'cero', no caso que tome o valor 0.

Solución

```
if vNumero = 0 then select 'zero';  
elseif vNumero > 0 then select 'positivo';  
else select 'negativo';  
end if;
```

Para probar o funcionamento da sentenza if, pódese crear un procedemento almacenado cun único bloque de programación:

```
/* Escribir un bloque de programación no que se mostre o texto 'positivo'  
no caso de que a variable vNumero tome un valor maior que zero, 'negativo'  
en caso de que tome un valor menor a 0, e 'zero', no caso que tome o valor 0.*/  
drop procedure if exists tarefa3;  
delimiter //  
create procedure tarefa3()  
begin  
declare vNumero integer;  
set vNumero = -90;  
if vNumero = 0 then select 'zero';  
elseif vNumero > 0 then select 'positivo';  
else select 'negativo';  
end if;  
end;  
//  
delimiter ;  
call tarefa3();
```

Resultado da execución:



negativo

1.3.4 Tarefa 4. Utilizar a sentenza case

A tarefa consiste en escribir e comprobar que funcione un bloque de programación, no que se mostre a descrición do estado civil dunha persoa, en función do valor que toma a variable *vCodigo* que garda o código do estado civil desa persoa. Táboa de interpretación de códigos de estado civil:

Código	Descrición
S	Solteiro/a
C	Casado/a
D	Divorciado/a
V	Viúvo/a
U	Unión Libre/a
P	Separado/a

Solución

```
case vCodigo  
when 'S' then select 'Solteiro/a';  
when 'C' then select 'Casado/a';  
when 'D' then select 'Divorciado/a';  
when 'V' then select 'Viúvo/a';  
when 'U' then select 'Unión Libre';
```

```

when 'P' then select 'Separado/a';
else select 'Erro no código';
end case;

```

Para probar o funcionamento da sentenza *case*, pódese crear un procedemento almacenado cun único bloque de programación:

```

delimiter //
create procedure estado_civil ()
begin
    declare vCodigo varchar(1) default 'b';
    case vCodigo
        when 'S' then select 'Solteiro/a';
        when 'C' then select 'Casado/a';
        when 'D' then select 'Divorciado/a';
        when 'V' then select 'Viúvo/a';
        when 'U' then select 'Unión Libre';
        when 'P' then select 'Separado/a';
        else select 'Erro no código';
    end case;
end
//
delimiter ;
call estado_civil();

```

Resultado da execución:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Erro no código			
▶ Erro no código			

1.3.5 Tarefa 5. Utilizar a sentenza while

A tarefa consiste en escribir e comprobar que funcione un bloque de programación, que utiliza unha sentenza *while*, para calcular a cantidade de números pares que hai entre dous valores que se gardan en dúas variables de tipo enteiro sen signo chamadas *vNumeroInicio* e *vNumeroFinal*.

```

declare vNumero, vNumeroInicio, vNumeroFinal smallint unsigned;
declare vContadorPares smallint unsigned default 0;
set vNumeroInicio = 25;
set vNumeroFinal = 50;
set vNumero = vNumeroInicio;
while vNumero <= vNumeroFinal do
    if vNumero % 2 = 0 then
        set vContadorPares = vContadorPares + 1;
    end if;
    set vNumero = vNumero + 1;
end while;

```

Para probar o funcionamento da sentenza *while*, pódese crear un procedemento almacenado cun único bloque de programación:

```

/*Escribir un bloque de programación, utilizando unha sentenza while, que calcule a
cantidade de números pares que hai entre dous valores que se gardan en dúas variables de
tipo enteiro sen signo chamadas vNumeroInicio e vNumeroFinal*/
drop procedure if exists contarPares;
delimiter //
create procedure contarPares()
begin
    declare vNumero, vNumeroInicio, vNumeroFinal smallint unsigned;
    declare vContadorPares smallint unsigned default 0;

```

```

set vNumeroInicio = 25;
set vNumeroFinal = 50;
set vNumero = vNumeroInicio;
while vNumero <= vNumeroFinal do
  if vNumero % 2 = 0 then
    set vContadorPares = vContadorPares + 1;
  end if;
  set vNumero = vNumero + 1;
end while;
select vNumeroInicio, vNumeroFinal, vContadorPares;
end;
//
delimiter ;
call contarPares();

```

Resultado da execución:

Result Grid				Filter Rows:	Export:
	vNumeroInicio	vNumeroFinal	vContadorPares		
▶	25	50	13		

1.3.6 Tarefa 6. Utilizar a sentenza repeat

A tarefa consiste en escribir e comprobar que funcione un bloque de programación, que utiliza a sentenza *repeat*, para calcular o número de intentos realizados ata acertar o valor entre 1 e 100 almacenado nunha variable, xerando un número aleatorio en cada volta do bucle.

