Creación da estrutura de bases de datos relacionais

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 podemos empregar nomes que só leven letras, números e guión baixo, tratando de evitar outros carácteres especiais (ñ, acentos, ...). A razón desta recomendación é que aínda que o noso SXBD permita o uso de carácteres especiais pode que no futuro teñamos que migrar as nosas bases de datos a outro SXBD, ou a outro sistema operativo, que teña normas máis restritivas nos nomes que se poden empregar.
- **Uso indistinto de maiúsculas, minúsculas, ou unha mestura de ambas, para nomear.** Debería-se 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 como se almacenan os obxectos das bases de datos (*tablespaces*), ou os ficheiros de datos (*datafiles*).

Linguaxe SQL

SQL corresponde ao acrónimo de *Structured Query Language* (Linguaxe Estruturado de Consultas) e é unha ferramenta para organizar, xestionar e recuperar datos almacenados nunha base de datos relacional. Inclúe varias linguaxes, cada unha delas cunha función específica:

- **DDL = Data Definition Language = Linguaxe de Definición de Datos:**

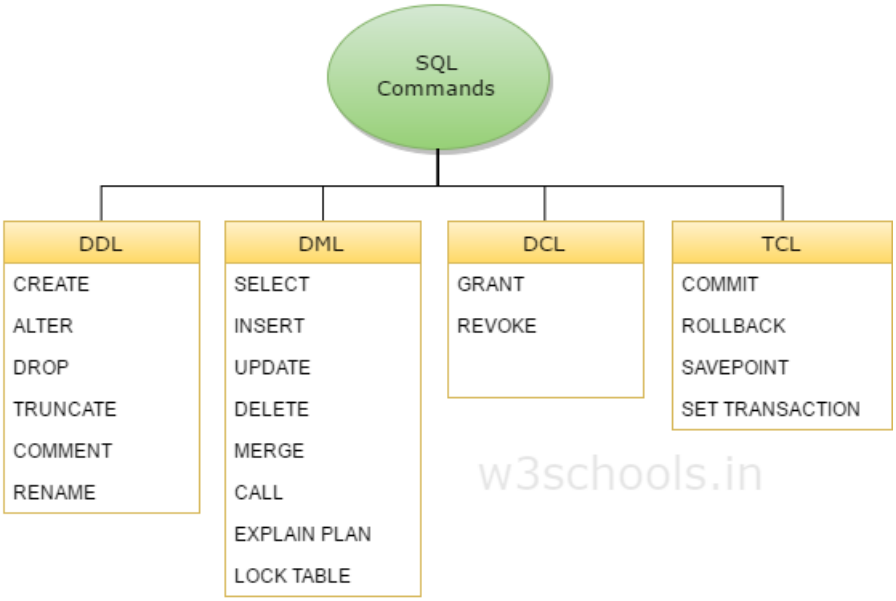
  Permite a creación do esquema da base de datos e a organización física dos datos almacenados.

- **DML = Data Manipulation Language = Linguaxe de Manipulación de Datos:** 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.

- **DCL = Data Control Language = Linguaxe de Control de Datos:** 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.

- **TCL = Transaction Control Language = Linguaxe de Control de Transacións**

  Permite manter a integridade dos datos ao ser manipulados mediante operacións que requiren varios pasos. O seu obxectivo é garantir que a operación se realiza de forma completa, ou ben non se realiza en absoluto.



Formas de utilizar SQL

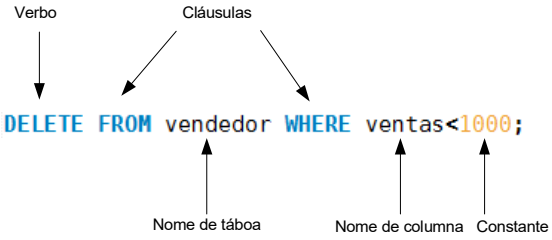
SQL é unha linguaxe declarativa de alto nivel, é dicir, non procedimental. Isto significa que as sentenzas SQL especifican o que se quere obter, pero non a forma de conseguilo. A orde interna na que se executan as operacións asociadas a unha sentenza SQL é establecida polo optimizador de consultas do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

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, como a creación dunha nova táboa, a consulta de datos dunha ou máis táboas, ou a inserción de novos datos na base de datos.



