

## Manexo de datos. Inserción, Borrado e Actualización

<b>1. Manipulación de datos con SQL .....</b>	<b>3</b>
1.1 Sentenza INSERT .....	3
Exemplos de sentenzas de inserción .....	6
Opción VALUES.....	6
Opción SET.....	7
Opción SELECT.....	7
1.2 Sentenza Replace .....	8
1.3 Modo SQL do servidor MySQL e valores asignados ás columnas .....	8
1.4 Restricións de integridade e consistencia da información .....	10
1.4.1 Restricións de clave primaria (PRIMARY KEY) .....	10
1.4.2 Restricións de unicidade (UNIQUE).....	11
1.4.3 Restricións de valor nulo (NOT NULL).....	11
1.4.4 Restricións de claves foráneas (FOREIGN KEY) .....	11
1.4.5 Restricións DEFAULT .....	12
1.4.6 Restricións CHECK.....	12
1.4.7 Restricións dos tipos ENUM, e SET .....	13
1.5 Sentenza UPDATE .....	13
1.6 Sentenza DELETE.....	15
1.7 Borrado lóxico de filas dunha táboa .....	17
1.8 Uso de subconsultas nas sentenzas de edición de datos .....	18
1.9 Guións de sentenzas de edición de datos nas táboas .....	20

# 1. Manipulación de datos con SQL

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

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

## 1.1 Sentenza INSERT

A sentenza INSERT permite inserir novas filas nas táboas existentes. Existen varias opcións para realizar a inserción: os valores poden escribirse como un conxunto de valores pechados entre parénteses para cada fila nova, unha colección de expresións de asignación, ou como unha sentenza SELECT.

Opción especificando os valores a inserir de forma explícita con VALUE:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]
[INTO] nome_táboa [(lista_columnas)]
{VALUES | VALUE} ({expresión | DEFAULT | NULL},...),(...),...
[ ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ... ]
```

Opción especificando os valores a inserir de forma explícita con SET:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]
[INTO] nome_táboa
SET col_name={expresión | DEFAULT | NULL}, ...
[ ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ... ]
```

Opción inserindo filas con datos contidos noutras táboas:

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY]
[INTO] nome_táboa [(lista_columnas)]
SELECT ...
[ ON DUPLICATE KEY UPDATE col_name=expr [, col_name=expr] ... ]
```

Consideracións sobre a sintaxe anterior:

- As partes opcionais LOW\_PRIORITY, DELAYED e HIGH\_PRIORITY permiten indicar o nivel de prioridade que ten a operación de inserción.

Cando se utiliza a opción LOW\_PRIORITY, o servidor agarda a que non haxa clientes lendo na táboa para facer a inserción. Cando se utiliza a opción HIGH\_PRIORITY, anúlase o efecto da opción *--low-priority-updates*, se estivera habilitada. É dicir, dase prioridade as operacións de insert, update, delete fronte as de select. Estas opcións afectan só a táboas con motores que utilizan bloqueo a nivel de táboa (MyISAM, MEMORY, MERGE).

Cando se utiliza a opción DELAYED, o servidor garda a sentenza nun buffer de memoria e libera ao cliente no caso de que a táboa estea bloqueada por outros clientes; cando a táboa queda libre, o servidor fai a inserción. **Esta opción está en desuso a partir de MySQL 5.6.6.**

- nome\_táboa* é o nome da táboa na que se van a inserir datos.

- *lista\_columns* é a relación de columnas nas que se van a inserir datos, separadas por comas. Utilizárase cando se especifican os datos como unha lista de valores (VALUES), ou cando se collen os datos mediante unha consulta (SELECT).

As columnas que non figuran na lista toman o valor que teñan definido por defecto, ou o valor por defecto implícito se non teñen definido un valor por defecto. Os valores por defecto implícitos son 0 para os tipos numéricos, a cadea valeira (") para os tipos de cadeas de caracteres, e o valor "zero" para tipos de data e hora.

Este é o comportamento de MySQL cando non se está a executar no modo SQL estrito. Cando se activa o modo SQL estrito, prodúcese un erro se non se especifican valores explicitamente para todas as columnas que non teñen definido un valor por defecto.

As columnas de tipo AUTO\_INCREMENT non deben aparecer na lista de columnas, e neste caso o servidor asígnalles o valor seguinte ao da última fila que se inseriu.

No caso de non poñer unha lista de columnas a continuación do nome da táboa, considérase que se van a inserir datos en todas as columnas, na orde en que se crearon. No caso de non utilizar un cliente gráfico e non recordar os nomes das columnas ou a orde en que se crearon, pódese utilizar a sentenza DESCRIBE para consultalo. Exemplo:

1 • describe practicas5.departamento;

Field	Type	Null	Key	Default	Extra
codigo	tinyint(3) unsigned	NO	PRI	NULL	auto_increment
nome	char(25)	YES		NULL	
tipo	enum('H','B','A')	NO		NULL	
cidade	varchar(35)	YES		NULL	
id_provincia	smallint(6)	YES	MUL	NULL	

A lista de columnas utilízase no caso de que non se introduzan valores para todas as columnas, ou se escriban os valores en distinta orde á que teñen as columnas no esquema da táboa.

- Na opción 2, SET permitirá indicar explicitamente os nomes das columnas e os valores que van a tomar mediante expresións de asignación.
- Na opción 1, VALUES | VALUE permitirán indicar explicitamente a lista de valores que van a tomar as columnas da fila separados por comas e pechados entre parénteses. VALUE é un sinónimo de VALUES.

A expresión correspondente a cada columna debe devolver un valor do mesmo tipo que a columna. No caso de non ser do mesmo tipo, MySQL fai a conversión de tipos.

Nunha expresión pódese facer referencia a unha columna das que aparecen antes na lista de columnas. Tamén se poden utilizar as palabras reservadas NULL para facer referencia ao valor nulo, ou DEFAULT para facer referencia ao valor por defecto

Exemplo:

```
insert into nome_taboa (col1, col2, col3, col4) values (15, col1 * 2, null, default);
```

Unha excepción no uso de referencias a outras columnas son as columnas que teñen a propiedade `AUTO_INCREMENT`. Calquera referencia a esas columnas toma o valor 0.

O número de valores ten que coincidir co número de columnas relacionadas na lista de columnas.

MySQL permite inserir varias filas cunha única sentenza `INSERT...VALUE`, incluíndo varias listas de valores, cada unha pechada entre parénteses, e separadas por comas. Exemplo:

```
insert into nome_taboa (col1, col2, col3)
values (15, null, default),(25, 'Proba1', null),(55, 'Proba2', 'Proba3');
```

Esta sentenza insire tres filas na táboa.

- Na opción 3, as filas que se van a inserir e os valores que toman as columnas desas filas, obtéñense da execución dunha sentenza `SELECT`.
- Cando se utiliza a cláusula `ON DUPLICATE KEY UPDATE` e se fai a inserción dunha fila que pode provocar un valor duplicado nunha clave `PRIMARY` ou `UNIQUE`, faise unha actualización na fila que xa existe, modificando os valores das columnas que se indican na cláusula. Exemplo:

```
insert into taboa1 (a,b,c) values (1,2,3)
on duplicate key update c=c+1;
```

A sentenza modifica o contido da columna `c`, sumándolle unha unidade, no caso que exista unha fila co valor 1 na columna `a`, supoñendo que esta é a clave primaria. O que se actualiza é a fila que xa existe.

Isto é útil nas ocasións nas que nos pode interesar inserir un rexistro ou actualizalo si este xa existe. Para facer isto podemos ter varias opcións, como por exemplo estas dúas:

- Opción A:
  - Comprobar si existe o rexistro.
  - Si non existe, o inserimos.
  - Si existe, actualizámolo.
- Opción B:
  - Intentamos inserir o rexistro.
  - Si se produce erro, o actualizamos.

Pero MySQL proporcionanos unha maneira máis elegante de facer isto. É a sentenza `ON DUPLICATE KEY UPDATE`, que tamén nos permite a súa aplicación cando de sexamos inserir máis dun rexistro.

```
insert into taboa1 (a,b,c) values (1,2,3),(4,5,6)
on duplicate key update b=values(b), c=values(c);
```

Desta forma, collerá os valores b=2 e c=3 para a clave a=1 e b=5 e c=6 para a clave a=4

## Exemplos de sentenzas de inserción

Exemplos de sentenzas INSERT na base de datos *practicass5*.

### Opción VALUES

- Sen utilizar lista de columnas

```
insert into empregado
values (31852963,'Varela Mendez','Luisa',4,29500,54528788);
```

Asígnanse valores para todas as columnas da táboa. Os valores van ordenados segundo a definición do esquema da táboa.

- Utilizando lista de columnas

```
insert into departamento (nome, tipo, cidade, id_provincia)
values ('OficinaProba1', 'H', 'Ares', 15);
```

Asígnanse valores poñendo de forma explícita as columnas e os valores que se van a asignar. O primeiro valor da lista de valores ('OficinaProba1') asígnase á primeira columna da lista de columnas (*nome*), e así co resto. O número de valores ten que coincidir co número de columnas, e os tipos de datos teñen coincidir cos tipos de columnas.




Nesta sentenza non se incluíu na lista de columnas a columna *codigo* xa que é de tipo AUTO\_INCREMENT, e por tanto asígnaselle automaticamente o valor seguinte ao que ten esa columna para a última fila que se inseriu. Consulta de datos:

	codigo	nome	tipo	cidade	id_provincia
	11	OficinaProba1	H	Ares	15

- Utilizando a sintaxe estendida de MySQL para poder inserir máis dunha fila cunha sentenza INSERT

```
insert into departamento (nome, tipo, cidade, id_provincia)
values ('OficinaProba2', 'H', 'Ferrol', 15),
       ('OficinaProba3', 'A', 'Ourense', 32),
       ('OficinaProba4', 'H', 'Allariz', 32);
```




Fai a inserción de tres filas na táboa, e mostra a mensaxe indicando o número de filas inseridas. Consulta de datos:

Result Grid			Filter Rows:		Edit:
	codigo	nome	tipo	cidade	id_provincia
	12	OficinaProba2	H	Ferrol	15
	13	OficinaProba3	A	Ourense	32
	14	OficinaProba4	H	Allariz	32

## Opción SET



```
insert into departamento
set nome = 'OficinaProba5',
    tipo = 'A',
    cidade = 'Lugo',
    id_provincia = 27;
```

Insire unha fila na táboa cos valores que se deron a cada columna. Consulta de datos:

Result Grid				Filter Rows:	Edit: 
	codigo	nome	tipo	cidade	id_provincia
	15	OficinaProba5	A	Lugo	27

A función `LAST_INSERT_ID()` permite obter o último valor que se almacenou na columna con propiedade `AUTO_INCREMENT` da táboa sobre a que se fixo a última operación de inserción. Por exemplo, se a última sentenza `INSERT` que se executou referíase á táboa *departamento* que ten a columna *codigo* coa propiedade `AUTO_INCREMENT`:

```
select last_insert_id();
```

Result Grid			Filter Rows:
	LAST_INSERT_ID()		
	15		

## Opción SELECT

Esta inserción correspóndese coa terceira opción das expostas anteriormente e permite inserir novas filas nunha táboa copiando os datos que xa están gravados noutras táboas, tomando como entrada o resultado dunha consulta feita coa sentenza `SELECT`.

A sentenza `SELECT` debe ter na lista de selección o mesmo número de columnas, e o mesmo tipo de datos, que hai na lista de columnas, ou ben, o mesmo número de columnas que ten o esquema da táboa, se non se pon a lista de columnas. Exemplo:

```
/* Créase unha táboa temporal para gardar datos dos departamentos da provincia de Lugo
co mesmo esquema que a táboa departamento */
create temporary table departamento_lugo like departamento;
/* Faise a inserción dos datos dos departamentos de Lugo (contéñen o valor 27 na
columna id_provincia). */
insert into departamento_lugo
select * from departamento where id_provincia = 27;
-- Consúltanse os datos da táboa temporal
select * from departamento_lugo;
```

	codigo	nome	tipo	cidade	id_provincia
1	Central	H	Lugo	27	
2	Oficina1	H	Monforte	27	
6	Oficina5	A	Villalba	27	
8	Oficina7	H	Lugo	27	
15	OficinaProba5	A	Lugo	27	

## 1.2 Sentenza Replace

A sentenza REPLACE é unha alternativa para INSERT que só se diferencia en que se existe algunha fila anterior co mesmo valor para unha clave primaria (PRIMARY KEY) ou única (UNIQUE), elimínase a fila que xa existía con ese valor na clave, e insírese no seu lugar a fila cos novos valores. Exemplo:

```
insert into departamento (codigo,nome,tipo,cidade,id_provincia)
values (16,'OficinaProba10','H','Carballo',15);
```

52 20:45:40 insert into departamento (codigo,nome,tipo,cidade,id\_provincia) values (16,'OficinaProba10'... Error Code: 1062. Entrada duplicada '16' para la clave 'PRIMARY'

Xa existe un departamento co valor 16 na clave primaria (*codigo*), o que fai que se mostre unha mensaxe de erro e a fila non pode ser inserida. Execútase a sentenza REPLACE:

```
replace into departamento (codigo,nome,tipo,cidade,id_provincia)
values (16,'OficinaProba10','H','Carballo',15);
```

50 20:37:54 replace into departamento (codigo,nome,tipo,cidade,id\_provincia) values (16,'OficinaProba10','H','Carballo',15) 2 row(s) affected

A mensaxe informa que foron afectadas dúas filas. Unha é a que existía co mesmo valor na clave primaria e outra é a nova fila que a substitúe. Consulta de datos:

	codigo	nome	tipo	cidade	id_provincia
16	OficinaProba10	H	Carballo	15	

## 1.3 Modo SQL do servidor MySQL e valores asignados ás columnas

O comportamento do servidor MySQL cando se producen erros asociados aos tipos de datos dependerá de se está traballando en modo estrito ou non. No caso de non estar activado o modo estrito, os datos que non se axustan á definición da columna son engadidos á táboa pero se mostra unha mensaxe de alerta (*warning*). No caso de estar activado o modo estrito, xa non se permite que os datos sexan engadidos.

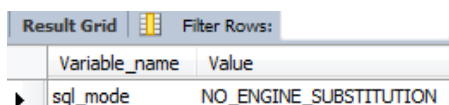
*'MySQL server puede operar en distintos modos SQL, y puede aplicar estos modos de forma distinta a diferentes clientes. Esto permite que cada aplicación ajuste el modo de operación del servidor a sus propios requerimientos.'*



*Los modos definen qué sintaxis SQL debe soportar MySQL y que clase de chequeos de validación de datos debe realizar. Esto hace más fácil de usar MySQL en distintos entornos y usar MySQL junto con otros servidores de bases de datos.'*<sup>1</sup>

Para ver o modo SQL activo no servidor, pódese ver o contido da variable de sistema `sql_mode`, executando unha das sentenzas:

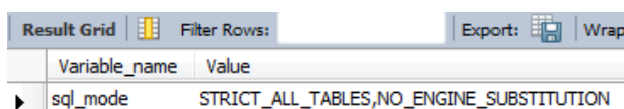
```
select @@sql_mode; # ou ben:
show variables like 'sql_mode';
```



Variable_name	Value
sql_mode	NO_ENGINE_SUBSTITUTION

Para cambiarlle o valor á variable `sql_mode`, pódese executar unha sentenza de asignación SET ou cambiar o valor da variable no ficheiro de configuración de MySQL (`my.ini` ou `my.cnf`). Exemplo:

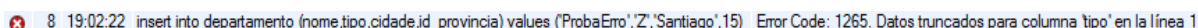
```
set sql_mode = concat('STRICT_ALL_TABLES', @@sql_mode);
show variables like 'sql_mode';
```



Variable_name	Value
sql_mode	STRICT_ALL_TABLES,NO_ENGINE_SUBSTITUTION

Cando está activado o modo SQL estrito e se asignan a unha columna valores fóra do rango permitido, móstrase unha mensaxe de erro e abórtase a inserción. Exemplo de inserción cando está activado o modo SQL estrito:

```
insert into departamento (nome,tipo,ciade,id_provincia)
values ('ProbaErro','Z','Santiago',15);
```



8 19:02:22 insert into departamento (nome,tipo,ciade,id\_provincia) values ('ProbaErro','Z','Santiago',15) Error Code: 1265. Datos truncados para columna 'tipo' en la línea 1

Algunhas consideracións sobre os valores asignados ás columnas que poden provocar mensaxes de advertencia (*warning*) cando non está activado o modo SQL estrito:

- Cando se asigna un valor NULL a unha columna declarada NOT NULL, prodúcese unha mensaxe de advertencia (*warning*) e gárdase na columna o valor por defecto implícito (0 para columnas de tipo numérico, unha cadea baleira " para columnas de tipo cadea, ou o valor 0 para columnas tipo data e hora).
- Cando se asigna un valor fora de rango a unha columna de tipo numérico, o valor trúncase.
- Cando se asigna un valor tipo cadea a unha columna de tipo numérico, cópanse os díxitos numéricos se os tivera e non se teñen en conta o resto de caracteres, no caso de non ter ningún dígito numérico gárdase o valor 0. Exemplo: o valor '25.23 euros' almacénase como 25.23 nunha columna de tipo numérico e non se ten en conta o texto euros.
- Cando se asigna unha cadea que excede do tamaño dunha columna tipo cadea, trúncase ata a lonxitude máxima.

---

<sup>1</sup> MySQL 5.6. Reference Manual, apartado 5.1.7 Server SQL Modes.

- Cando se asigna un valor non válido a unha columna relacionada co tempo (data, hora ou data e hora), gárdase o valor cero.
- Cando se asigna un valor diferente dos permitidos a unha columna tipo ENUM, gárdase na columna o valor por defecto implícito. Exemplo:

```
insert into departamento (nome, tipo, cidade, id_provincia)
values ('ProbaErro', 'Z', 'Santiago', 15);
```

48 20:18:34 insert into departamento (nome, tipo, cidade, id\_provincia) values ('ProbaErro', 'Z', 'Santiago', 15) 1 row(s) affected, 1 warning(s): 1265 Datos truncados para colu...

A columna *codigo* foi definida como ENUM('H','B','A'). O valor 'Z' non é un valor dos permitidos, pero como o servidor non está en modo SQL estrito, a fila insírese na táboa e nesa columna gárdase o valor por defecto implícito ("), unha cadea baleira. Consulta de datos:

codigo	nome	tipo	cidade	id_provincia
16	ProbaErro		Santiago	15

## 1.4 Restricións de integridade e consistencia da información

Os datos almacenados nas columnas dunha táboa dunha base de datos relacional deberían cumprir coas:

- Restricións de clave primaria (PRIMARY KEY).
- Restricións de unicidade (UNIQUE).
- Restricións de valor nulo (NOT NULL).
- Restricións de claves foráneas (FOREIGN KEY).
- Restricións DEFAULT.
- Restricións CHECK.
- Restricións dos tipos ENUM, e SET.

Cando se executan sentenzas que fan modificacións nos datos das bases de datos hai que asegurarse que cumpren estas restricións, para poder seguir mantendo a integridade dos datos. Se as restricións establecéronse correctamente no momento do deseño e na creación das bases de datos, o servidor fará ese traballo por nós, avisando cando algunha sentenza vulnere algunha das restricións impostas. De aí a importancia do deseño correcto nunha BD relacional.

### 1.4.1 Restricións de clave primaria (PRIMARY KEY)

As restricións que debe cumprir a clave primaria son:

- As columnas que forman parte da clave primaria non poden tomar o valor NULL.
- Os valores que toman as columnas que forman a clave primaria teñen que ser únicos nunha táboa. Isto significa que para cada fila, a clave primaria ten que tomar un valor diferente ao resto das filas.

### 1.4.2 Restricións de unicidade (UNIQUE)

As restricións de unicidade refírense a que as columnas definidas como de tipo UNIQUE teñen que tomar valores únicos para cada fila. Non pode haber dúas filas que teñan o mesmo valor na columna.

### 1.4.3 Restricións de valor nulo (NOT NULL)

As restricións de valor nulo indican que as columnas ás que se lle asigna a propiedade NOT NULL, teñen que ter sempre un valor válido non admitindo o valor descoñecido (NULL).

### 1.4.4 Restricións de claves foráneas (FOREIGN KEY)

Unha columna definida como clave foránea só pode ter como valor algún dos da clave primaria á que fai referencia ou o valor NULL se é que ese valor está permitido.

As táboas transaccionais de MySQL (por exemplo, as que utilizan o motor de almacenamento InnoDB) son as únicas táboas de MySQL que soportan restricións de claves foráneas. Tanto a táboa que contén a restrición de clave foránea como a táboa á que se fai referencia, teñen que ser táboas transaccionais e non poden ser táboas temporais.

Sintaxe para definir unha restrición de clave foránea en táboas transaccionais:

```
[CONSTRAINT nome_restrición] FOREIGN KEY (nome_columna [,nome_columna] ...)
REFERENCES nome_de_táboa (nome_índice [,nome_índice] ...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

- Se existe *nome\_restrición*, debe ser único na base de datos; se non se subministra, InnoDB crea o nome automaticamente.
- O servidor non permitirá realizar ningunha INSERT ou UPDATE que intente asignar un valor a unha columna definida como clave foránea que non coincida con algún valor da clave primaria da táboa á que fai referencia, ou tome o valor NULL no caso de que estea permitido este valor para esa columna.
- Cando se intenta realizar unha operación UPDATE ou DELETE sobre unha fila da táboa pai (a que figura en REFERENCES) e existen táboas que conteñen claves foráneas que sinalan a esa fila, terase en conta a acción especificada nos apartados ON DELETE e ON UPDATE da cláusula FOREIGN KEY. As accións permitidas en MySQL son:

- CASCADE.

Borra ou actualiza a fila na táboa pai e automaticamente borra ou actualiza as filas con claves foráneas que fan referencia a esa fila.

