

1. A01. Creación da estrutura de bases de datos relacionais

1.1 Actividade

1.1.1 Deseño físico

O deseño físico é o proceso de implantación definitiva da base de datos deseñada, sobre un SXBD concreto, utilizando directamente instrucións do xestor da base de datos, ou ben, mediante a axuda dalgunha ferramenta gráfica que facilite a creación da base de datos sen necesidade de coñecer a sintaxe da linguaxe utilizada polo xestor para ese fin. Realízase unha vez rematadas as fases de deseño conceptual e deseño lóxico.

É imprescindible empregar o manual de referencia do SXBD seleccionado para coñecer as normas de sintaxe propias do fabricante. Destácanse as seguintes precaucións:

- Nomes de bases de datos, táboas, columnas. Como regra xeral poderíamos empregar nomes que só leven letras, números e o guión baixo, tratando de evitar o uso doutros caracteres especiais incluídos o ñ, os acentos ou calquera outro símbolo. A razón desta recomendación é que aínda que o noso SXBD permita o uso de caracteres especiais pode que no futuro teñamos que migrar as nosas bases de datos a outro SXBD, ou a outro sistema operativo, que teña unhas normas máis restritivas en canto aos nomes que se poden empregar.
- Uso indistinto de maiúsculas, minúsculas, ou unha mestura de ambas, para nomear. Deberíase de empregar unha norma xeral, como por exemplo, poñer os nomes sempre en minúsculas xa que se tiveramos que migrar dun sistema Windows a outro sensible a maiúsculas e minúsculas (*case sensitive*), como pode ser Linux, pódense xerar problemas.
- Tipos de columnas. Cando se fai unha migración dunha base de datos dun SXBD a outro, é importante revisar os tipos de columnas propios dos SXBD utilizados, xa que, a pesar de existir unha norma ANSI estándar para os tipos de datos, cada fabricante introduce algunhas modificacións para mellorar o rendemento do seu motor. Así é posible que no SXBD dun determinado fabricante existan tipos de datos que non existen nos demais, ou que un mesmo tipo de dato se comporte de distinta maneira en canto a rangos de valores permitidos e espazo que ocupa no almacenamento.
- Definición das estruturas físicas de almacenamento. É importante coñecer a maneira en que o noso SXBD almacena os obxectos das bases de datos (*tablespaces*), ou os ficheiros de datos (*datafiles*).

1.1.2 Linguaxe SQL

SQL corresponde ao acrónimo de *Structured Query Language* (Linguaxe Estructurado de Consultas) e é unha ferramenta para organizar, xestionar e recuperar datos almacenados nunha base de datos relacional. Naceu como software de consulta pero actualmente aínda que esa é unha das súas funcións máis importantes, utilízase para controlar todas as funcións que subministra un SXBDR aos seus usuarios, incluíndo todas as funcións 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.

1.1.2.1 Definición de datos

A linguaxe de definición de datos ou LDD (en inglés *Data Definition Language* ou *DDL*) permite a creación do esquema da base de datos e a organización física dos datos almacenados.

1.1.2.2 Manipulación de datos

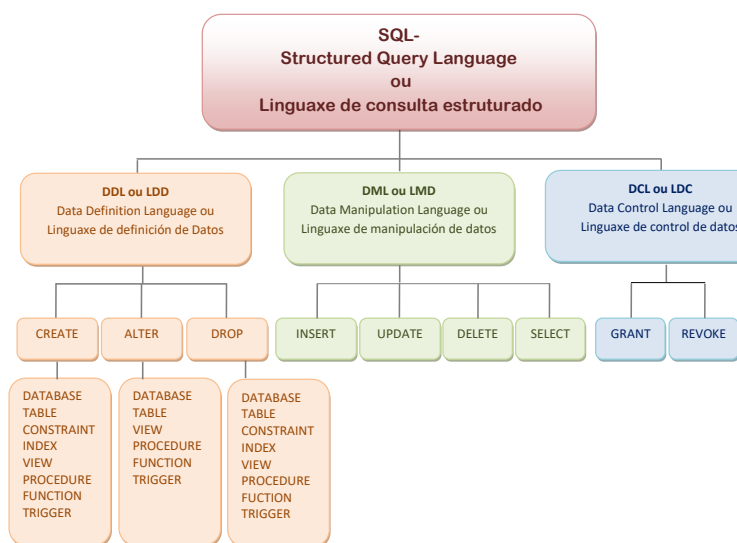
A linguaxe de manipulación de datos ou LMD (en inglés *Data Manipulation Language*, ou *DML*) permite:

- Recuperación de datos. SQL permite recuperar e utilizar os datos almacenados nunha base de datos a un usuario ou a un programa.
- Mantemento de datos. SQL permite actualizar a base de datos a un usuario ou a un programa, engadindo novos datos, borrando e modificando os que xa están almacenados.

1.1.2.3 Control de datos

A linguaxe de control de datos ou LCD (en inglés *Data Control Language* ou *DCL*) permite:

- Control do acceso. SQL pode ser utilizado para xestionar as contas dos usuarios e restrinxir a súa capacidade para recuperar, engadir e modificar datos, protexendo os datos almacenados contra accesos non autorizados.
- Control do acceso concorrente aos datos. SQL permite establecer sistemas de bloqueos de datos para permitir que varios usuarios poidan acceder ao mesmo tempo á base de datos para compartir información, evitando interferencias entre eles.
- Integridade de datos. SQL permite establecer medidas de seguridade para protexer os datos de ataques externos ou ante fallos do sistema e a recuperación da base de datos para volver a estar dispoñible para os usuarios.



1.1.2.4 Estándares SQL

Un dos elementos clave da aceptación de SQL no mercado é a existencia dun estándar oficial adoptado polo *American National Standards Institute* (ANSI) e a *International Standards Organization* (ISO). Sen embargo, hai outros dous estándares de SQL importantes que inclúen o SQL definido por DB2 de IBM e o estándar X/OPEN de SQL para UNIX.

Os inicios do estándar oficial de SQL foron en 1982, cando ANSI encargou ao seu

comité X3H2 a definición dunha linguaxe de base de datos relacional. O estándar ANSI para SQL resultante está baseado en gran medida no SQL de DB2, aínda que contén algunhas diferenzas importantes.

Resumo de revisións do estándar SQL ¹	
SQL-86	Primeira publicación feita por ANSI. Confirmada por ISO en 1987.
SQL-89, ou SQL1	Supón unha revisión menor da norma anterior
SQL-92, ou SQL2	Supón unha revisión importante, con normas máis rixidas. Mentres que o estándar orixinal de 1986 ocupaba menos de 100 páxinas, o estándar oficial SQL2, ou SQL-92, ocupou case 600.
SQL-99 ou SQL3	Incorpora a utilización de <i>triggers</i> ou disparadores, novos tipos de datos e novas características enfocadas á programación orientada a obxectos.
Revisións dos anos 2003, 2005, 2008 e 2011	Centran a maioría dos esforzos en introducir algunhas características de XML, definindo a maneira en que SQL pode ser utilizado conxuntamente co XML.