Solución

```

declare vNumero, vNumeroAleatorio tinyint unsigned default 0;
declare vContadorIntentos smallint unsigned default 0;
set vNumero = 25;
repeat
  /*seleccionar un número aleatorio entre 1 e 100*/
  set vNumeroAleatorio = floor(1 + (rand() * 99));
  set vContadorIntentos = vContadorIntentos + 1;
until vNumero = vNumeroAleatorio
end repeat;

```

Para probar o funcionamento da sentenza *repeat*, pódese crear un procedemento almacenado cun único bloque de programación:

```

/*Escribir un bloque de programación, utilizando a sentenza repeat, que calcule
o número de intentos realizados ata acertar o valor entre 1 e 100, almacenado
nunha variable, xerando un número aleatorio en cada execución do bucle.
*/
drop procedure if exists buscarNumero;
delimiter //
create procedure buscarNumero()
begin
  declare vNumero, vNumeroAleatorio tinyint unsigned default 0;
  declare vContadorIntentos smallint unsigned default 0;
  set vNumero = 25;
  repeat
    /*seleccionar un número aleatorio entre 1 e 100*/
    set vNumeroAleatorio = floor(1 + (rand() * 99));
    set vContadorIntentos = vContadorIntentos + 1;
  until vNumero = vNumeroAleatorio
  end repeat;

```

```

select vNumero, vNumeroAleatorio, vContadorIntentos;
end;
//
delimiter ;
call buscarNumero();

```

Resultado da execución que dependerá dos números aleatorios xerados:

Result Grid	Filter Rows:	Export:
vNumero	vNumeroAleatorio	vContadorIntentos
25	25	39

1.3.7 Tarefa 7. Utilizar sentenzas preparadas

A tarefa consiste en crear dúas sentenzas preparadas:

- Tarefa 7.1. Crear unha sentenza preparada que insira unha columna na táboa *provincia* da base de datos *utilidades*, pasándolle os valores contidos nas variables de usuario chamadas *codigo* e *nome*.
- Tarefa 7.2. Crear unha sentenza preparada que mostre os nomes das táboas dunha base de datos. O nome da base de datos se obtén dunha variable chamada *nomeBD*.

Solución

■ Tarefa 7.1

```

/* Crear unha sentenza preparada que insira unha columna na táboa departamento
da base de datos practicas1, pasándolle os valores para as columnas en variables
de usuario.*/
prepare insertDepartamento from 'insert into practicas1.departamento values (?, ?, ?, ?)';
set @codigo = '50';
set @nome = 'compras';
set @localizacion = 'Lugo';
set @nssXefe = '33221144';
execute insertDepartamento using @codigo, @nome, @localizacion, @nssXefe;
select * from practicas1.departamento;
deallocate prepare insertDepartamento;

```

Resultado da execución:

Result Grid		Filter Rows:	Edit:
codigoDepartamento	nome	localizacion	nssXefe
10	xerencia	Lugo	33221144
20	producción1	Lugo	33221144
30	producción 2	Ourense	3202588520
40	producción3	Coruña	150251236
50	compras	Lugo	33221144
NULL	NULL	NULL	NULL

Columna inserida

■ Tarefa 7.2

```

/* Crear unha sentenza preparada que mostre os nomes das táboas dunha base
de datos. O nome da base de datos se obtén dunha variable chamada nomeBD.*/
set @nomeBD = 'practicas1';
set @senteza = concat('select table_name from information_schema.tables where
table_schema = ', @nomeBD, '');
prepare tablasBD from @senteza;
execute tablasBD;
deallocate prepare tablasBD;

```

Resultado da execución:

Result Grid		Filter Rows:	Export:
	table_name		
▶	actuacion		
	ciclista		
	concerto		
	contador		
	departamento		
	empregado		
	equipo		
	fabricante		
	grupo		
	nacionalidade		
	pelicula		
	piso		
	representante		
	xenero		