Introdução a SQL

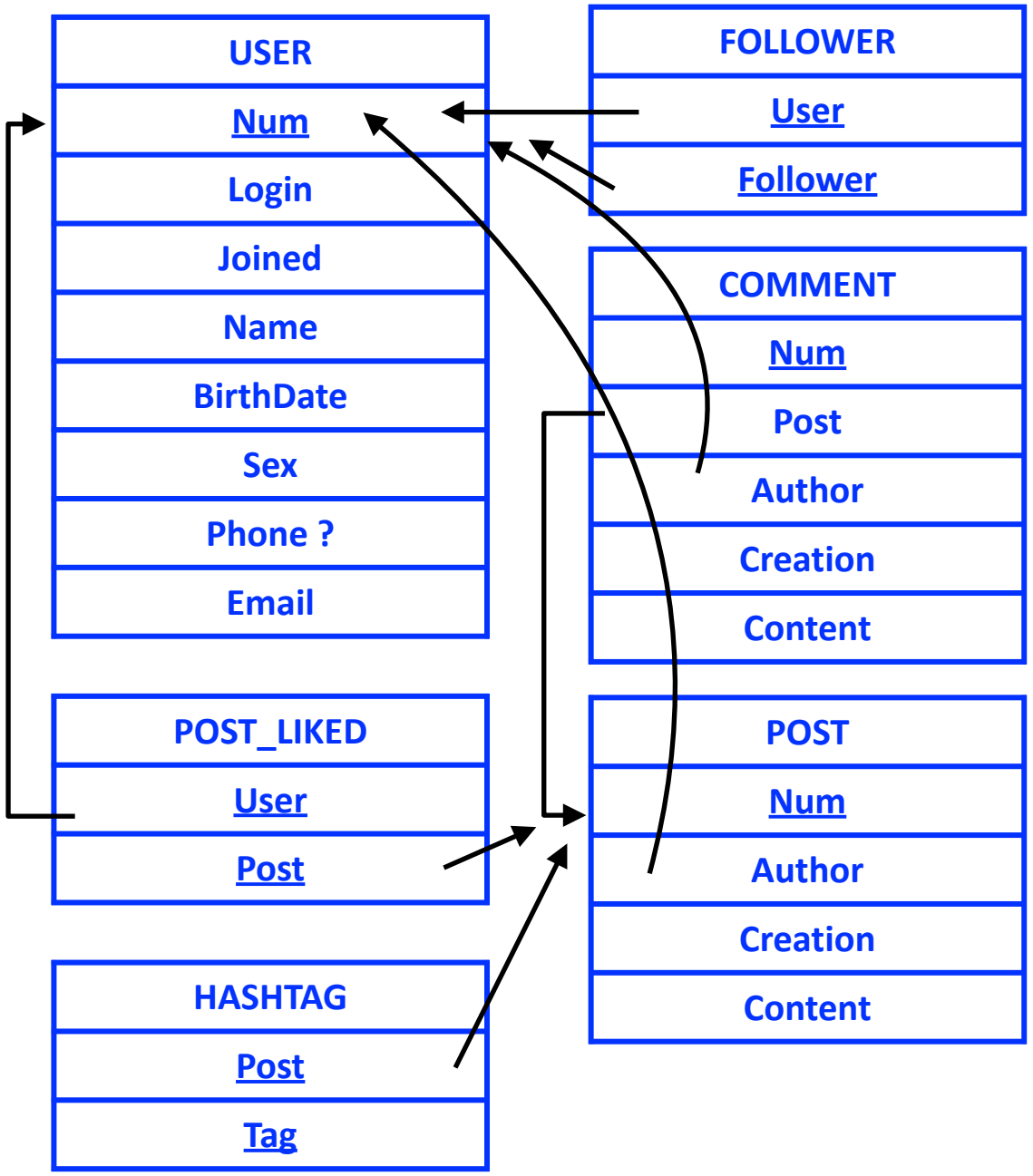
Bases de Dados (CC2005)
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

Eduardo R. B. Marques — DCC/FCUP

SQL

- A **SQL (Structured Query Language)** é uma linguagem declarativa que permite definir e manipular bases de dados.
- É a linguagem standard para SGBDs baseados no modelo relacional.
- A SQL é uma **DDL ("Data Definition Language")** porque permite definir a estrutura da BD — o **esquema**.
- A SQL é uma **DML ("Data Manipulation Language")** porque permite manipular o conteúdo da BD — os **dados** que instanciam um **esquema**.

BD exemplo



- Variante da rede social "Livro das Caras" que vimos nas aulas práticas.
- Utilizadores, "posts", comentários têm todos um identificador numérico único.
- Código SQL disponível **aqui** para uma pequena instância com:
  - 11 utilizadores
  - 28 relações de seguimento entre utilizadores
  - 12 "posts"
  - 18 "hashtags" aplicadas a "posts"
  - 15 "likes"
  - 6 comentários a "posts"