A pesar destas normas de estandarización, cada fabricante de SXBDR incorpora o seu propio dialecto SQL co fin de ofertar un produto que o diferencie dos seus competidores e incluso chega a poñerlle un nome diferente. Exemplos:

- Access SQL de Microsoft Access
- Tansact SQL de Microsoft SQL Server
- PL/SQL de Oracle
- MySQL de MySQL
- PgSQL de PostgreSQL

1.1.2.5 Formas de utilizar SQL

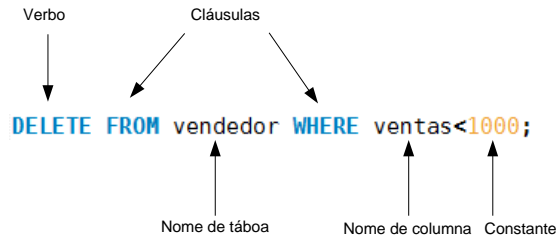
SQL é unha linguaxe declarativa de alto nivel, é dicir, non procedimental. Isto significa que as sentenzas SQL especifican o que se quere obter, pero non a forma de conseguilo. A orde interna na que se executan as operacións asociadas a unha sentenza SQL é establecida polo optimizador de consultas do SXBD. As sentenzas SQL poden utilizarse:

- Escribindo directamente as sentenzas coa axuda dun cliente en modo texto, en modo gráfico, ou na pantalla dun terminal interactivo; o servidor devolve o resultado da petición.
- Gardando un conxunto de sentenzas nun ficheiro de ordes (*scripts*), que se executan todas seguidas de forma secuencial.
- Escribindo as sentenzas de SQL incluídas (embebidas) en programas escritos con distintas linguaxes de programación como PHP, Java ou C#.

1.1.2.6 Forma básica das sentenzas SQL

A linguaxe SQL consta dun conxunto de sentenzas. Cada sentenza indica unha acción específica a realizar por parte do SXBD, tal como a creación dunha nova táboa, a consulta de datos dunha ou máis táboas, ou a inserción de novos datos na base de datos.

¹ As copias impresas dos estándares SQL teñen restricións de copyright. Pódense atopar á venda na páxina web de [ANSI](#), seleccionando 'Access Standards', e en 'Standards Store' buscar (*search*): "SQL Language."



Todas as sentenzas SQL teñen a mesma estrutura:

- Empezan cun verbo, que é unha palabra clave que indica a acción que ten que executar o SXBD. Algúns destes verbos son: CREATE, INSERT, DELETE, ou SELECT.
- A sentenza continúa cunha ou máis cláusulas. Unha cláusula pode especificar os datos sobre os que debe actuar a sentenza, ou proporcionar máis detalles acerca da forma en que se ten que executar. Todas as cláusulas empezan tamén cunha palabra clave, tal como WHERE, FROM, INTO e HAVING e van seguidas de expresións, nomes de táboas ou nomes de columnas. Algunhas cláusulas teñen que aparecer de forma obrigatoria na sentenza, pero outras son opcionais.

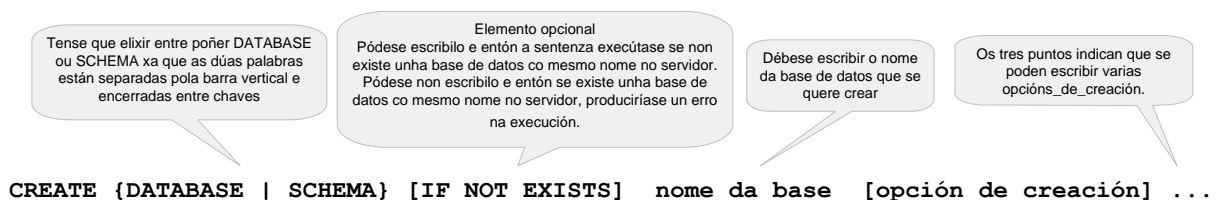
O estándar SQL ANSI/ISO especifica as palabras clave da linguaxe SQL que se utilizan como verbos ou como cláusulas das sentenzas. Ademais, establece que estas palabras clave non poden ser empregadas para designar obxectos da base de datos, tales como táboas, vistas, columnas ou usuarios. Moitos dialectos de SQL suavizan esta restrición, pero é recomendable evitar o uso de palabras clave ao nomear táboas, columnas, ou calquera outro obxecto da base de datos.

1.1.2.7 Notación para a sintaxe

SQL, do mesmo xeito que outras linguaxes, está formada por un conxunto de palabras e un conxunto de normas de sintaxe para a construción das sentenzas. A notación empregada nos manuais para a explicación da sintaxe das sentenzas resúmese na seguinte táboa.

Resumo de notacións utilizadas na sintaxe das sentenzas SQL	
MAIÚSCULAS	O texto escrito en maiúsculas representa palabras reservadas, que non poden ser utilizadas para outros fins. Á hora de escribir a sentenza non se diferencia entre minúsculas e maiúsculas
Minúsculas	Representan partes da instrución nas que temos que substituír a palabra ou frase escrita en minúsculas polo que representa
[texto]	A parte da sentenza encerrada entre corchetes é optativa
{ opción1 opción2 }	Representa unha alternativa. Ao escribir a instrución tense que elixir unha das opcións encerradas entre as chaves, que van separadas por barras verticais ()
...	Tres puntos seguidos indican que a parte da sentenza que está inmediatamente antes deles, pódese repetir varias veces

No seguinte exemplo de sintaxe, as palabras CREATE, DATABASE, SCHEMA, IF, NOT, e EXISTS son palabras reservadas que non se poden utilizar noutro lugar da sentenza diferente ao que aparecen na norma de sintaxe.



A escritura das sentenzas SQL pode facerse en maiúsculas, minúsculas ou nunha combinación de ambas.

1.1.2.8 Comentarios en SQL

Os comentarios poden formar parte dunha sentenza SQL e non serán executados polo servidor. Poden estar delimitados ou non e ocupar unha liña ou máis dunha. Os símbolos utilizados para definir comentarios nun *script* de SQL ou nunha sentenza, son os que se mostran na seguinte táboa.

Símbolos para definir comentarios	
--	Dous guións ao comezo dunha liña indican que esa liña sexa interpretada como un comentario
#	O símbolo # indica que o texto que hai dende el ata o final da liña sexa interpretado como un comentario
/* texto */	Define o texto como comentario. O texto pode ocupar varias liñas

Exemplo de código con comentarios:

```
-- Script de creación da estrutura da base de datos horas_extra
create schema if not exists /* só no caso de non existir */ horas_extra;
/* Forma de comentario que ocupa máis dunha liña,
ou cando se quere inserir o comentario no medio dunha sentenza */
use empresa; # Selección da base de datos empresa
```

Así se verían os comentarios no editor de MySQL Workbench:

```
1 -- Script de creación da estrutura da base de datos horas_extra
2 • create schema if not exists /* só no caso de non existir */ horas_extra;
3  /* Forma de comentario que ocupa mais dunha liña,
4  ou cando se quere inserir o comentario no medio dunha sentencia */
5 • use empresa; # Selección da base de datos empresa
```

MySQL permite utilizar os símbolos /* e */ propios dos comentarios para definir sentenzas executables. Esta utilización debe facerse con precaución para non ter problemas ao executar arquivos de scripts de MySQL noutros SXBDR.