Todas as sentenzas SQL teñen a mesma estrutura:

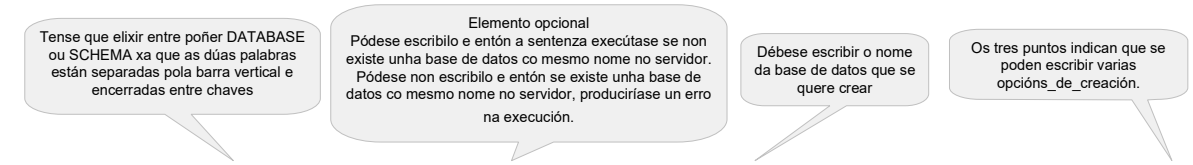
- Empezan cun verbo, que é unha palabra clave que indica a acción que ten que executar o SXBD. Algúns destes verbos son: CREATE, INSERT, DELETE, ou SELECT.
- A sentenza continúa cunha ou máis cláusulas. Unha cláusula pode especificar os datos sobre os que debe actuar a sentenza, ou proporcionar máis 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.

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 diferenza 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, 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.




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

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

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.

BASE DE DATOS:

1.- Crear unha base de datos

Sintaxe da sentenza para crear unha base de datos:

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

Onde:

- **DATABASE** e **SCHEMA** son sinónimos. Pódese utilizar calquera das dúas opcións.
- A parte optativa **IF NOT EXISTS**, indica que só se creará a base de datos no caso de que non exista previamente; se existe mostrarase unha mensaxe de advertencia (*warning*). Se non se pon esta opción, prodúcese un erro no caso de que a base de datos exista.
- **nome_BD** é nome que se vai asignar á base de datos e 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
  ```

2.- Poñer en uso unha base de datos

Permite seleccionar unha base de datos e poder traballar con ela.

Sintaxe:

```
USE nome_BD;
```

Exemplo: `USE proba;`

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

3.- Borrar unha base de datos

Sintaxe da sentenza para borrar unha base de datos:

```
DROP DATABASE [IF EXISTS] nome_BD;
```

Exemplo: `DROP DATABASE IF EXISTS proba;`

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

4.- Modificar unha base de datos

Sintaxe da sentenza que permite cambiar as características globais dunha base de datos:

```
ALTER {DATABASE | SCHEMA} nome_BD [opcións_a_modificar] ... ;
```

TÁBOA

1.- 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] ;
```

Onde:

- **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, opcionalmente, asociarlle as restricións necesarias.

  Os SXBDR soportan varios tipos de datos divididos en categorías (numéricos, data e hora, cadeas de carácteres, ..), que se detallan 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 carácteres.
  - **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, InnodDB, ...}]
  [[DEFAULT] CHARACTER SET nome_xogo_carácteres]
  [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).
  - **CHARACTER SET** e **COLLATE**, permiten establecer o conxunto de carácteres por defecto para as columnas que se crean nesa táboa, e a forma en que se ordenan e comparan os carácteres.
  - **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 *practicas1*:

```
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 DEFAULT 'Apelidos e nome do autor',
    data        DATE                NULL COMMENT 'Data en que foi tomada',
    PRIMARY KEY (id)
) ENGINE = InnoDB;
```

1.1.- 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 carácteres		`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 0	8388607 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 <= p <= 24, 8 bytes se 25 <= p <= 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 determínanse separadamente. Cada múltiplo de nove díxitos require catro bytes, e o díxito "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 carácteres para MySQL

Tipo	Almacenamento requirido
CHAR(M), BINARY(M)	M bytes, 0 <= M <= 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 carácteres 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
LONGBLOB, 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 os tipos en MySQL:

- **VALORES DE TIPO ENTEIRO:**

  Tipos do estándar `SQL`, `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.