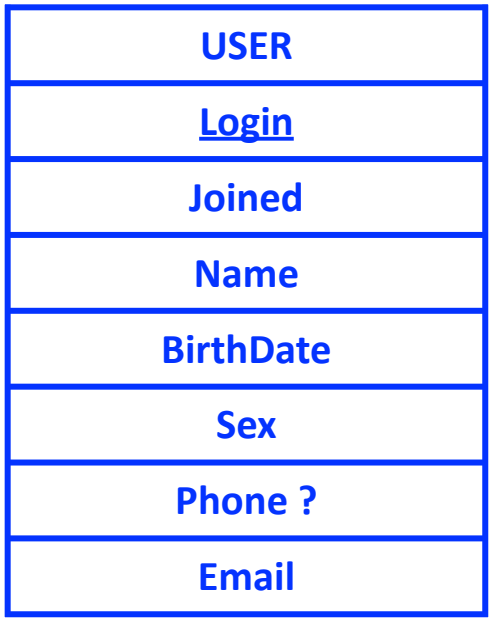
Criação de tabelas

Criação de uma tabela em SQL

```
CREATE TABLE <NOME DA TABELA>
(
    <NOME ATRIBUTO 1>   <TIPO 1> <RESTRIÇÕES DE DOMINIO 1> ,
    … ,
    <NOME ATRIBUTO N>   <TIPO N> <RESTRIÇÕES DE DOMINIO N>
    …
    PRIMARY KEY(<ATRIBUTOS QUE FORMAM CHAVE PRIMÁRIA>),
    …
    UNIQUE(<ATRIBUTOS QUE FORMAM UMA CHAVE (SECUNDÁRIA)>),
    …
    FOREIGN KEY(<ATRIBUTOS QUE FORMAM UMA CHAVE EXTERNA>)
    REFERENCES <NOME_DE_OUTRA_TABELA 1>(<CHAVE_PRIMÁRIA>)
    …
);
```

Tabela USER (primeira formulação)

```
CREATE TABLE USER
(
  Login      PRIMARY KEY VARCHAR(16),
  Joined     DATE NOT NULL,
  Name       VARCHAR(128) NOT NULL,
  BirthDate  DATE NOT NULL,
  Sex        ENUM('M', 'F') NOT NULL,
  Phone      INT DEFAULT NULL,
  Email      VARCHAR(64) NOT NULL
);
```



- **Chave primária**: Login.
- Tabela não tem chaves externas.
- Atributos não-opcionais indicados com NOT NULL (todos excepto EMAIL).
- NOT NULL não precisa de ser especificado para uma chave primária (não são admitidos valores NULL).

Tipos principais para campos (MySQL)

Números Inteiros	TINYINT (1 byte), SMALLINT (2) MEDIUMINT (3), INT (4), BIGINT (8)
Números de vírgula flutuante	DECIMAL, NUMERIC, FLOAT, DOUBLE
Sequências de caracteres	CHAR, VARCHAR, ENUM, TEXT / MEDIUMTEXT /LONGTEXT
Tempo	DATE, DATETIME, TIME, TIMESTAMP, YEAR

- Referência: MySQL Reference Manual - Data Types

Tabela USER — reformulação

```
CREATE TABLE USER
(
  Num       INT PRIMARY KEY AUTO_INCREMENT,
  Login     VARCHAR(16) UNIQUE NOT NULL,
  Joined    DATE NOT NULL,
  Name      VARCHAR(128) NOT NULL,
  BirthDate DATE NOT NULL,
  Sex       ENUM('M', 'F') NOT NULL,
  Phone     INT DEFAULT NULL,
  Email     VARCHAR(64) NOT NULL
);
```

- **Chave primária**: Num
- **Chave secundária** (por via de UNIQUE): Login

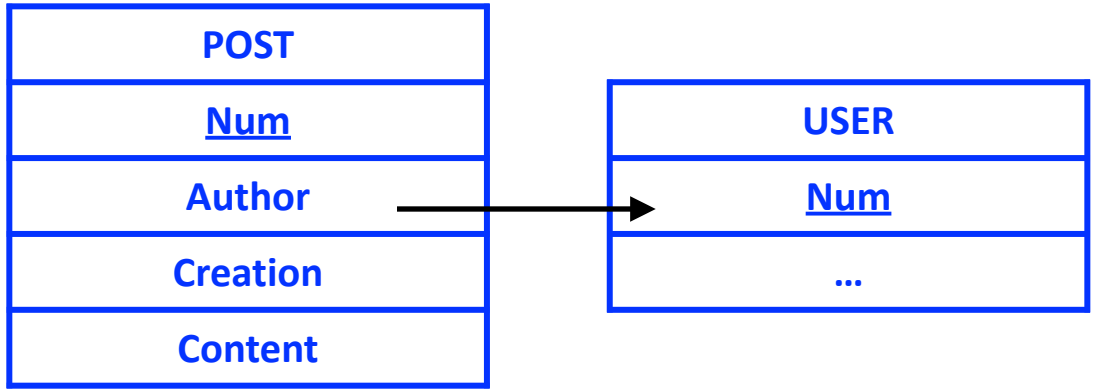
Uso de AUTO_INCREMENT

```
CREATE TABLE USER
(
  Num       INT PRIMARY KEY AUTO_INCREMENT,
  Login     UNIQUE VARCHAR(16) NOT NULL,
  Joined    DATE NOT NULL,
  Name      VARCHAR(128) NOT NULL,
  BirthDate DATE NOT NULL,
  Sex       ENUM('M', 'F') NOT NULL,
  Phone     INT DEFAULT NULL,
  Email     VARCHAR(64) NOT NULL
);
```

- O uso de **AUTO_INCREMENT** permite que o valor de um chave primária tenha um valor sequência associado que é automaticamente incrementado em cada inserção (como veremos depois).
- É uma solução muitas vezes conveniente, pois facilita referências entre tabelas via chaves externas usando campos que ocupam pouco espaço.