Utilización especial dos comentarios en MySQL	
/*! código_MySQL */	En MySQL, interprétase o código como unha sentenza executable e execútase; no resto de servidores, interprétase como comentario
/*!version código_MySQL */	En MySQL, interprétase o texto como unha sentenza executable e execútase nun servidor cunha versión igual ou superior á versión especificada; no resto de servidores, interprétase como comentario

Exemplo de código e vista do mesmo editado en MySQLWorkbench:

```
/*! create database empresa */;
/*!50021 use empresa */;
create /*!32302 temporary */ table proba (columna integer);
```

```
1 • /*! create database empresa */;
2 • /*!50021 use empresa */;
3 • create /*!32302 temporary */ table proba (columna integer);
```

1.1.3 Linguaxe de definición de datos

A linguaxe de definición de datos, LDD ou DDL, está formada polo conxunto de instrucións do SQL que permiten ao administrador crear o esquema da base de datos e facer os cambios necesarios, no seu esquema, unha vez que está creada. Permite crear, modificar e suprimir bases de datos, táboas e índices. O núcleo da LDD está baseado en tres verbos de SQL:

- CREATE, que define e crea un obxecto da base de datos.
- DROP, que elimina un obxecto existente na base de datos.

- ALTER, que modifica a definición dun obxecto existente na base de datos.

1.1.3.1 Consulta do dicionario de datos dun servidor MySQL

O dicionario de datos ou catálogo, garda información sobre os obxectos almacenados no servidor. O que se coñece como metadatos (datos sobre os datos).

Cando se instala MySQL créanse automaticamente catro bases de datos: *mysql*, *information_schema*, *performance_schema* e *test*. As dúas primeiras gardan información dos metadatos.

O estándar ANSI/ISO SQL:2003 no apartado referido ao esquema (*Part 11 'Schemata'*) define a estrutura das táboas da base de datos *information_schema* que facilita a consulta dos metadatos. Son táboas de só lectura.

Para consultar información sobre os metadatos, MySQL dispón dun conxunto propio de sentenzas identificadas co nome SHOW, e ademais, a partir da versión 5.0.2 está dispoñible a base de datos *information_schema* que lle permite adaptarse á norma ANSI SQL.

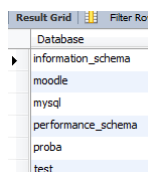
Todos os usuarios de MySQL teñen permiso para acceder ás táboas da base de datos *information_schema*, pero só poderán ver aquelas filas correspondentes a obxectos sobre os que teñan permisos.

Exemplo de consulta dos metadatos nun servidor de MySQL. Para ver os nomes das bases de datos creadas nun servidor pódense utilizar dúas opcións:

- Executar a sentenza SHOW DATABASES que só funciona en MySQL.

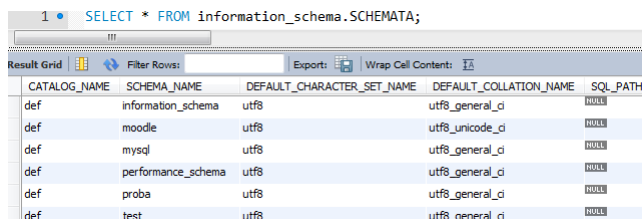
`show databases;`

O resultado de executar a sentenza será unha lista cos nomes das bases de datos creadas no servidor.



Database
information_schema
moodle
mysql
performance_schema
proba
test

- Consultar a táboa SCHEMATA da base de datos *information_schema*.



CATALOG_NAME	SCHEMA_NAME	DEFAULT_CHARACTER_SET_NAME	DEFAULT_COLLATION_NAME	SQL_PATH
def	information_schema	utf8	utf8_general_ci	NULL
def	moodle	utf8	utf8_unicode_ci	NULL
def	mysql	utf8	utf8_general_ci	NULL
def	performance_schema	utf8	utf8_general_ci	NULL
def	proba	utf8	utf8_general_ci	NULL
def	test	utf8	utf8_general_ci	NULL

Cando se traballa utilizando clientes gráficos como MySQL Workbench ou PHPMyAdmin, ao establecer a conexión, móstranse os nomes das bases de datos. No caso de MySQL Workbench, as táboas e outros obxectos relacionados coa base de datos, móstranse na zona de navegación.

1.1.3.2 Nomes dos obxectos

Tódolos manuais de referencia dos SXBDR teñen especificadas as normas para nomear os obxectos do esquema das bases de datos: base de datos, táboas, índices, columnas, desencadeadores, ou procedementos almacenados. Resumo das normas para nomear obxectos en MySQL:

- Lonxitude máxima de caracteres para o nome: 64.

- Carácteres permitidos: calquera letra do alfabeto inglés (sen ñ nin acentos) tanto en minúscula como en maiúsculas, números e os carácteres especiais `_` e `$`. Pódese empregar, aínda que non se recomenda, o acento grave (```) para delimitar nomes de obxectos con carácteres non permitidos.
- Dentro dunha mesma base de datos non pode haber nomes de táboas repetidas. Para facer referencia a unha táboa dunha determinada base de datos, pódese utilizar a expresión: `nome_base_de_datos.nome_táboa`.
- Dentro dunha mesma táboa non pode haber nomes de columnas repetidas. Para facer referencia a unha columna dunha determinada táboa, pódese utilizar a expresión: `nome_táboa.nome_columna`.

1.1.3.3 Base de datos

Creación dunha base de datos

A sintaxe da sentenza para a creación da base de datos é:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nome_da_base [opcións_de_creación] ...
```

- `DATABASE` e `SCHEMA` son sinónimos. Pódese utilizar calquera das dúas opcións.
- Cando se utiliza a parte optativa `IF NOT EXISTS`, créase a base de datos no caso de que non exista; se existe, móstrase unha mensaxe de advertencia (*warning*). Se non se pon esta opción, prodúcese un erro no caso de que a base de datos exista.
- O `nome_da_base` debe cumprir as normas vistas para nomear obxectos.
- As posibles opcións de creación son:

```
[DEFAULT] CHARACTER SET [=] nome_xogo_carácteres
```

```
[DEFAULT] COLLATE [=] nome_sistema_ordenación
```

- A cláusula `CHARACTER SET` establece o conxunto de carácteres por defecto para as táboas que se crean nesa base de datos; no caso de non especificalo, tomarase o que ten definido por defecto na configuración do servidor.

Para ver os xogos de carácteres permitidos en MySQL, pódese executar a sentenza:

```
show character set;
```

Ao executar esta sentenza, móstrase unha táboa que contén catro columnas: o nome do xogo de carácteres, a descrición, o sistema de colación por defecto e a lonxitude máxima en bytes por cada carácter. Exemplo de captura dos resultados mostrados:

Charset	Description	Default collation	Maxlen
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
tis620	TIS620 Thai	tis620_thai_ci	1
latin5	ISO 8859-9 Turkish	latin5_turkish_ci	1
utf8	UTF-8 Unicode	utf8_general_ci	3
latin7	ISO 8859-13 Baltic	latin7_general_ci	1
utf8mb4	UTF-8 Unicode	utf8mb4_general_ci	4
utf16	UTF-16 Unicode	utf16_general_ci	4

Os xogos de carácteres nacionais requiren un byte para cada carácter. O xogo de carácteres Unicode (`utf8`, `utf16` e `utf32`), que permiten gardar texto en calquera alfabeto ou conxunto de carácteres do mundo, poden necesitar máis espazo. Por exemplo, en MySQL o xogo de carácteres `utf8` pode ser: `utf8` e `utf8mb4`; o primeiro utiliza dende un ata tres bytes por carácter e o segundo utiliza dende un ata catro bytes por carácter.

- A cláusula `COLLATE` establece o conxunto de regras para comparar carácteres. Para ver os sistemas de colación permitidos en MySQL, pódese executar a sentenza:

```
show collation;
```


Collation	Charset	Id	Default	Compiled	Sortlen
latin1_general_ci	latin1	48		Yes	1
latin1_general_cs	latin1	49		Yes	1
latin1_german1_ci	latin1	5		Yes	1
latin1_german2_ci	latin1	31		Yes	2
latin1_spanish_ci	latin1	94		Yes	1
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin2_bin	latin2	77		Yes	1
latin2_croatian_ci	latin2	27		Yes	1

No caso de non especificar ningún sistema de colación de maneira explícita, o sistema colle o que corresponde por defecto ao xogo de caracteres elixido.

Exemplo de creación dunha base de datos:

```
create database if not exists proba
default character set utf8
default collate utf8_spanish_ci;
```

Esta sentenza crea a base de datos chamada *proba*, co xogo de caracteres por defecto *utf8* e o sistema de colación (utilizado para comparacións) *utf8_spanish_ci*. Os obxectos que se crean nesa base de datos utilizarán ese xogo de caracteres e ese sistema de comparación, se non se especifica ningún outro no momento da creación do obxecto.

Poñer en uso unha base de datos

A sintaxe da sentenza para seleccionar unha base de datos e poder traballar con ela é:

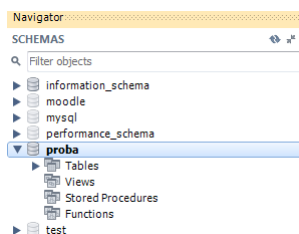
```
USE nome_da_base
```

Todas as sentenzas que se executen despois da sentenza `USE` afectan só a esa base de datos. Exemplo:

```
use proba;
```

Tamén se poden cualificar os nomes dos obxectos afectados por unha sentenza indicando de maneira explícita o nome da base de datos á que vai a afectar a sentenza, engadíndolle o nome da base de datos diante do nome do obxecto. Exemplo: *practicass1.grupo* fai referencia o obxecto *grupo* que pertence a base de datos *practicass1*.

Cando se traballa utilizando MySQL Workbench como cliente gráfico, pódese poñer en uso unha base de datos pinchando dúas veces sobre o nome da base de datos na zona de navegación. O nome da base de datos en uso aparece destacado en letra negra.



Borrar unha base de datos

A sintaxe da sentenza para borrar unha base de datos é:

```
DROP DATABASE [IF EXISTS] nome_da_base
```

Cando se borra unha base de datos, bórranse todos os obxectos que contén a base de datos e non se poden recuperar. Exemplo:

```
drop database if exists proba;
```


1.1.3.4 Táboa

Creación dunha táboa

Para crear unha táboa, hai que darlle un nome, definir as columnas que vai a ter e elixir as opcións da táboa e particionamento. Sintaxe resumida para MySQL:

```
CREATE [TEMPORARY] TABLE nome_táboa
(
  nome_columna  tipo_de_dato  [propiedades_da_columna]  [, ...]
)
[opcións_de_táboa] [opcións_de_partición]
```

- **TEMPORARY** é opcional, e permite crear una táboa temporal mentres dura a sesión.
- A definición de cada columna consiste en darlle un nome, asignarlle un tipo de dato e asociarlle as restricións necesarias. As dúas primeiras partes (nome e tipo) son obrigatorias.

Os SXBDR soportan un gran número de tipos de datos divididos en varias categorías: numéricos, data e hora, e cadeas de caracteres. Ver o apartado "Tipos de datos en MySQL" que se explica máis adiante.

Para definir as propiedades da columna, pódense utilizar as cláusulas:

- **NOT NULL**, non permite que a columna tome o valor nulo (descoñecido).
 - **NULL**, permite que a columna tome o valor nulo.
 - **UNSIGNED**, permite almacenar só valores positivos en columnas de tipo numérico.
 - **ZEROFILL**, enche con ceros pola esquerda as columnas de tipo numérico.
 - **BINARY**, diferenza entre maiúsculas e minúsculas en columnas de tipo cadea de caracteres.
 - **DEFAULT *valor***, indica o valor por defecto para a columna, no caso de non asignarlle ningún valor cando se engade unha fila á táboa. Está permitido utilizar algúns valores proporcionados por funcións do sistema no lugar de valores constantes definidos polo usuario. Por exemplo, **USER** para referirse ao usuario que estableceu a conexión, ou **CURRENT_TIMESTAMP** para referirse á data e hora actual.
 - **AUTO_INCREMENT**, indica que o valor que toma a columna é calculado polo sistema caso de non asignarlle ningún valor á columna ou en caso de asignarlle o valor nulo. O sistema calcula o valor sumándolle 1 ao valor que toma a columna para a última fila engadida.
 - **PRIMARY KEY**, define a columna como clave primaria. Tamén se pode definir como clave primaria creando unha restrición.
- Algunhas opcións de táboa son:

```
[DATA DIRECTORY= 'directorio']
[INDEX DIRECTORY= 'directorio']
[{ENGINE | TYPE} = {ISAM, MyISAM, Innodb, ...}]
[[DEFAULT] CHARACTER SET nome_xogo_caracteres]
[COLLATE nome_sistema_colación]]
[AUTO_INCREMENT = número],
[COMMENT texto]
```

 - **DATA DIRECTORY** e **INDEX DIRECTORY**, especifican as rutas absolutas nas que se almacenan os datos e os índices.
 - **ENGINE**, indica o motor de almacenamento asociado á táboa (**TYPE** está en desuso nas últimas versións). Ver o apartado "Motores de almacenamento en MySQL" que se explica máis adiante.
 - **CHARACTER SET** e **COLLATE**, permiten establecer o conxunto de caracteres por defecto para as columnas que se crean nesa táboa, e a forma en que se ordenan e comparan os caracteres.

- AUTO_INCREMENT, indica o número de comezo para a columna de tipo autoincremental.
- As opcións de particionamento explicaranse no apartado "Táboas particionadas".

Exemplo de código para crear unha táboa sinxela na base de datos *practicasi*:

```
create table test.fotografia      # nome de táboa cualificado
( id          integer unsigned not null auto_increment,
  titulo      varchar(60) not null default 'sen título',
  autor       varchar(60) not null comment 'Apelidos e nome do autor ou autores',
  data        date null comment 'Data en que foi tomada',
  primary key (id)
) engine = InnoDB;
```

Tipos de datos en MySQL

A elección do tipo de columna faise en función dos datos que se queren almacenar nela. MySQL soporta unha gran cantidade de tipos de datos; a maioría cumpren o estándar ANSI e outros son propios de MySQL. A seguinte táboa contén un resumo dos tipos de datos extraído do manual de referencia de MySQL.

Resumo dos tipos de datos para MySQL, e atributos que poden ter asociadas		
Numéricos	Enteiros	TINYINT[(tamaño)] [UNSIGNED] [ZEROFILL] SMALLINT[(tamaño)] [UNSIGNED] [ZEROFILL] MEDIUMINT[(tamaño)] [UNSIGNED] [ZEROFILL] INT[(tamaño)] [UNSIGNED] [ZEROFILL] INTEGER[(tamaño)] [UNSIGNED] [ZEROFILL] BIGINT[(tamaño)] [UNSIGNED] [ZEROFILL]
	Non enteiros	REAL[(tamaño,decimais)] [UNSIGNED] [ZEROFILL] DOUBLE[(tamaño,decimais)] [UNSIGNED] [ZEROFILL] FLOAT[(tamaño,decimais)] [UNSIGNED] [ZEROFILL] DECIMAL(tamaño,decimais) [UNSIGNED] [ZEROFILL] NUMERIC(tamaño,decimais) [UNSIGNED] [ZEROFILL] BIT[(tamaño)]
Data e hora		DATE TIME TIMESTAMP DATETIME
Cadeas de caracteres		CHAR(tamaño) [BINARY ASCII UNICODE] VARCHAR(tamaño) [BINARY] BINARY(tamaño) VARBINARY(tamaño) TINYBLOB BLOB LONGBLOB TINYTEXT [BINARY] TEXT [BINARY] MEDIUMTEXT [BINARY] LONGTEXT [BINARY]
Outros tipos		ENUM(valor1,valor2, valor3,...) SET(valor1,valor2,valor3,...)

As seguintes táboas conteñen resumos extraídos do manual de referencia de MySQL sobre o almacenamento requirido para algúns tipos de datos. No caso dos datos de tipo enteiro, móstrase o rango de valores posibles tanto para os enteiros con signo coma sen signo.

Resumo de tipos enteiros para MySQL			
Tipo	Bytes	Valor mínimo con signo e sen signo	Valor máximo con signo e sen signo
TINYINT	1	-128 0	127 255
SMALLINT	2	-32768 0	32767 65535
MEDIUMINT	3	-8388608	8388607

		0	16777215
INT	4	-2147483648 0	2147483647 4294967295
BIGINT	8	-9223372036854775808 0	9223372036854775807 18446744073709551615
Resumo doutros tipos numéricos non enteiros para MySQL			
Tipo	Almacenamento requirido		
FLOAT(p)	4 bytes se $0 \leq p \leq 24$, 8 bytes se $25 \leq p \leq 53$		
FLOAT	4 bytes		
DOUBLE [PRECISION], REAL	8 bytes		
DECIMAL(M,D), NUMERIC(M,D)	Dende MySQL 5.0.3, os valores para columnas DECIMAL máis longos represéntanse usando un formato binario que empaqueta nove díxitos decimais en catro bytes. O almacenamento para a parte enteira e decimal determináanse separadamente. Cada múltiplo de nove díxitos require catro bytes, e o dígito "de resto" require algunha fracción de catro bytes		
BIT(M)	Aproximadamente $(M + 7)/8$ bytes Para especificar valores tipo bit, para os valores pódese usar a notación b'value'. Exemplo: b'1101'		
Resumo de tipos data e hora para MySQL			
Tipo	Almacenamento requirido		
	Ata a versión MySQL 5.6.4	Dende a versión MySQL 5.6.4	
DATE	3 bytes	3 bytes	
DATETIME	8 bytes	5 bytes + parte fraccionaria (1 byte cada dous díxitos de precisión)	
TIMESTAMP	4 bytes	4 bytes + parte fraccionaria (1 byte cada dous díxitos de precisión)	
TIME	3 bytes	3 bytes + parte fraccionaria (1 byte cada dous díxitos de precisión)	
YEAR	1 byte	1 byte	
Resumo de tipos cadea de caracteres para MySQL			
Tipo	Almacenamento requirido		
CHAR(M), BINARY(M)	M bytes, $0 \leq M \leq 255$		
VARCHAR(M), VARBINARY(M)	L+1 bytes, se M está entre 0 e 255, e L+2 se M é maior que 255 L representa o número de caracteres da cadea que garda. A lonxitude máxima é de 65.532 bytes		
TINYBLOB, TINYTEXT	L +1 byte, onde $L < 2^8$		
BLOB, TEXT	L +2 bytes, onde $L < 2^{16}$		
MEDIUMBLOB, MEDIUMTEXT	L+ 3 bytes, onde $L < 2^{24}$		
LOBLOB, LONGTEXT	L +4 bytes, onde $L < 2^{32}$		
ENUM('value1','value2',...)	1 o 2 bytes, dependendo do número de valores da enumeración (65.535 valores como máximo)		
SET('value1','value2',...)	1, 2, 3, 4, o 8 bytes, dependendo do número de membros do conxunto (64 membros como máximo)		

Algunhas consideracións sobre estes tipos:

- MySQL soporta os tipos do estándar SQL para valores de tipo enteiro, INTEGER (ou INT) e SMALLINT. Como unha extensión ao estándar, MySQL engade os tipos TINYINT, MEDIUMINT e BIGINT.

MySQL soporta unha extensión que permite especificar o ancho en pantalla dos datos de tipo enteiro, poñendo entre parénteses un número que representa o ancho de pantalla para a columna. O ancho de pantalla non limita o rango de valores que se poden almacenar na columna. Cando se utiliza o ancho de pantalla en combinación coa opción ZEROFILL, o recheo con ceros pola esquerda para cantidades con menos díxitos que os especificados como ancho de pantalla, faise ata o tamaño do ancho de pantalla. Exemplo:

```
fillos tinyint(2) unsigned zerofill
```

A columna *fillos* almacena valores numéricos enteiros que poden tomar valores entre 0 e 255, pero o 2 indica o ancho de pantalla que pode ser utilizado polas aplicacións que manexan o dato. Se toma o valor 3, móstrase como 03.

- Para poder almacenar valores numéricos con cifras decimais exactas, tamén coñecidos de coma fixa, MySQL dispón dos tipos de datos que son sinónimos, DECIMAL (ou DEC), e NUMERIC. Sintaxe:

DECIMAL (m, d)

- *m* é o número máximo de cifras en total (a escala). O rango permitido é de 1 a 65.
- *d* é o número de cifras despois de la coma (a precisión). O rango permitido é de 0 a 30 e non pode tomar un valor maior que *m*.

Se non se especifica o tamaño, o valor por defecto para *m* é 10, e para *d* é 0. Exemplo:

salario **decimal(7,2)**

A columna *salario* almacena valores numéricos que poden ter un máximo de 7 díxitos, e o 2 indica que poden ter dous decimais. O rango de valores permitido é de -99999.99 a 99999.99.

- Para almacenar valores decimais aproximados, tamén coñecidos como de coma flotante, MySQL dispón dos tipos FLOAT (para simple precisión) e DOUBLE (para dobre precisión).
- Dende MySQL 5.0.3, o tipo de datos BIT pode usarse para gardar valores dun bit. Un tipo BIT(M) permite o almacenamento de valores de M-bit. M ten un rango de 1 a 64. Para especificar valores bit, pódese usar a notación b'*value*'. A palabra *value* representa un valor binario escrito usando ceros e uns. Exemplo, b'111' e b'100000000' representan 7 e 128, respectivamente.

Cando se asigna un valor a unha columna BIT(M) con menos de M bits, o valor complétase pola esquerda con ceros. Exemplo, ao asignar un valor b'101' a unha columna BIT(6) é o mesmo que asignar b'000101'.

En versións anteriores, o tipo BIT era equivalente a un TINYINT.

- Para poder almacenar valores que representen datas e horas, MySQL dispón dos tipos DATE, TIME, DATETIME e TIMESTAMP.
 - O tipo DATE utilízase para almacenar datas co formato 'AAAA-MM-DD'. O rango permitido é '1000-01-01' a '9999-12-31'. Admite utilizar guión (-) ou barra como separador (/), ou ben, omitir o separador.
 - O tipo TIME utilízase para representar horas. Ten o formato HH:MM:SS[.fracción].
 - Os tipos DATETIME e TIMESTAMP utilízanse para valores que conteñen data e hora co formato 'AAAA-MM-DD HH:MM:SS[.fracción]'. O rango permitido para o tipo DATETIME é '1000-01-01 00:00:00' a '9999-12-31 23:59:59', e para o tipo TIMESTAMP é '1970-01-01 00:00:01' a '2038-01-19 03:14:07'. Admítese utilizar guión (-) ou barra como separador (/) para a data.
 - A partir da versión MySQL 5.6.4, os tipos TIME, DATETIME e TIMESTAMP teñen a posibilidade de gardar fraccións de segundo ata o microsegundo (6 díxitos) como máxima precisión. Hai que indicar o número de decimais entre parénteses a continuación do nome do tipo. Exemplo:

nome_columna **timestamp(2)**

O 2 que vai entre parénteses indica que ten unha precisión de décimas de segundo.

- O tipo YEAR ocupa 1 byte e é usado para representar valores de anos. Pode ser declarado como YEAR(4) ou como YEAR(2).
- Os tipos CHAR e VARCHAR permiten almacenar cadeas de caracteres e diferéncianse no tipo de lonxitude (o primeiro ten lonxitude fixa e o segundo ten lonxitude variable) e na lonxitude máxima para as columnas (para o tipo CHAR é 255 caracteres e para o

tipo VARCHAR é de 65532). Este tipo de columnas non diferencian entre minúsculas e maiúsculas, a non ser que se lles asocie a opción BINARY.

Para o tipo VARCHAR, os requirimentos de almacenamento dependen da lonxitude da cadea almacenada e o valor que vai entre parénteses representa a lonxitude máxima que pode ter a cadea, en caracteres. Exemplo:

Tipo de columna	Valor asignado	Bytes que ocupa
CHAR(30)	'pepe'	30
VARCHAR(30)	'pepe'	5

O tipo VARCHAR aproveita mellor o espazo de almacenamento, pero ten como contrapartida que require máis tempo de proceso do servidor.

- Os tipos BINARY e VARBINARY son similares a CHAR e VARCHAR, pero almacénanse como cadeas de caracteres binarias. Conteñen cadeas de bytes, en lugar de cadeas de caracteres, polo que diferencian as minúsculas das maiúsculas.
- Os tipos BLOB e TEXT son de lonxitude variable e permiten tratar unha gran cantidade de datos variables. As columnas tipo BLOB trátanse como cadeas de caracteres binarias (de byte). As columnas tipo TEXT trátanse como cadeas de caracteres non binarias (de caracteres). Cada valor BLOB ou TEXT represéntase internamente como un obxecto a parte.

As columnas tipo BLOB non teñen conxunto de caracteres e ordénanse e compáranse baseándose nos valores numéricos dos bytes.

Hai catro tipos BLOB e catro tipos TEXT que se diferencian entre eles só na lonxitude máxima dos valores que poden tratar.

- Tipos BLOB: TINYBLOB, BLOB, MEDIUMBLOB e LONGBLOB.
- Tipos TEXT: TINYTEXT, TEXT, MEDIUMTEXT e LONGTEXT.

As columnas tipo BLOB son obxectos binarios que poden almacenar calquera tipo de información, dende un arquivo de texto, una imaxe, arquivos de son ou vídeo.

- Os tipos ENUM e SET representan unha cadea de caracteres á que se engade unha restrición de valor, indicando entre parénteses a lista de valores permitidos separados por comas. No caso de ENUM, só pode almacenar un deses valores; no caso de SET, pode almacenar máis dun deses valores. Exemplos:

```
talla enum('superpequena','pequena','mediana','grande','supergrande')
aficions set('deporte','fotografía','lectura','maquetismo')
```

- A columna *talla* pode almacenar unicamente algún dos cinco valores representados na lista, ou ben o valor nulo. Pode estar asociada a un campo tipo *radio* dun formulario HTML.
- A columna *aficions* pode almacenar un ou máis dos valores da lista. Pode estar asociada a un campo tipo *checkbox* dun formulario HTML.

Estes tipos de columnas funcionan dunha maneira similar aos *strings* de cadeas de caracteres. Internamente almacénanse como valores enteiros que representan a posición do valor dentro da lista, o que supón un importante aforro de espazo de almacenamento. Exemplo: Se a táboa na que está a columna *talla* tivese un millón de filas que toman o valor 'mediana' nesa columna, requírense 1 millón de bytes de almacenamento, en lugar dos 8 millóns que se utilizarían para almacenar o valor 'mediana' se a columna fose definida como tipo VARCHAR.

A elección do tipo de columna é unha tarefa delicada que require prestarlle atención antes de empezar a crear as táboas. Hai que ter en conta dous factores que poden influír na

decisión: o espazo de almacenamento utilizado polo tipo de columna e o tempo de proceso que require o servidor para manexar ese tipo de columna.

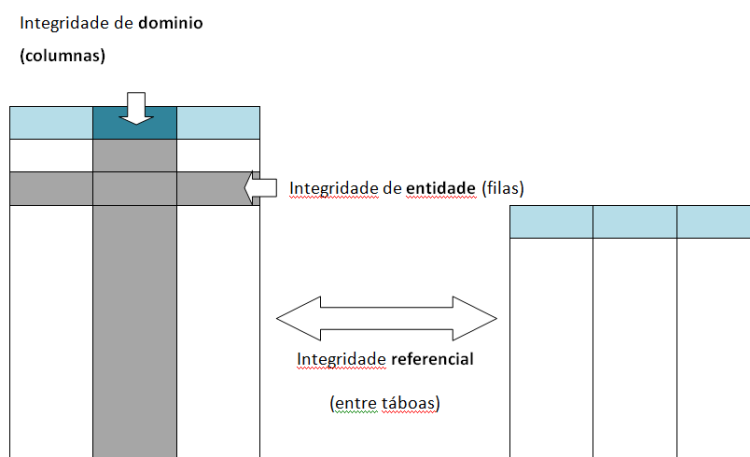
O manual de Mysql contén recomendacións para optimizar a estrutura dunha base de datos, facendo referencia especial ao tamaño dos datos e aos tipos de columnas².

Implementación da integridade mediante restricións

Un paso importante no deseño dunha base de datos é decidir a mellor forma de implementar a integridade dos datos. A integridade fai referencia á coherencia e a precisión dos datos que están almacenados nunha base de datos. Tipos de integridade:

- **Integridade de dominio (ou columna).** Especifica o conxunto de valores de datos que son válidos para unha columna e se admite valores nulos. A integridade de dominio adóitase implementar mediante o uso de comprobacións de validez e tamén mediante a restrición do tipo de datos, o formato ou o intervalo dos valores posibles permitidos nunha columna.
- **A integridade de entidade (ou táboa).** Require que todas as filas dunha táboa teñan un identificador exclusivo, coñecido como clave primaria. A modificación do valor da clave primaria ou a eliminación da fila enteira depende do nivel de integridade requirido entre a clave primaria e calquera outra táboa.
- **Integridade referencial.** Esta integridade asegura que sempre se manteñen as relacións entre as claves principais (na táboa á que se fai referencia) e as claves externas (nas táboas que fan referencia). Non se pode eliminar unha fila, nin se pode modificar a clave primaria, se unha clave externa fai referencia á fila, salvo que se permita a acción 'en cascada'. Pódense definir relacións de integridade referencial entre táboas diferentes ou entre columnas da mesma táboa.

No seguinte gráfico ilústranse os tres tipos de integridade:



Métodos para implementar a integridade

A integridade dos datos pódese esixir mediante dous métodos: integridade de datos declarativa ou integridade de datos procedimental.

- Coa integridade declarativa, defínense os criterios que teñen que cumprir os datos como parte da definición dun obxecto; despois, o xestor asegura automaticamente que os

² Sección 8.4.2 *Optimizing MySQL Data Types* do manual de referencia MySQL 5.6.

datos cumplan esos criterios. O método preferido para implementar a integridade de datos básica é a integridade declarativa. Hai que ter en conta os feitos seguintes sobre o método declarativo:

- Declárase como parte da definición da base de datos, mediante o uso de restricións declarativas que se definen directamente nas táboas e nas columnas.
 - Impleméntase mediante a utilización de restricións, valores predeterminados e regras.
- Coa integridade procedimental, escríbense secuencias de comandos que definen os criterios que teñen que cumprir os datos e comprobamos que os devanditos criterios se cumpren. Débese limitar o uso da integridade procedimental a situacións excepcionais e a aquelas cunha lóxica complicada. Por exemplo, utilízase a integridade procedimental cando se desexa implementar un tipo de eliminación do que non dispón o xestor. Os feitos seguintes aplícanse á integridade procedimental:
- Pódese implementar no cliente ou no servidor mediante outras linguaxes e ferramentas de programación.
 - Impleméntase utilizando disparadores (*triggers*) e procedementos almacenados.

Definición de restricións

As restricións son o método máis adecuado para conseguir a integridade dos datos e son un método estándar ANSI para implementar a integridade dos datos. Cada tipo de integridade de datos (dominio, entidade e referencial) impleméntase con tipos de restricións diferentes. As restricións aseguran que os datos que se escriben nas columnas sexan válidos e que se manteñan as relacións entre as táboas. A táboa seguinte describe os diferentes tipos de restricións.

Tipos de integridade	Tipo de restrición	Descrición
Dominio	DEFAULT	Especifica o valor que se proporciona para a columna cando non se especifica explicitamente cando se insire unha nova fila.
	CHECK	Especifica mediante unha expresión os valores dos datos que se aceptan nunha columna.
	REFERENTIAL	Especifica os valores de datos que se aceptan como actualización en función dos valores dunha columna doutra táboa
Entidade	PRIMARY KEY	Identifica de forma exclusiva cada unha das filas; asegura que os usuarios non escriban valores duplicados e que se cre un índice para aumentar o rendemento. Non se permiten valores nulos.
	UNIQUE	Impide a duplicación de claves alternativas (non principais) e asegura que se cree un índice para aumentar o rendemento.
Referencial	FOREIGN KEY	Define unha columna ou combinación de columnas nas que os seus valores coinciden coa clave primaria da mesma ou outra táboa.
	CHECK	Especifica os valores dos datos que se aceptan nunha columna en función dos valores doutras columnas da mesma táboa.

Creación de táboas e restricións

Dende a norma ANSI SQL2, na sentenza CREATE TABLE, ademais da definición de columnas, pódense introducir algunhas cláusulas que nos permiten completar a descrición da táboa, introducindo algunhas restricións. Entre estas cláusulas merecen ser destacadas as seguintes:

- Restrición de clave primaria. Sintaxe:
- ```
[CONSTRAINT nome_restrición] PRIMARY KEY [tipo_índice] (nome_de_columna [, ...])
```
- Define unha clave primaria nunha táboa para identificar de forma exclusiva cada unha das súas filas. Hai que ter en conta que:
- Só se pode definir unha restrición PRIMARY KEY por táboa.
  - A clave primaria pode ser unha columna ou unha combinación de columnas.



- Os valores que se gardan na clave primaria teñen que ser exclusivos. Non pode haber dúas filas que tomen o mesmo valor para a clave primaria.
  - Ningunha columna que forme parta da clave primaria pode tomar o valor NULL.
- **Restrición de clave foránea (allea ou externa). Sintaxe:**

```
[CONSTRAINT nome_restrición] FOREIGN KEY (nome_de_columna [, ...])
REFERENCES nome_de_táboa (nome_de_columna [,...])
[{ON DELETE|ON UPDATE} {RESTRICT|SET NULL|SET DEFAULT|CASCADE}]
```

É importante recordar que a clave foránea e a clave primaria á que fai referencia teñen que pertencer ao mesmo dominio (mesmo tipo, tamaño e propiedades). O código de erro 150, móstrase cando non se pode crear unha clave foránea e a maioría das veces prodúcese porque non se cumpre esta restrición.

A restrición FOREIGN KEY implementa a integridade referencial. Esta restrición define unha columna que fai referencia a unha columna que ten que ter unha restrición PRIMARY KEY ou UNIQUE na mesma ou noutra táboa.

Hai que ter en conta que:

- A clave foránea pode estar formada por unha ou máis columnas. O número de columnas e os tipos de datos que se especifican na instrución FOREIGN KEY ten que coincidir co número de columnas e os tipos de datos da cláusula REFERENCES.
- As columnas definidas como clave foránea deben ter asociado un índice para optimizar o seu rendemento. A partir da versión 5.0 de MySQL, cando se crea unha clave foránea créase o índice de forma automática.
- Cando se intenta realizar unha operación de modificación ou borrado dunha fila da táboa pai (a que figura en REFERENCES) e hai filas nalgunha táboa que conteñen claves foráneas que sinalan a esa fila, terase en conta a acción especificada nos apartados ON UPDATE e ON DELETE da cláusula FOREIGN KEY. As accións permitidas para conservar a integridade referencial nas operacións de borrado (DELETE) e modificación (UPDATE), 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. Exemplo: No caso de cambiar o valor da clave primaria para unha fila na táboa de clientes, cámbiase 'en cascada' o valor da columna definida como clave foránea na táboa de facturas para facer referencia ao cliente ao que pertence a factura, nas filas que fan referencia a ese cliente.