- SET NULL

Borra ou actualiza a fila na táboa pai e garda o valor NULL na columna que é clave foránea e fai referencia a esa fila. Para poder empregar esta opción a columna que é clave foránea debe permitir almacenar o valor NULL, é dicir, non ter NOT NULL na súa definición.

- NO ACTION

Non é estándar ANSI SQL-92 e significa que non está permitido borrar ou modificar

unha fila na táboa pai, se hai algunha fila que teña unha clave foránea que faga referencia a ela.

- **RESTRICT**

En MySQL, NO ACTION e RESTRICT son equivalentes.

- **SET DEFAULT**

Pon na clave foránea o valor definido por defecto para esa columna. Non está implementada en MySQL.

Cando se intenta facer unha operación non permitida polas restricións de integridade referencial, prodúcese un erro e móstrase unha mensaxe de erro do tipo:

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('practicas5`.`empregado`, CONSTRAINT `fk\_empregado\_empregado1` FOREIGN KEY (`dni\_xefe`) REFERENCES `empregado` (`dni`) ON DELETE NO ACTION ON UPDATE NO ACTION)

Na mensaxe de erro infórmase de que se intenta modificar ou borrar una fila pai (que hai filas doutras táboas que fan referencia a ela) e mostra información da definición de clave foránea que provoca o erro. Na mensaxe anterior, a fila que se intenta borrar ou modificar corresponde a un empregado que ten un dni que coincide co valor que toma en algunha fila a columna *dni\_xefe*, definida como clave foránea e que ten asociada a operación NO ACTION para o caso de borrado e modificación.

- Para mellorar o rendemento do funcionamento das claves foráneas e das restricións de integridade asociado a elas, as columnas definidas como clave foránea e a clave primaria á que fai referencia deben ter asociado un índice. A partir da versión 5.0 de MySQL, cando se crea unha clave foránea créase o índice de forma automática.
- Pódese desactivar a verificación de restricións de clave foránea de forma temporal, mediante a variable `FOREIGN_KEY_CHECKS`, por exemplo para facer operacións de mantemento de táboas, como pode ser a recuperación de copias de seguridade. É posible establecer o valor desta variable dende a liña de comandos do cliente MySQL ou engadila nun script de sentenzas SQL. Exemplo:

```
set foreign_key_checks = 0 # desactiva a verificación de restricións de clave foráneas
set foreign_key_checks = 1 # activa a verificación de restricións de clave foráneas
```

### 1.4.5 Restricións DEFAULT

Pódese asignar un valor por defecto a unha columna coa cláusula DEFAULT. Ese será o valor que toma a columna no caso de non asignarlle un valor de maneira explícita.

### 1.4.6 Restricións CHECK

A restrición CHECK asocia unha restrición de valor a unha columna, poñendo entre parénteses unha expresión coas condicións que teñen que cumprir os datos almacenados nesa columna. Cando se insiren ou modifican datos nunha columna co atributo CHECK, compróbase que se verifiquen as condicións establecidas para esa columna.

### 1.4.7 Restricións dos tipos ENUM, e SET

Estes tipos de datos son propios de MySQL, e permiten asociar unha restrición de valor a unha columna poñendo entre parénteses o conxunto de valores válidos para a columna. As columnas de tipo ENUM só poden tomar un valor do conxunto de valores permitidos, e as de tipo SET poden tomar un ou máis valores do conxunto de valores permitidos.

Cando se insiren ou modifican datos nunha columna do tipo ENUM ou SET, hai que comprobar que os valores que toma a columna estean na lista de valores permitidos.

## 1.5 Sentenza UPDATE

Permite modificar os datos contidos nas táboas. Existen varias opcións de sintaxe.

Opción para modificar datos dunha soa táboa:

```
UPDATE [LOW_PRIORITY] nome_táboa
SET nome_columna = {expresión | DEFAULT | NULL}
    [, nome_columna = {expresión | DEFAULT | NULL}] [, ...]
[WHERE condición]
[ORDER BY expresión]
[LIMIT número_filas]
```

Opción para modificar datos de varias táboas:

```
UPDATE [LOW_PRIORITY] referencia_táboas
SET nome_columna = {expresión | DEFAULT | NULL}
    [, nome_columna = {expresión | DEFAULT | NULL}] [, ...]
[WHERE condición]
```

- A opción `LOW_PRIORITY` permite indicar o nivel de prioridade que ten a operación de actualización. Con esta opción, o servidor espera que non haxa clientes lendo na táboa para facer a modificación. Só afecta a táboas con motores que utilizan bloqueo a nivel de táboa (MyISAM, MEMORY, MERGE).
- Cando se utiliza a opción para actualizar varias táboas, a sentenza `UPDATE` actualiza as filas das táboas nomeadas en *referencia\_táboas* que cumpren as condicións establecidas na cláusula `WHERE`. Cada xogo de filas actualízase só unha vez, aínda que cumpran varias veces as condicións.
- A cláusula `SET` indica as columnas que se van a modificar e os valores novos que van a recibir. Para os novos valores que se asignan ás columnas nunha operación de modificación aplícase o visto no apartado '2.2.3.2. Modo SQL do servidor MySQL e valores asignados ás columnas'.
- A cláusula `WHERE` é opcional e especifica as filas que deben actualizarse. **Moi importante: se non se escribe a cláusula `WHERE`, actualízanse todas as filas.**
- A cláusula `ORDER BY` é opcional e indica a orde na que se actualizan as filas. Non se pode utilizar na actualización de múltiples táboas.