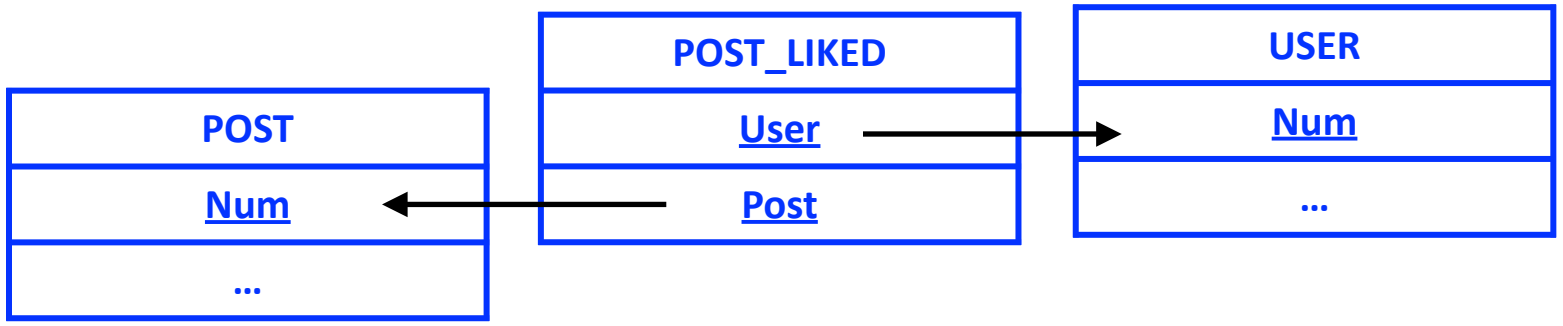
Chaves externas — exemplo

```
CREATE TABLE POST
(
  Num       INT PRIMARY KEY AUTO_INCREMENT,
  Author    INT NOT NULL,
  Creation DATETIME NOT NULL,
  Content   TEXT NOT NULL,
  FOREIGN KEY(Author) REFERENCES USER(Num)
);
```



Chaves externas — outro exemplo

```
CREATE TABLE POST_LIKED
(
  User INT NOT NULL,
  Post INT NOT NULL,
  PRIMARY KEY(User, Post),
  FOREIGN KEY(User) REFERENCES USER(Num),
  FOREIGN KEY(Post) REFERENCES POST(Num)
);
```



A instrução DROP TABLE

```
DROP TABLE IF EXISTS
  POST_LIKED,
  COMMENT,
  HASHTAG,
  POST,
  FOLLOWER,
  USER ;
```

- DROP TABLE **remove do esquema da BD uma ou mais tabelas.**
  - A tabelas deixa de existir no esquema da BD.
  - ... e por inerência os dados associados às tabelas são removidos.
- **Nota**: a cláusula IF EXISTS verifica previamente se as tabelas existem para evitar erros. Analogamente CREATE TABLE suporta a cláusula simétrica IF NOT EXISTS.

CREATE DATABASE / USE / DROP DATABASE

CREATE DATABASE [IF NOT EXISTS] SOCIAL_NETWORK;

- Deve ser executada para criar uma nova base de dados (sinónimo: **CREATE SCHEMA**) com o nome de dado. Sobre a BD poderemos criar tabelas, inserir dados, etc.
- **Nota**: nos PCs dos laboratórios do DCC não podemos criar BDs novas, por questões de configuração estamos limitados ao uso da BD existente **guest.**

USE SOCIAL_NETWORK;

- Para uma BD já criada empregamos a instrução **USE** para indicar qual a BD activa para execução de comandos.

DROP DATABASE [IF EXISTS] SOCIAL_NETWORK;

- Remove uma BD por inteiro (tabelas e dados) de forma incondicional!

Tipos de dados em mais detalhe

Tipos inteiros — modificador UNSIGNED

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
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	-2^{63}	0	$2^{63}-1$	$2^{64}-1$

- MySQL Reference Manual (11.2.1)
- O modificador **UNSIGNED** restringe o domínio a valores não negativos (>= 0), ex. **INT UNSIGNED.**
- **BOOL / BOOLEAN** são implementados como **TINYINT** em MySQL (**TRUE = 1, FALSE = 0**). **INTEGER** é sinónimo de **INT.**

Tipos de vírgula flutuante

- **FLOAT**
  - Números reais representáveis em 32 bits com precisão arbitrária
- **DOUBLE**
  - Números reais representáveis em 64 bits com precisão arbitrária
- **DECIMAL(D, P)**
  - Números reais representáveis na escala de **D** dígitos decimais com precisão de **P** dígitos
  - Exemplo — valores correspondentes a DECIMAL(3,2) podem ser: 100.23 -2.35 2.1 -999.99
  - ... mas não 1234 1.234 -1234.567

Sequências de caracteres

- **CHAR(N)**
  - Sequência de caracteres de **tamanho fixo N**, onde **N <= 255 (2^8-1).** Para um valor de tamanho **L <= N** , a BD armazena sempre **N** caracteres usando o caracter para "padding" à direita.
- **VARCHAR(N)**
  - Sequência de caracteres de **tamanho variável até N caracteres**, onde **N <= 65535 (2^{16} - 1)**. Para um valor de tamanho **L <= N** , a BD só armazena **L** caracteres.
- **ENUM('VALOR 1', …, 'VALOR N')**
  - Tipo enumerado (como exemplificado para USER.Sex)
- **TEXT / MEDIUMTEXT / LONGTEXT**
  - Texto de comprimento variável até 64 KB / 16 MB / 4GB

Tipos de dados temporais

- **YEAR**
  - Ano
- **DATE**
  - Representação de datas com a forma YYYY-MM-DD
- **TIME**
  - Representação de horas com a forma [H]HH:MM:SS
- **DATETIME, TIMESTAMP**
  - Representação de data e hora.
  - Valores com a forma YYYY-MM-DD [H]HH:MM:SS[.dddddd]
  - 6 dígitos de precisão (até micro-segundos)
  - **TIMESTAMP** guardado na BD com conversão para "UTC time"