- **VALORES NUMÉRICOS CON CIFRAS DECIMAIS EXACTAS OU DE COMA FIXA:**

  Tipos de datos `DECIMAL` (ou `DEC`) e `NUMERIC` (son sinónimos).

  Sintaxe: `DECIMAL(m,d)`

  Onde:

  - ***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.

- **VALORES DECIMAIS APROXIMADOS OU DE COMA FLOTANTE**

  Tipos `FLOAT` (para simple precisión) e `DOUBLE` (para dobre precisión).

- **VALORES DUN BIT**

  Tipo de datos `BIT`, dende MySQL 5.0.3

  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*'**, onde *value* é un valor binario. Exemplos: 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`.

- **VALORES QUE REPRESENTAN DATAS E HORAS**

  Dispón dos tipos `DATE`, `TIME`, `DATETIME` e `TIMESTAMP`.

  - `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.
  - `TIME` utilízase para representar horas co formato **HH:MM:SS[.fracción]**.
  - `DATETIME` e `TIMESTAMP` utilízase para representar valores que inclúen data e hora, con 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'.

    O rango permitido 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.

- **ANOS**:

  O tipo `YEAR` ocupa 1 byte e é usado para representar valores de anos.

  Pode ser declarado como `YEAR(4)` para anos de 4 cifras (2022) ou como `YEAR(2)` para anos de 2 cifras (22), sendo recomendable o primeiro.

- **CADEAS DE CARACTERES:**

  Tipos `CHAR` e `VARCHAR`.

  - `CHAR(n)` permite almacenar cadeas de carácteres de lonxitude fixa, ata un máximo de 255.
  - `VARCHAR(n)` permite almacenar cadeas de carácteres de lonxitude variable, ata un máximo de 65532.

  Exemplo:

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

  Por defecto `VAR` e `VARCHAR` non diferencian entre minúsculas e maiúsculas, para que diferencien deben acompañarse da opción `BINARY`.

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

- **CADEAS DE BYTES**:

  Tipos `BINARY` e `VARBINARY`

  Son similares a `CHAR` e `VARCHAR`, pero almacénanse como cadeas de carácteres binarias, é dicir, conteñen cadeas de bytes en lugar de cadeas de carácteres. Esto implica que diferencian as minúsculas das maiúsculas.

- **CADEA DE CARACTERES OU CADEAS DE BYTES DE LONXITUDE VARIABLE:**

  Tipos **BLOB** e **TEXT**, para tratar gran cantidade de datos variables.

  Cada valor **BLOB** ou **TEXT** represéntase internamente como un obxecto a parte.

  - As columnas tipo **TEXT** trátanse como cadeas de caracteres non binarias (de caracteres).

    Hai catro tipos: **TINYTEXT**, **TEXT**, **MEDIUMTEXT** e **LONGTEXT**.

  - As columnas tipo **BLOB** trátanse como cadeas de caracteres binarias (de byte). Permiten almacenar calquera tipo de información: arquivos de texto, imaxes, arquivos de son ou vídeo, ...

    Para ordenalas e comparalas úsase o valor numéricos dos bytes.

    Hai catro tipos: **TINYBLOB**, **BLOB**, **MEDIUMBLOB** e **LONGBLOB**.

- **CADEAS DE CARACTERES CON RESTRICIÓNS (ENUMERADOS E CONXUNTOS):**

  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**, pode almacenar un só 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 carácteres. 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 columnas[1].