```
constraint fk_facturas_clientes foreign key (cliente) references clientes (id)
on delete restrict
on update cascade
```

- 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: no estándar ANSI SQL-92, a opción NO ACTION 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: borra ou actualiza a fila na táboa pai e garda o valor definido por defecto para a columna que é clave foránea e fai referencia a esa fila. Non está implementada en MySQL.
- **Restricións de unicidade para crear índices que non permiten valores repetidos nunha columna. Sintaxe:**

```
[CONSTRAINT nome_restrición] UNIQUE [INDEX|KEY]
[nome_de_índice] [tipo_índice] (nome_de_columna [, ...])
```

A restrición UNIQUE ou de clave única especifica que dúas filas non poden ter o

mesmo valor na columna. Esta restrición implementa a integridade de entidade creando un índice único para a columna. É útil cando xa se ten unha clave primaria, como por exemplo un identificador de empregado, pero se desexa garantir que outras columnas, como o número do permiso de conducir dun empregado ou o DNI, tamén sexan exclusivos.

```
constraint pk_empregado primary key (identificador),
constraint iu_empregado_dni unique (dni), # a columna dni non admite valores repetidos
```

Hai que ter en conta que:

- Pode permitir o valor nulo.
- Pode haber varias restricións UNIQUE nunha mesma táboa.
- Pode aplicar a restrición UNIQUE a unha columna ou a unha combinación de varias columnas que teñen que ter valores exclusivos.

■ Restricións CHECK para comprobación de valores do dominio válidos.

```
CHECK (expresión_lóxica).
```

Restrinxe os valores que se poden almacenar nunha columna, que teñen que cumprir a condición representada pola expresión lóxica. Exemplo: o código de oficina non pode ter un valor inferior a 1, nin superior a 100, é dicir, que o código da oficina estea nun rango comprendido entre 1 e 100.

```
check (codOficina between 1 and 200),
```

Hai que ter en conta que:

- A restrición comproba os datos cada vez que se insire unha nova fila (INSERT), ou cando se modifica unha fila que xa existe (UPDATE).
- A expresión pode facer referencia a outras columnas da mesma táboa.
- A expresión non pode conter subconsultas.

Esta restrición non está implementada en MySQL actualmente; non dá erro de sintaxe pero non fai a comprobación de valores.

Para poder borrar ou modificar as restricións, hai que facer referencia ao nome da restrición, polo que é recomendable asignarlles un nome no momento que se crean empregando unha regra. No caso de non darlles un nome, MySQL asígnalles un nome de maneira automática. Unha regra posible para nomear as restricións podería ser:

- Para claves primarias: pk\_nomeTáboa.
- Para claves foráneas fk\_nomeTáboa\_nomeTáboareferenciada.
- Para índices únicos iu\_nomeTáboa\_nomeColumna.

Exemplo de sentenza para a creación da táboa de oficinas tendo en conta as seguintes restricións de integridade impostas polo modelo:

- A clave primaria é a columna *num\_oficina*.
- A columna *num\_oficina* só permite valores entre 1 e 200, ambos incluídos.
- En cada cidade só pode haber unha oficina.
- A columna *provincia* toma por defecto o valor 'Lugo'.
- A columna *director* é unha clave foránea que fai referencia á táboa *empregado*. Contén o *id* do empregado que dirixe a oficina. No caso de executar esta sentenza podudiríase un erro porque á táboa *empregado* non existe.

```
create table oficina
(
 num_oficina integer not null,
 nome varchar(60),
 enderezo varchar(60),
 codpostal char(5),
 director integer,
 cidade char(20),
 provincia char(20) default "Lugo",
 salario decimal(8,2),
```

```

constraint pk_oficina primary key (num_oficina),
constraint iu_oficina_cidade unique (cidade), # só unha oficina por cidade
constraint fk_oficina_empregado foreign key (director) references empregado (id)
 on delete set null
 on update cascade,
check (num_oficina between 1 and 200)
) engine = InnoDB;

```

Pódense crear novas restricións e modificar ou borrar as restricións que xa existen utilizando a sentenza ALTER TABLE, que se ve na seguinte actividade.

## Asociar estruturas de índices ás columnas

No momento da creación da táboa, pódense crear estruturas de índices asociadas ás columnas, utilizando a cláusula INDEX. Sintaxe:

```
{KEY|INDEX} [nome_de_índice] (nome_de_columna[(lonxitude)] [ASC|DESC] [, ...])
```

Os índices serven para axilizar as operacións de busca. Poden estar formados por máis dunha columna ou incluso por parte dunha columna. MySQL permite seleccionar a orde do índice (ascendente ou descendente). Exemplos:

```

-- índice asociado á columna nome
index idx_oficina_nome (nome)

-- índice asociado aos 15 primeiros caracteres da columna enderezo
key idx_oficina_enderezo (enderezo(15))

-- índice asociado á combinación das columnas numero_oficina e nome en orde descendente
index idx_oficina_numeroNome (numero_oficina,nome) desc

```

Para poder borrar ou modificar as estruturas de índices, hai que facer referencia ao nome do índice, polo que é recomendable asignarlle un nome empregando unha regra. Exemplo: *idx\_nomeTaboa\_nomeColumna*. No caso de non asignarlle un nome, o sistema asígnalle un de forma automática.

## Información sobre táboas

Para obter información sobre as táboas creadas, e as restricións asociadas a elas, pódense utilizar sentenzas SHOW propias de MySQL, ou as táboas da base de datos estándar ANSI INFORMATION\_SCHEMA. Exemplos:

- Coa sentenza SHOW, propia de MySQL:

```

show tables from practicas1; # mostra os nomes das táboas da base de datos practicas1
show create table fotografia; # mostra a sentenza CREATE TABLE para esa táboa

```

- Consultando a base de datos INFORMATION\_SCHEMA:

```

-- consulta das táboas do servidor
select * from INFORMATION_SCHEMA.TABLE
-- restricións de táboa: claves primaria, únicas e foráneas
select * from INFORMATION_SCHEMA.TABLE_CONSTRAINTS
-- restricións de integridade referencial
select * from INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS

```

Na seguinte táboa móstranse os nomes e a información que conteñen algunhas das táboas da base de datos INFORMATION\_SCHEMA que poden ser de utilidade nesta actividade:

| Nome da táboa           | Información que contén   | Equivalencia SHOW       |
|-------------------------|--------------------------|-------------------------|
| SCHEMATA                | Bases de datos           | SHOW DATABASES          |
| TABLES                  | Táboas da base de datos  | SHOW TABLES             |
| COLUMNS                 | Columnas das táboas      | SHOW COLUMNS FROM táboa |
| TABLE_CONSTRAINTS       | Restricións de táboas    |                         |
| REFERENTIAL_CONSTRAINTS | Restricións referenciais |                         |
| ENGINES                 | Motores de almacenamento | SHOW ENGINES            |

|            |                              |                       |
|------------|------------------------------|-----------------------|
| STATISTICS | Información sobre os índices | SHOW INDEX FROM táboa |
|------------|------------------------------|-----------------------|

## Borrado dunha táboa

A sentenza DROP TABLE permite borrar táboas. Sintaxe:

```
DROP [TEMPORARY] TABLE [IF EXISTS] nome_táboa [, nome_táboa] ...
```

Exemplo:

```
drop table copia_oficina;
```