Manipulação de registos

Formas básicas das instruções

INSERT, SELECT, UPDATE, e DELETE

Instruções essenciais de manipulação de dados

- **INSERT**
  - **Inserção** de registos
- **SELECT**
  - **Consulta** (leitura) de registos (não altera BD)
- **UPDATE**
  - **Actualização** de registos já existentes
- **DELETE**
  - **Remoção** de registos

Inserção de registos — forma básica

```
INSERT INTO
  <NOME DA TABELA>(<ATRIBUTO 1>, …., <ATRIBUTO N>)
VALUES
  (<VALOR 1.1>, …., <VALOR 1.N>),
  (<VALOR 1.2>, …., <VALOR 2.N>),
  … ,
  (<VALOR K.2>, …., <VALOR K.N>);
```

- É identificado o nome da tabela e os atributos a definir na inserção, seguido dos valores para um conjunto de registos.

Exemplo de inserção de registos

```
INSERT INTO USER(Login, Joined, Name, BirthDate, Sex, Email)
VALUES
 ('batman',   '2019-12-16',   'Bruce Wayne',    '1939-01-01', 'M', 'batman@dccomics.com'),
 ('catwoman', '2019-12-16',   'Selina Kyle',    '1940-01-01', 'F', 'catwoman@dccomics.com'),
 ('ziggy',    '2019-12-31',   'Ziggy Stardust', '1972-01-01', 'M', 'ziggy@mars.com'),
 ('lady.z',   '2020-01-01',   'Lady Stardust',  '1972-01-01', 'F', 'ladyz@mars.com');
```

- Números de utilizadores (campo Num) gerados automaticamente.
- Valor de nº de telefone (Phone) usa valor configurado pela cláusula DEFAULT (no caso NULL).

Consulta de registos — forma básica

```
SELECT
   <ATRIBUTO 1>, …., <ATRIBUTO N>
FROM
   <NOME DA TABELA>
WHERE
   <CONDIÇÃO>;
```

- Permite a consulta ("query") dos valores atributos da tabela indicada sob determinada **condição de selecção**, definida pelo que chama a **cláusula WHERE**.
- A "wildcard" * para os atributos define implicitamente **todos os atributos**. Na ausência da cláusula **WHERE** os valores são devolvidos para **todos os registos** da tabela. Vários operadores de seleção podem ser aplicados na cláusula **WHERE**. Damos a seguir apenas alguns exemplos simples.

Exemplos de consultas

```
/* Consulta todos os atributos de todos os utilizadores. */
SELECT * FROM USER;
```

Num	Login	Joined	Name	BirthDate	Sex	Phone	Email
1	joao.pinto	2019-12-01	João Pinto	1976-12-19	M	NULL	azulibranco@fcp.pt
2	semedo	2019-12-02	Carlos Semedo	1985-06-02	M	223774327	semedo@xpto.com
3	maria	2019-12-02	Maria Silva	2005-11-17	F	939939939	maria@xyz.pt
4	pedro	2019-12-10	Pedro Costa	1982-01-03	M	NULL	pc12345@xpto.com
5	mendocas	2019-12-11	Filipa Mendes	2002-05-03	F	NULL	filipa.mendes@gmail.com
6	eva	2019-12-15	Eva Mendes	1975-11-18	F	NULL	i_ate_the_apple@paradise.com
7	pedro2	2019-12-16	Pedro Simões	1993-02-22	M	213884445	simoes333@gmail.com
8	batman	2019-12-16	Bruce Wayne	1939-01-01	M	NULL	batman@dccomics.com
9	catwoman	2019-12-16	Selina Kyle	1940-01-01	F	NULL	catwoman@dccomics.com
10	ziggy	2019-12-31	Ziggy Stardust	1972-01-01	M	NULL	ziggy@mars.com
11	lady.z	2020-01-01	Lady Stardust	1972-01-01	F	NULL	ladyz@mars.com

```
/* Consulta login e nome de utilizadores do sexo masculino. */
SELECT Login,Name FROM USER
WHERE SEX='M';
```

Login	Name
joao.pinto	João Pinto
semedo	Carlos Semedo
pedro	Pedro Costa
pedro2	Pedro Simões
batman	Bruce Wayne
ziggy	Ziggy Stardust

Actualização de registos — forma básica

```
UPDATE
  <NOME DA TABELA>
SET
  <ATRIBUTO 1> = <VALOR 1>,
  …
  <ATRIBUTO K> = <VALOR K>
WHERE
  <CONDIÇÃO>;
```

- Permite a actualização dos valores de um sub-conjunto de atributos para registos que verificam determinada condição.
- Na ausência da cláusula **WHERE** **todos os registos** são considerados para actualização.

Exemplos de actualizações

```
/* Actualiza todos os números de telefone com NULL. */
UPDATE USER
SET Phone = NULL;

/* Actualiza telefone e email para utilizador com login 'batman'. */
UPDATE USER
SET Phone = 238488532, Email = 'batman@gotham.com'
WHERE Login = 'batman';

/* Actualiza hora de criação para todos os posts com número inferior a 5
feitos em 2019*/
UPDATE POST
SET Creation = '2020-01-01 00:00:00'
WHERE Num < 5 AND Year(Creation) = 2019;
```

Remoção de registos — forma básica

```
DELETE FROM
  <NOME DA TABELA>
WHERE
  <CONDIÇÃO>;
```

- Permite a remoção de registos de uma tabela indicada que verifiquem determinada condição.
- Na ausência da cláusula **WHERE todos os registos** são considerados para remoção.