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

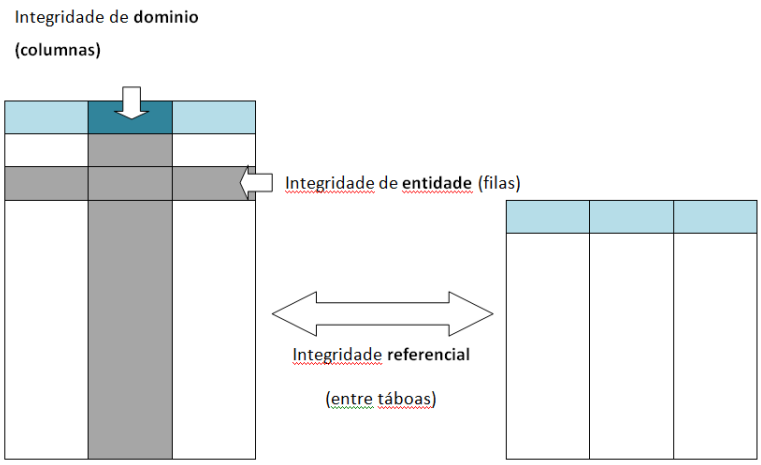
1.2.- Implementación da integridade mediante restricións

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 datos cumpran eses 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 comproban que os devanditos criterios se cumpren. Débese limitar o uso da integridade procedimental a situacións excepcionais e a aquelas cunha lóxica complicada.

  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 explícitamente 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.

Especificar restricións ao crear unha táboa:

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

  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. Define unha columna que fai referencia a outra cunha 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, 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, usarase a acción especificada nos apartados `ON UPDATE` e `ON DELETE` da cláusula `FOREIGN KEY` para conservar a integridade. Estas 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`: 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.

Exemplo:

No caso de cambiar o valor da clave primaria para unha fila na táboa de clientes, cámbiase 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
```

- **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), # dni non admite 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.**

  Sintaxe:

  ```
  CHECK (expresión_lóxica)
  ```

  Restrinxe os valores que se poden almacenar nunha columna, que teñen que cumprir a condición representada pola expresión lóxica.

  Exemplo:

  ```
  CHECK (codOficina BETWEEN 1 AND 200),
  ```

  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.

  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_nomeTaboa_nomeColumna.

Exemplo: 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 FOREING KEY (director) REFERENCES empregado (id)
                    ON DELETE SET NULL
                    ON UPDATE CASCADE,
                CHECK (num_oficina BETWEEN 1 AND 200)
) ENGINE = InnoDB;
```

1.3.- Asociar estruturas de índices ás columnas

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).

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] [, ...])
```

Exemplos:

```
-- índice asociado á columna nome
INDEX idx_oficina_nome (nome)
-- índice asociado aos 15 primeiros carácteres da columna enderezo
KEY idx_oficina_enderezo (enderezo(15))
-- índice asociado á combinación de 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.

2.- Creación dunha táboa a partir doutra que xa existe

MySQL permite crear unha táboa partindo do esquema doutra táboa que xa exista.

Sintaxe:

```
CREATE [TEMPORARY] TABLE nome_táboa {LIKE nome_táboa_existente | sentenza SELECT};
```

Aspectos a ter en conta:

- No caso de utilizar **LIKE** crearase unha táboa baleira baseada na definición da outra táboa, incluíndo algúns atributos das columnas e os índices definidos na táboa orixinal.
- No caso de utilizar **SELECT** créase a estrutura da táboa e ademais cópianse os datos resultantes da sentenza. No caso de producirse un erro cando se copian só datos, non se crea a táboa.

En calquera dos dous casos hai atributos que non se van a copiar, e haberá que telo en conta[2].

Algunha das cousas a ter en conta cando se crea unha táboa partindo doutra:

- Claves foráneas: As restricións **FOREIGN KEYS** non se copian en ningún dos casos. Pódense engadir utilizando unha sentenza **ALTER TABLE** para incluílas despois de facer a copia.
- En táboas co motor de almacenamento **MyISAM**, as opcións de táboa **DATA DIRECTORY** e **INDEX DIRECTORY** tampouco se copian. Os ficheiros de datos e índices gárdanse no directorio de datos especificado para a base de datos destino.