Pode ser útil en certas situacións como por exemplo para actualizar unha columna que non admite valores duplicados, engadíndolle unha unidade ao valor que ten a columna.

```
update táboa
set columna = columna + 1;
```

Supoñendo que a columna ten os valores 1,2,3,4 e 5. Cando se actualiza a primeira fila asígnaselle á columna o valor 2, que é o valor que ten a segunda fila nesa columna, polo que se produce un erro de '*clave duplicada*' e non se pode facer a actualización. Utilizando a cláusula `ORDER BY`:

```
update táboa
set columna = columna + 1
order by columna desc;
```

Neste caso a primeira fila que se actualiza é a que ten o valor maior, neste caso o 5. Despois da actualización a columna toma o valor 6 que non existe en ningunha outra fila, e desta maneira, solúciónase o problema de claves duplicadas.

- A cláusula LIMIT establece o número de filas a actualizar. Non se pode utilizar na actualización de múltiples táboas.

Exemplos de sentenzas UPDATE na base de datos *practicass5*:

- Actualizar todas as filas dunha táboa.

Aumentar nun 5% os valores da columna *tipo\_imposto* da táboa *irpf*.

```
update irpf
set tipo_imposto=round(tipo_imposto*1.05,2);
```

86 19:31:31 update irpf set tipo\_imposto=round(tipo\_imposto\*1.05,2) 7 row(s) affected Líneas correspondientes: 7 Cambiadas: 7 Avisos: 0

A mensaxe informa que foron modificadas as 7 filas que ten a táboa.

- Actualizar as filas que cumpren unha condición.

Aumentar nun 3% os valores das columnas *limite\_inferior* e *limite\_superior* da táboa *irpf*, para todas as filas excepto a primeira fila na que *limite\_inferior* ten o valor 0.

```
update irpf
set limite_inferior=round(limite_inferior*1.03,2),
    limite_superior=round(limite_superior *1.03,2)
where limite_inferior > 0 ;
```

88 20:01:48 update irpf set limite\_inferior=round(limite\_inferior\*1.03,2), limite\_superior=round(limit... 6 row(s) affected, 1 warning(s): 1264 Out of range value for column 'limit...

A mensaxe informa que foron modificadas 6 filas e mostra un aviso advertindo que o valor para a columna *limite\_superior* da fila 7 da táboa ten un valor fora de rango, e como non está activado o modo SQL estrito, trúncase o valor. O texto completo da mensaxe é o seguinte:

6 row(s) affected, 1 warning(s): 1264 Out of range value for column 'limite\_superior' at row 7 Líneas correspondientes: 6 Cambiadas: 6 Avisos: 1

- Actualizar un número de filas limitado.

Reducir un 5% o salario bruto dos empregados que teñen os 2 salarios máis altos, do departamento número 4.

```
update empleado
set salario_bruto=round(salario_bruto*0.95,2)
where departamento=4
order by salario_bruto desc
limit 2 ;
```

91 20:27:16 update empleado set salario\_bruto=round(salario\_bruto\*0.95,2) where departamen... 2 row(s) affected Líneas correspondientes: 2 Cambiadas: 2 Avisos: 0

A mensaxe informa que foron modificadas 2 filas.

- Actualizar columnas de máis dunha táboa.

Cambiar o tipo de departamento ao departamento número 1 que pasa a ser de tipo

'A', e ao mesmo tempo, aumentarlle un 5% ao salario bruto aos empregados dese departamento.

```
update empregado as em, departamento as de
set     de.tipo='A',
        em.salario_bruto=round(salario_bruto*1.05,2)
where  em.departamento=de.codigo
        and em.departamento=1;
```

✓ 110 20:55:19 update empregado as em, departamento as de set de.tipo='A', em.salario\_bruto=rou... 4 row(s) affected Líneas correspondientes: 4 Cambiadas: 4 Avisos: 0

Na cláusula WHERE faise a combinación das táboas relacionando cada fila da táboa empregado coa fila correspondente da táboa departamento. Se non se pon esta condición, a modificación faríase sobre a táboa formada polo produto cartesiano das dúas táboas. A condición inclúe tamén o resto de condicións que teñen que cumprir as filas que hai que modificar; neste caso, selecciónanse só os empregados que corresponden ao departamento número 1.

A mensaxe informa que foron modificadas 4 filas, 3 delas corresponden ás filas da táboa *empregado* correspondentes aos tres empregados do departamento número 1, e a outra corresponde á fila da táboa *departamento* correspondente ao departamento número 1.

## 1.6 Sentenza DELETE

Permite eliminar filas das táboas. Existen varias opcións de sintaxe.

Opción 1 para borrar filas dunha soa táboa:

```
DELETE [LOW_PRIORITY] FROM nome_táboa
[WHERE condición]
[ORDER BY expresión]
[LIMIT número_filas]
```

Opción 2 para borrar filas de múltiples táboas:

```
DELETE [LOW_PRIORITY] [nome_táboa[.*]] [,nome_táboa[.*]] ...
{FROM | USING} referencia_táboas
[WHERE condición]
```

Opción 3 para borrar filas de múltiples táboas:

```
DELETE [LOW_PRIORITY]
FROM nome_táboa[.*]] [,nome_táboa[.*]] ...
USING referencia_táboas
[WHERE condición]
```