Exemplos de remoções

```
/* Remove todos os registos da tabela FOLLOWER */
DELETE FROM FOLLOWER;

/* Remove seguidores do utilizador 1*/
DELETE FROM FOLLOWER
WHERE User = 1;

/* Remove registo para seguimento do utilizador 1 pelo utilizador 2*/
DELETE FROM FOLLOWER
WHERE User = 1 AND Follower = 2;
```

Funções e operadores

Funções e operadores

- Em SQL estão definidas funções e operadores que podemos usar em associação às instruções em SQL.

- Os principais operadores/funções permitem
  - Operações tradicionais como as de índole aritméticas, relacional ou lógica
  - Outras operações específicas sobre tipos de dados como sequências de caracteres ou dados temporais

- Referência para MySQL: MySQL Reference Manual - Functions and Operators

Alguns dos principais operadores

Operadores aritméticos	+ - * / % (os usuais)
Operadores relacionais	= (igualdade) <> (desigualdade) < <= > >= IS NULL IS NOT NULL
Operadores lógicos	AND / && OR / || NOT / !

Algumas funções sobre datas

- **NOW()**
  - Devolve a data e tempo actual (DATETIME)
- **YEAR(x) MONTH(x) DAY(x)**
  - Devolvem o ano, mês, ou dia de um valor DATE ou DATETIME
- **HOUR(x) MINUTE(x), SECOND(x)**
  - Análogo para horas, minutos e segundos para um valor DATETIME ou TIME
- **TIMESTAMPDIFF(ESCALA, A, B)**
  - Calcula diferença entre **B** e **A** em determinada escala (**ESCALA**=YEAR|MONTH|DAY|HOUR| MINUTE|SECOND|…)

Exemplos de operações sobre strings

- **LENGTH(S)**
  - Devolve tamanho de **S**
- **CONCAT(S1,S2, …)**
  - Devolve concatenação de strings **S1, S2, …**
- **REPLACE(S, A, B)**
  - Obtém resultado de substituir **A** por **B** em **S**
- **REVERSE(S)**
  - Obtém **S** invertida.
- **S [NOT] LIKE <EXP REGULAR>**
  - Verifica S face a expressão regular SQL.
  - **_** : um qualquer caracter; **%**: qualquer sequência de 0 ou mais caracteres

Modificadores de consultas

ORDER BY, LIMIT, e DISTINCT

Uso de ORDER BY

```
SELECT <Atributos>
FROM <TABELA>
…
ORDER BY <Criterio Ord. 1> [ASC|DESC],
                …,
                <Criterio Ord. k> [ASC|DESC]
```

- **ORDER BY** - especifica critério(s) de ordenação para resultados devolvidos por uma consulta.
  - **ASC**: ordem ascendente (valor por omissão).
  - **DESC**: ordem ascendente.

Uso de ORDER BY - exemplos

```
/* Sem critério de ordenação. */
SELECT Login, BirthDate FROM USER;
```

semedo	1985-06-02
maria	2005-11-17
pedro	1982-01-03
mendocas	2002-05-03
eva	1975-11-18
pedro2	1993-02-22
batman	1939-01-01
catwoman	1940-01-01
ziggy	1972-01-01
lady.z	1972-01-01

```
/* Ordena por login. */
SELECT Login, BirthDate FROM USER
ORDER BY Login;
```

batman	1939-01-01
catwoman	1940-01-01
eva	1975-11-18
lady.z	1972-01-01
maria	2005-11-17
mendocas	2002-05-03
pedro	1982-01-03
pedro2	1993-02-22
semedo	1985-06-02
ziggy	1972-01-01

```
/* Ordena por BirthDate de forma  decrescente */
SELECT Login, BirthDate FROM USER
ORDER BY BirthDate DESC;
```

maria	2005-11-17
mendocas	2002-05-03
pedro2	1993-02-22
semedo	1985-06-02
pedro	1982-01-03
eva	1975-11-18
ziggy	1972-01-01
lady.z	1972-01-01
catwoman	1940-01-01
batman	1939-01-01

ORDER BY — exemplos (cont.)

```
/* Ordena por dia de aniversário:
     (mês, dia de BirthDate),
     depois ano de BirthDate
      e finalmente Login.
 */
SELECT Login, BirthDate FROM USER
ORDER BY MONTH(BirthDate),
                DAY(BirthDate),
                YEAR(BirthDate),
                Login;
```

batman	**1939-01-01**
catwoman	**1940-01-01**
lady.z	**1972-01-01**
ziggy	**1972-01-01**
pedro	1982-**01-03**
pedro2	1993-**02-**22
mendocas	2002-**05-**03
semedo	1985-**06-**02
maria	2005-**11-17**
eva	1975-**11-18**

Uso de LIMIT

```
SELECT  …
FROM …
[WHERE …]
[ORDER BY ….]
LIMIT n;
```

- **LIMIT n** : limita número de resultados devolvidos por uma consulta a **n**
- Uso frequente em conjunção com **ORDER BY** para obter os primeiros **n** registos pelo critério de ordenação.