Exemplo:

```
CREATE TABLE copia_oficina LIKE oficina;
```

2 http://cambrico.net/mysql/duplicar-o-clonar-tablas-en-mysql

3.- 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;  # nomes das táboas da BD practicas1
SHOW CREATE TABLE fotografia; # 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

4.- 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;
```

5.- Modificación do esquema dunha táboa

Co paso do tempo é normal que se teñan que facer adaptacións no esquema das táboas das bases de datos, pola aparición de novos requirimentos, novas restricións, ou a desaparición dalgunhas das existentes. Algunhas veces, ter que facer cambios débese a non terlle dedicado o tempo suficiente ás fase de deseño conceptual e lóxico; é moi recomendable pararse a facer un bo deseño antes de empezar a escribir código para crear a base de datos.

A sentenza **ALTER TABLE** permite facer modificacións no esquema dunha táboa que xa existe.

Sintaxe:

```
ALTER TABLE nome_táboa
    [especificacion_alter][,
     especificacion_alter]
     ...;
```

A *especificación_alter* pode ser:

```
      opcións de táboa
    | RENAME [TO|AS] nome_táboa_novo

    | ADD    [COLUMN]  nome_columna definición_columna [FIRST | AFTER nome_columna]
    | ADD    [COLUMN] (nome_columna definición_columna, …)
    | RENAME  COLUMN  nome_columna TO|AS nome_novo_columna
    | MODIFY [COLUMN] nome_columna           nova_definición_columna [FIRST|AFTER nome_columna],
    | CHANGE [COLUMN] nome_columna nome_novo nova_definición_columna [FIRST|AFTER nome_columna],
    | ALTER  [COLUMN] nome_columna {SET DEFAULT valor | DROP DEFAULT},
    | DROP   [COLUMN] nome_columna,

    | ADD [CONSTRAINT [nome_restrición]] PRIMARY KEY (lista_columnas),
    | ADD [CONSTRAINT [nome_restrición]] FOREIGN KEY [nome_índice](lista_columnas)
         REFERENCES nome_táboa (lista_de_columnas) [ON DELETE opción] [ON UPDATE opción]
    | ADD [CONSTRAINT [nome_restrición]] CHECK (expr)
    | ADD [CONSTRAINT [nome_restrición]] UNIQUE {INDEX|KEY} [nome_indice](lista_columnas),
    | DROP PRIMARY KEY,
    | DROP FOREIGN KEY nome_restrición,

    | ADD {INDEX|KEY} [nome_índice] (columnas_índice),
    | DROP {INDEX | KEY} nome_índice,
```

Consideracións sobre a sintaxe:

- As **opcións de táboa** afectan ás características da táboa e son as mesmas que se utilizan na sentenza **CREATE TABLE**.

  Algunhas opcións de táboa son:

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

- `DATA DIRECTORY` e `INDEX DIRECTORY` permiten cambiar as rutas absolutas nas que se almacenan os datos e os índices.
- `ENGINE` permite cambiar o motor de almacenamento asociado á táboa.
- `CHARACTER SET` e `COLLATE` permiten cambiar o conxunto de carácteres e o sistema de colación predeterminados para as columnas que se crean nesa táboa. Non afecta ás columna que xa están creadas. Para cambiar o contido das columnas tipo cadea de carácteres, hai que utilizar a cláusula `CONVERT TO CHARACTER SET`.
- `AUTO_INCREMENT` permite cambiar o número inicial para columnas autoincrementais.

- As cláusulas `ADD` permiten engadir novas propiedades e as cláusulas `DROP` permiten eliminalas.
- Se ao engadir unha columna (`ADD`) non se especifica a cláusula `FIRST` | `AFTER`, a nova columna engádese ao final da táboa.
- Non é posible engadir unha columna tipo autoincremento se a táboa non está baleira.
- Non se pode modificar `NULL` por `NOT NULL` se a táboa contén valores nulos para a columna a modificar; a operación inversa non presenta ningún problema.
- Para cambiar a definición dunha columna, hai que utilizar a cláusula `MODIFY`, pero se ademais da definición tamén se quere cambiar o nome, entón hai que utilizar a cláusula `CHANGE`. O editor da actual versión de Workbench non recoñece a cláusula `MODIFY` e marca a liña como un erro aínda que se executa correctamente.
- A cláusula `ALTER` permite modificar o valor por defecto para unha columna.
- A cláusula `RENAME` permite cambiar o nome da táboa.

Exemplo de modificación da estrutura da táboa *fabricante* da base de datos *practicas1*:

```
ALTER TABLE fabricante
    ADD COLUMN pais VARCHAR(60) DEFAULT NULL,
    ADD COLUMN enderezo VARCHAR(200) NOT NULL AFTER idFabricante,
    ADD INDEX idx_fabricante_nome (nome),
    ENGINE = Innodb;
```

6.- Engadir relacións e restricións de clave foránea

Pódense engadir relacións e restricións de clave foránea de dúas formas:

- Crear ao mesmo tempo as táboas e as relacións, empregando a sentenza `CREATE TABLE`. Neste caso hai que ter en conta a orde en que se crean as táboas, xa que non se pode crear unha táboa que conteña unha clave foránea se aínda non está creada a táboa á que fai referencia. Isto pode representar un problema no caso de relacións bidireccionais.
- Crear primeiro as táboas e establecer as relacións despois empregando sentenzas `ALTER TABLE.`

Por exemplo, supóñase que a entidade *empregado* está relacionada coa entidade *departamento* cunha relación *traballa* de tipo N:1, e *departamento* está relacionada con *empregado* coa relación *dirixe*, de tipo 1:1, con cardinalidade mínima 0. Ao crear primeiro a táboa *empregado* e despois *departamento*, provocaríase un erro porque na orde de creación da táboa *empregado* se define unha clave foránea que fai referencia á táboa *departamento*, que aínda non existe. O mesmo ocorre se empeza creando *departamento*.

En MySQL pódese solucionar o problema anterior desactivando a verificación de claves foráneas, poñendo o valor **0** ou **OFF** na variable **`foreign_key_checks`**.

```
SET foreign_key_checks = '0';    --desactivar
SET foreign_key_checks = '1';    --activar
```

ÍNDICES

Os índices son o principal medio para acelerar o acceso aos datos contidos nas táboas, en especial en aquelas consultas nas que se mostran datos de varias táboas. Hai algúns motores de almacenamento de MySQL que non permiten utilizar índices: Merge, Archive e CSV.

O SXBD sempre establece un índice para a clave primaria dunha táboa e pódense crear índices a maiores para as para as claves foráneas e as columnas que son utilizadas frecuentemente en condiciones de busca. Á hora de tomar unha decisión deste tipo, hai que ter en conta que aínda que a busca sexa máis rápida, o índice ocupa un espazo de almacenamento extra e ademais debe ser actualizado cada vez que se executa unha sentenza que insire unha fila nova e cada vez que se borre ou modifica unha fila da táboa.

A creación e borrado de índices pódese facer no momento da creación da táboa coa sentenza **CREATE TABLE**, ou posteriormente, utilizando a sentenza **ALTER TABLE**. Tamén se poden crear e modificar coas sentenzas propias para manexo de índices.

1.- Creación de índices

A sentenza **CREATE INDEX** permite a creación de novas estruturas de índices.

Sintaxe:

```
CREATE [UNIQUE] INDEX nome_indice ON nome_táboa (lista_columnas[ASC|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 como por exemplo: *idx_nomeTaboa_nomeColumna*.

Exemplo de sentenza que crea un índice asociado á columna *dni* da táboa *alumno*:

```
CREATE INDEX idx_alumno_dni
    ON alumno (dni);
```

2.- Borrado de índices

A sentenza **DROP INDEX** permite borrar un índice.

Sintaxe:

```
DROP INDEX nome_indice ON nome_táboa;
```

Exemplo de sentenza para borrar un índice asociado á columna *dni* da táboa *alumno:*

```
DROP INDEX idx_alumno_dni
    ON alumno;
```