- A opción LOW\_PRIORITY permite indicar o nivel de prioridade que ten a operación de borrado. Con esta opción o servidor espera que non haxa clientes lendo na táboa para facer a operación de borrado. Só afecta a táboas con motores que utilizan bloqueo a nivel de táboa (MyISAM, MEMOMRY, MERGE).
- Cando se utilizan as opcións 2 ou 3 da sintaxe, para borrar filas de varias táboas, a sentenza DELETE borra as filas das táboas nomeadas en *referencia\_táboas* que cumpren as condicións establecidas na cláusula WHERE.
- Na cláusula FROM noméanse as táboas nas que se van a borrar filas.
  - Cando se van a borrar filas de varias táboas utilizando a opción 2, establécense as relacións entre as táboas mediante a sentenza JOIN. Exemplo:



```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

- Cando se van a borrar filas de varias táboas utilizando a opción 3, na cláusula USING establécense as relacións entre as táboas mediante a sentenza JOIN. Exemplo:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

- A cláusula WHERE é opcional e especifica que filas deben borrarse.

Cando se utilizan as opcións 2 ou 3 para borrar filas en varias táboas, teñen que poñerse en primeiro lugar as condicións de enlace entre as táboas e despois o resto de condicións.

Moi importante: se non se escribe a cláusula WHERE, bórranse todas as filas da táboas nomeadas.

- A cláusula ORDER BY é opcional e indica a orde na que se borran as filas. Non se pode utilizar no borrado en múltiples táboas.
- A cláusula LIMIT establece o número de filas a borrar. Non se pode utilizar na actualización de múltiples táboas.

Exemplos de sentenzas DELETE na base de datos *practicar5*:

- Borrar as filas dunha táboa que cumpren unha condición.

Borrar os datos do empregado que ten o DNI número 12549563.

```
delete from empregado
where dni = 12549563;
```

197 21:47:24 delete from empregado where dni = 12549563 1 row(s) affected

A mensaxe informa que foi borrada unha fila.

- Borrar un número limitado de filas dunha táboa

Borrar os datos de empregado do departamento 1 que ten o salario máis baixo.

```
delete from empregado
where departamento=1
order by salario_bruto
limit 1;
```

200 21:54:52 delete from empregado where departamento=1 order by salario\_bruto limit 1 1 row(s) affected

- Borrar filas de máis dunha táboa

Borrar os departamentos que están na provincia de Madrid (*id\_provincia* 28) e todos os empregados que pertencen a el.

```
delete em, de
from empregado as em straight_join departamento as de
where em.departamento=de.codigo
and de.id_provincia=28;
```

208 22:14:25 delete em, de from empregado as em straight\_join de... 2 row(s) affected

A mensaxe informa que foron borradas dúas filas. Unha corresponde ao departamento número 10 que tiña o valor 28 na columna *id\_provincia*, e a outra corresponde ao empregado que traballaba nese departamento.



Na cláusula FROM noméanse as táboas que interveñen na operación de borrado e as relacións que hai entre elas. Utilízase a combinación STRAIGHT\_JOIN para forzar ao optimizador a que use as táboas na orde en que están escritas, desta maneira faise primeiro o borrado das filas na táboa *empregado* e despois as da táboa *departamento*. Se non se utiliza STRAIGHT\_JOIN, pode que o optimizador empece a borrar pola táboa departamento, e nese caso prodúcese un erro debido ás restricións de clave foránea xa que non está permitido borrar un departamento se hai empregados relacionados con el.

Na cláusula WHERE establécese a condición de enlace entre as táboas e o resto de condicións. Neste caso o resto das condicións son que o departamento pertenza á provincia de Madrid (o valor da columna *id\_provincia* tome o valor 28).

Pódese escribir esta mesma sentenza de borrado utilizando a opción.

```
delete
from em, de
using empregado as em straight_join departamento as de
where em.departamento=de.codigo
      and de.id_provincia=28;
```

No caso de non utilizar *straight\_join*, o optimizador pode escoller empezar a borrar pola táboa *departamento*, e nese caso produciríase un erro debido ás restricións de clave foránea. Código para facer a proba:

```
delete
from em, de
using empregado as em join departamento as de
where em.departamento=de.codigo
      and de.id_provincia=28;
```

✖ 3 12:16:50 delete from em, de using empregado as em join departamento as... Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('practicas5'...

A mensaxe de erro advirte que non se pode borrar unha fila pai da táboa *departamento* porque hai algunha fila na táboa *empregado* que fai referencia a ela, e a operación DELETE ten asociada a acción NO ACTION para esa clave foránea. O texto completo da mensaxe é o seguinte:

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails ('practicas5' : 'empregado', CONSTRAINT 'fk\_empregado\_departamento' FOREIGN KEY ('departamento') REFERENCES 'departamento' ('codigo') ON DELETE NO ACTION ON UPDATE NO ACTION)

## 1.7 Borrado lóxico de filas dunha táboa

A sentenza DELETE borra fisicamente as filas das táboas. Nas bases de datos, hai outra alternativa para 'eliminar' filas das táboas que se coñece como 'borrado lóxico' e consiste en marcar a fila poñendo nunha columna un dato que a identifica como non dispoñible. Por exemplo, pode ser unha columna que conteña unha data de baixa, ou unha columna que tome un valor lóxico (verdadeiro ou falso, 0 ou 1) que indique se a fila está eliminada ou non. Exemplos:

- Borrado físico:

```
delete from cliente
where id = 9563;
```

Esta sentenza elimina fisicamente a fila coa información do cliente que ten o identificador 9563. Se a táboa utiliza un motor transaccional, e hai filas doutras táboas que fan re-

ferencia a esa fila, aplicaranse as restricións de comportamento asociadas ás claves foráneas, restrinxindo a operación ou facendo un borrado en cascada.

- Borrado lóxico:

```
update cliente
  set data_baixa=curdate()
  where id = 9563;
```

Ou ben:

```
update cliente
  set eliminada=1
  where id = 9563;
```

Este tipo de borrado pode ser moi útil en certas ocasións, por exemplo, se o cliente volve a facer unha compra, evitamos borrar fisicamente os datos do cliente e ter que volver a inserilos cando se quere dar de alta de novo, ademais de poder conservar as referencias a ese cliente no resto de táboas.

O problema que produce este tipo de borrado é que en cada consulta que se fai na que se utilice esa táboa hai que comprobar o estado desa columna e comprobar as restricións de integridade referencial. De non facelo, pódese producir unha inconsistencia de datos, por exemplo porque pode haber filas en *ventas* que fan referencia a clientes 'eliminados'.

Tamén hai que ter ne conta que certas lexislacións, como é o caso da Lei Orgánica de Protección de datos (LOPD) en España, poden obrigar a facer borrado físico de certos tipos de datos.

Para xestionar as baixas lóxicas é recomendable utilizar disparadores (*triggers*), vistas ou procedementos almacenados.

## 1.8 Uso de subconsultas nas sentenzas de edición de datos

Para seleccionar as filas afectadas nunha operación de actualización nunha táboa é posible utilizar subconsultas na cláusula WHERE. Por exemplo, aumentar un 4% o salario a todos os empregados que traballan en departamentos da provincia de Ourense sen coñecer o código da provincia.

```
update empregado
  set salario_bruto=salario_bruto*1.04
  where departamento in (select distinct codigo
                        from departamento
                        where id_provincia=(select id_provincia
                                           from provincia
                                           where provincia='Ourense'));
```

Na primeira subconsulta (a máis interna) obtense o identificador da provincia de Ourense e na segunda obtéñense os códigos dos departamentos que pertencen a esa provincia. A sentenza UPDATE modifica as filas correspondentes aos empregados deses departamentos.

**Actualmente, MySQL non permite utilizar subconsultas nas que se fai referencia á táboa que se vai a actualizar pero existen solucións alternativas.** Por exemplo, borrar o empregado do departamento 2 que ten o salario máis baixo do departamento.

- Solución sen ter en conta a limitación de MySQL utilizando unha subconsulta para o cálculo do salario mínimo.

```
delete from empregado
where salario_bruto = (select min(salario_bruto)
                      from empregado
                      where departamento = 2)
and departamento=2;
```

✖ 219 13:14:30 delete from empregado where salario\_bruto=(select min(salario\_bruto) from empregado where departamento = 2) Error Code: 1093. You can't specify target table 'empregado' for update in FROM clause

Esta primeira solución dá lugar a un erro debido a que na subconsulta que calcula o salario máis baixo dos empregados do departamento 2 utilízase a táboa *empregado*, que é a mesma táboa na que se vai a facer a operación de borrado.

- Solución que evita a deficiencia de MySQL ordenando os resultados e limitando o número de filas

```
delete from empregado
where departamento = 2
order by salario_bruto desc
limit 1;
```

✔ 223 13:19:32 delete from empregado where departamento = 2 order by salario\_bruto desc limit 1 1 row(s) affected

Esta solución utilizando as cláusulas ORDER BY e LIMIT, funciona no caso que só se queira borrar unha fila. Se hai que borrar tódolos empregados que teñan o salario máis baixo, e hai máis dun, esta solución non funcionaría.

- Solución 3 que evita a limitación de MySQL utilizando variables para almacenar resultados intermedios.

```
/* cálculo de salario máis baixo e asignación a unha variable de calquera destas dúas
maneiras */
set @salario_minimo = (select min(salario_bruto)
                      from empregado
                      where departamento = 2);

-- ou ben:
select @salario_minimo:=min(salario_bruto)
from empregado
where departamento = 2
/* operación de borrado*/
delete from empregado
where salario_bruto = @salario_minimo
and departamento=2;
```

✔ 261 13:24:57 set @salario\_minimo = (select min(salario\_bruto) from empregado where departamento = 2) 0 row(s) affected

✔ 262 13:24:57 delete from empregado where salario\_bruto = @salario\_minimo and departamento=2 1 row(s) affected

Esta solución é a máis completa e borraría todos os empregados que teñan un salario igual ao salario máis baixo.

## 1.9 Guións de sentenzas de edición de datos nas táboas

As sentenzas de edición pódense executar de forma directa escribindo a sentenza e enviándoa ao servidor para que a execute, ou escribindo e gardando nun ficheiro de ordenes (*script*) para que se executen todas xuntas no momento que se envía o *script* ao servidor.

Un exemplo de utilización de guións de sentenzas de edición pode ser para borrar fisicamente as filas marcadas para eliminar en todas as táboas ao finalizar o ano. As sentenzas poderían ser as seguintes:

```
delete from cliente
  where eliminado = 1;
delete from artigo
  where eliminado = 1;
delete from empregado
  where eliminado = 1;
.... # aquí irían as sentenzas para borrar as fila eliminadas no resto das táboas
```

A práctica habitual para este exemplo consiste en gardar todas esas sentenzas nun ficheiro de texto coa extensión *.sql* e cando chega o final de cada ano, enviar o ficheiro ao servidor para que execute as sentenzas que contén. En Workbench, a execución faise abrindo o contido do *script* e dando a orde de executar.

Outra posibilidade para este exemplo consistiría en automatizar a tarefa anterior, creando un evento no servidor que se execute todos os anos o día 1 de xaneiro.

Outro exemplo de utilización de guións de sentenzas de edición pode ser para facer unha carga masiva de datos, xa que é posible crear un ficheiro de ordenes SQL (*script*), que conteña un grupo de instrucións INSERT para engadir filas.