Uso de LIMIT — exemplos

```
SELECT Login, BirthDate FROM USER
LIMIT 5;
```

```
| semedo    | 1985-06-02 |
| maria     | 2005-11-17 |
| pedro     | 1982-01-03 |
| mendocas  | 2002-05-03 |
| eva       | 1975-11-18 |
```

```
SELECT Login, BirthDate FROM USER
ORDER BY BirthDate DESC
LIMIT 5;
```

```
| maria     | 2005-11-17 |
| mendocas  | 2002-05-03 |
| pedro2    | 1993-02-22 |
| semedo    | 1985-06-02 |
| pedro     | 1982-01-03 |
```

```
/* Dados do último POST */
SELECT Creation, Content FROM POST
ORDER BY Creation DESC
LIMIT 1;
```

```
+---------------------+-----------------------------------------------------+
| Creation            | Content                                             |
+---------------------+-----------------------------------------------------+
| 2020-02-18 17:00:30 | Joker is a busy man, he has no time for social networking. |
+---------------------+-----------------------------------------------------+
```

Uso de DISTINCT

SELECT DISTINCT <Campo 1>,..., <Campo n>
FROM …
…

- **DISTINCT**: filtra dados duplicados nos resultados devolvidos por uma consulta:

Uso de DISTINCT — exemplos

```
/* Obtém todos os autores que escreveram algum post. */
SELECT DISTINCT Author
FROM POST
ORDER BY Author;
```

Author
2
8
9
10

```
/* Obtém todos os utilizadores
   que têm algum seguidor por ordem decrescente. */
SELECT DISTINCT User
FROM FOLLOWER
ORDER BY User DESC;
```

User
11
10
9
8
7
5
4
3
2

Preservação / violação de restrições de integridade

Restrições de integridade

- Vamos a seguir ver exemplos práticos de violação de cada um dos tipos de restrições de integridade. Relembremos (dos "slides" de "O Modelo Relacional")
  - **Integridade de domínio:** o valor de um atributo faz parte do domínio do atributo.
  - **Integridade de entidade:** o valor da chave primária não pode ser NULL (sob pena de não conseguirmos identificar registos). É um caso especial de integridade de domínio.
  - **Integridade de chave:** dois registos da mesma tabela não podem ter valores iguais para uma chave, em particular para a chave primária.
  - **Integridade referencial:** um valor definido (≠ NULL) para um atributo que seja chave externa deve referir-se a uma chave primária da tabela a que a chave externa se refere.
- ... e ainda falar sobre algum suporte para manutenção de integridade referencial em SQL em complemento à definição de chaves primárias e externas — cláusulas **ON UPDATE** e **ON DELETE** para chaves externas.

Violação da integridade de domínio

```
INSERT INTO USER(Login, Joined, Name, BirthDate, Sex, Phone, Email)
VALUES('jsilva', '2020-03-20 00:00:00', 'José Silva',
       '1975-02-30', 'M', 223457500, 'xpto@gmail.com');
```

ERROR 1292 (22007): Incorrect date value: '1975-02-30' for column 'BirthDate' at row 1

```
INSERT INTO USER(Login, Joined, Name, BirthDate, Sex, Phone, Email)
VALUES('jsilva', '2020-03-20 00:00:00',
       NULL, '1975-02-28', 'M', 223457500, 'xpto@gmail.com');
```

ERROR 1048 (23000): Column 'Name' cannot be null

```
INSERT INTO USER(Login, Joined, Name, BirthDate, Sex, Phone, Email)
VALUES('jsilva', '2020-03-20 00:00:00', 'José Silva', '1975-02-28', 'M',
       'PT-223457500', 'xpto@gmail.com');
```

ERROR 1366 (HY000): Incorrect integer value: 'PT-223457500' for column 'Phone' at row 1

```
/* Definição da tabela */
CREATE TABLE USER
(
  Num       PRIMARY KEY AUTO_INCREMENT,
  Login     UNIQUE VARCHAR(16) NOT NULL,
  Joined    DATE NOT NULL,
  Name      VARCHAR(128) NOT NULL,
  BirthDate DATE NOT NULL,
  Sex       ENUM('M', 'F') NOT NULL,
  Phone     INT DEFAULT NULL,
  Email     VARCHAR(64) NOT NULL
);
```

Violação da integridade de chave

```
INSERT INTO USER(Num, Login, Joined, Name, BirthDate, Sex, Phone,
Email)
VALUES(1, 'jsilva', '2020-03-20 00:00:00', 'José Silva',
        '1975-02-28', 'M', 223457500, 'xpto@gmail.com');
```

ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

```
/* Definição da tabela */
CREATE TABLE USER
(
  Num       PRIMARY KEY AUTO_INCREMENT,
  Login     UNIQUE VARCHAR(16) NOT NULL,
  Joined    DATE NOT NULL,
  Name      VARCHAR(128) NOT NULL,
  BirthDate DATE NOT NULL,
  Sex       ENUM('M', 'F') NOT NULL,
  Phone     INT DEFAULT NULL,
  Email     VARCHAR(64) NOT NULL
);
```

```
INSERT INTO USER(Login, Joined, Name, BirthDate, Sex, Phone, Email)
VALUES('batman', '2020-03-20 00:00:00', 'José Batman',
        '1975-02-28', 'M', 223457500, 'xpto@gmail.com');
```

ERROR 1062 (23000): Duplicate entry 'batman' for key 'Login'

```
UPDATE USER SET Login = 'batman' WHERE Login='joao.pinto';
```

ERROR 1062 (23000): Duplicate entry 'batman' for key 'Login'

Violação da integridade referencial

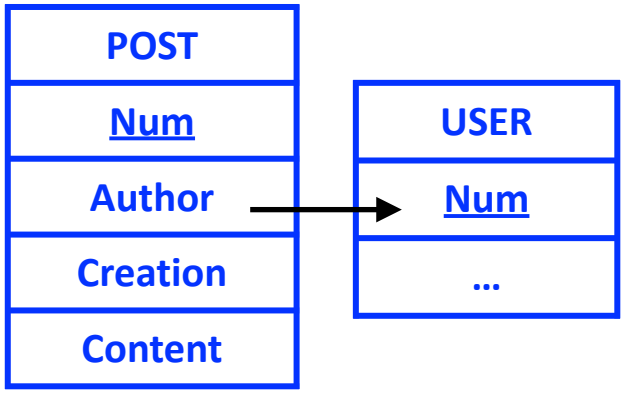
```
/* User 9999 does not exist. */
INSERT INTO POST(Author,Creation,Content)
VALUES(9999, NOW(), 'Hello World!');
```

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`guest`.`post`, CONSTRAINT `post_ibfk_1` FOREIGN KEY (`Author`) REFERENCES `USER` (`Num`))

```
/* Definição da tabela */
CREATE TABLE POST
(
  Num INT PRIMARY KEY AUTO_INCREMENT,
  Author INT NOT NULL,
  Creation DATETIME NOT NULL,
  Content TEXT NOT NULL,
  FOREIGN KEY(Author) REFERENCES USER(Num)
);
```

```
/* User 1 exists and is the author of some posts */
DELETE FROM USER WHERE Num = 1;
```

ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`guest`.`post`, CONSTRAINT `post_ibfk_1` FOREIGN KEY (`Author`) REFERENCES `USER` (`Num`))

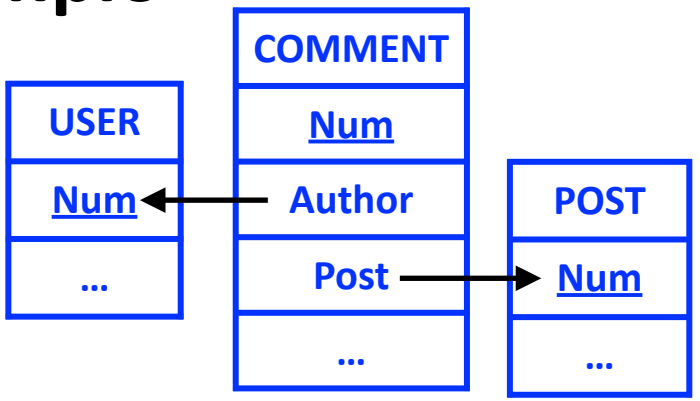


Uso de ON UPDATE e ON DELETE — forma geral

```
CREATE TABLE <NOME DA TABELA>
(
    …
    FOREIGN KEY(<ATRIBUTOS QUE FORMAM UMA CHAVE EXTERNA>)
    REFERENCES <NOME_DE_OUTRA_TABELA>(<ATRIBUTOS CHAVE>)
    [ON UPDATE <AÇÃO>]
    [ON DELETE <ACÇÃO>],
    …
);
```

- Para **chaves externas** podemos especificar o que deve acontecer quando a entrada correspondente à chave primária referenciada é actualizada (**ON UPDATE**) ou apagada (**ON DELETE**).

Uso de ON DELETE CASCADE — exemplo

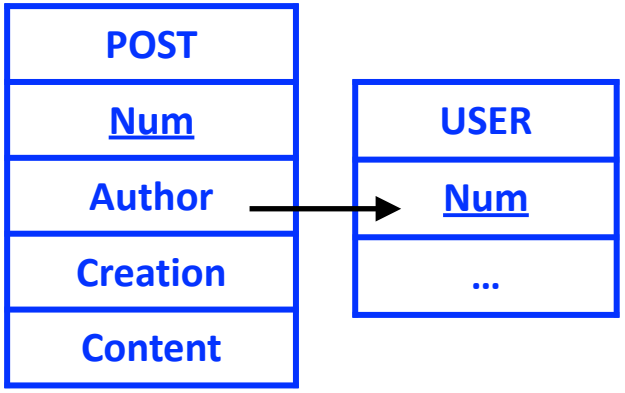



```
CREATE TABLE COMMENT
( Num INT PRIMARY KEY AUTO_INCREMENT,
  Post INT NOT NULL,
  Author INT NOT NULL,
  ...
  FOREIGN KEY(Post) REFERENCES POST(Num) ON DELETE CASCADE,
  FOREIGN KEY(Author) REFERENCES USER(Num) ON DELETE CASCADE
);
```

- **ON DELETE CASCADE**: apaga entradas em outras tabelas que referenciem a chave primária do registo apagado.
- Na BD exemplo esta restrição existe para várias chaves externas references a **USER(Num)** — e também **POST(Num)**— como é o caso da tabela **COMMENT** mas não p/a tabela **POST.**

Uso de ON DELETE CASCADE — aplicação a POST

```
CREATE TABLE POST
(
  Num INT PRIMARY KEY AUTO_INCREMENT,
  Author INT NOT NULL,
  …
  FOREIGN KEY(Author) REFERENCES USER(Num)
  ON DELETE CASCADE
);
```



- Aplicando **ON DELETE CASCADE** à chave externa já conseguimos remover o utilizador 1 (anteriormente a operação falhava) e todos os "posts", comentários, "likes", … que se referem ao utilizador 1 (as outras tabelas já estavam configuradas desta forma).

```
mysql> DELETE FROM USER WHERE Num=1;
Query OK, 1 row affected (0.00 sec)
mysql> SELECT * FROM POST WHERE Author=1;
Empty set (0.00 sec)
```

Uso de ON DELETE CASCADE (cont.)

```
mysql> SELECT Num,Post,Author FROM COMMENT ORDER BY Num;
+-----+------+--------+
| Num | Post | Author |
+-----+------+--------+
|   1 |    1 |      2 |
|   2 |    1 |      3 |
|   3 |    1 |      1 |
|   4 |    8 |      2 |
|   5 |    8 |      9 |
|   6 |   12 |     11 |
+-----+------+--------+
…
mysql> DELETE FROM USER WHERE Num = 1;
…
mysql> SELECT Num,Post,Author FROM COMMENT ORDER BY Num;
+-----+------+--------+
| Num | Post | Author |
+-----+------+--------+
|   4 |    8 |      2 |
|   5 |    8 |      9 |
|   6 |   12 |     11 |
+-----+------+--------+
3 rows in set (0.00 sec)
```

Comentários ao post nº 1 (numerados de 1 a 3) são todos removidos porque o autor do post nº 1 é também utilizador nº1. Portanto são removidos utilizador nº 1, post nº 1, e todos os comentários relacionados com o utilizador e/ou post em causa.

Uso de ON UPDATE CASCADE

```
CREATE TABLE POST
(
  Num INT PRIMARY KEY AUTO_INCREMENT,
  Author INT NOT NULL,
  …
  FOREIGN KEY(Author) REFERENCES USER(Num)
  ON DELETE CASCADE ON UPDATE CASCADE
);
```

- **ON UPDATE CASCADE**: análogo a **ON DELETE CASCADE** mas para actualizações de uma chave propaga valor da nova chave em chaves externas.
- Vamos exemplificar a seguir o efeito da mesma.

Uso de ON UPDATE CASCADE (cont.)

```
mysql> SELECT Num,Post,Author FROM COMMENT ORDER BY Num;
+-----+------+--------+
| Num | Post | Author |
+-----+------+--------+
|   1 |    1 |      2 |
|   2 |    1 |      3 |
|   3 |    1 |      1 |
|   4 |    8 |      2 |
|   5 |    8 |      9 |
|   6 |   12 |     11 |
+-----+------+--------+
…
mysql> UPDATE USER SET Num = 1001 WHERE Num = 1;
…
mysql> UPDATE POST SET Num = 9001 WHERE Num = 1;
…
mysql> SELECT Num,Post,Author FROM COMMENT ORDER BY Num;
+-----+------+--------+
| Num | Post | Author |
+-----+------+--------+
|   1 | 9001 |      2 |
|   2 | 9001 |      3 |
|   3 | 9001 |   1001 |
|   4 |    8 |      2 |
|   5 |    8 |      9 |
|   6 |   12 |     11 |
+-----+------+--------+
```

Uso de ON UPDATE e ON DELETE — outras opções

- **ON UPDATE SET NULL / ON DELETE SET NULL**
  - Substitui valor da chave externa por **NULL.**
  - Válido desde que o domínio da chave externa admita o valor **NULL.**
- **ON UPDATE SET DEFAULT / ON DELETE SET DEFAULT**
  - Substitui valor da chave externa por valor **DEFAULT.**
  - Válido desde que o atributo da chave externa tenha associado um valor **DEFAULT.**
- **ON UPDATE RESTRICT / ON DELETE RESTRICT - a definição por omissão!**
  - **Operação de actualização / remoção é rejeitada.**

Evolução de esquemas
— uso de ALTER TABLE —

Evolução de esquemas

- **Problema geral**:
  - O esquema de uma BD tem normalmente necessidade de mudar ao longo do tempo.
  - Por exemplo a nossa "imitação" de rede social poderia "crescer" por forma a termos grupos de utilizadores, tipos de "like", definições de privacidade, etc.
  - Mudanças ao esquema: não devem levar à perda de dados existentes (a não ser que esse seja o objectivo) ou a perda sua consistência, e idealmente (embora nem sempre possível) sem tornar indisponível o acesso a utilizadores.
- Este problema é complexo. Neste contexto damos apenas uma ideia de algumas alterações/evoluções simples de um esquemas usando o comando **ALTER TABLE**.

A instrução ALTER TABLE

Forma geral:

ALTER TABLE <NOME DA TABELA> <ACÇÃO> <OPÇÕES>;

- Permite alterar o esquema ou outras propriedades de uma tabela. Entre outros usos, a instrução permite adicionar / modificar remover / rever campos de tabelas.
- Referência completa → Documentação MySQL

1º exemplo — alteração de tablea

```
/* Adiciona campo Country  à tabela USER,
  com valor NULL por omissão.*/
ALTER TABLE USER
ADD COLUMN  Country VARCHAR(32) DEFAULT NULL;

/* Remove campo Email  da tabela USER. */
ALTER TABLE USER
DROP COLUMN Email;

/* Muda tipo do campo Name
   (redefine tamanho máximo de 128 para 256). */
ALTER TABLE USER
MODIFY Name VARCHAR(256) NOT NULL;
```

```
mysql> desc USER;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| Num      | int(11)      | NO   | PRI | NULL    | auto_increment |
| Login    | varchar(16)  | NO   | UNI | NULL    |                |
| Joined   | date         | NO   |     | NULL    |                |
| Name     | varchar(128) | NO   |     | NULL    |                |
| BirthDate| date         | NO   |     | NULL    |                |
| Sex      | enum('M','F')| NO   |     | NULL    |                |
| Phone    | int(11)      | YES  |     | NULL    |                |
| Email    | varchar(64)  | NO   |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

```
mysql> desc USER;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| Num      | int(11)      | NO   | PRI | NULL    | auto_increment |
| Login    | varchar(16)  | NO   | UNI | NULL    |                |
| Joined   | date         | NO   |     | NULL    |                |
| Name     | varchar(256) | NO   |     | NULL    |                |
| BirthDate| date         | NO   |     | NULL    |                |
| Sex      | enum('M','F')| NO   |     | NULL    |                |
| Phone    | int(11)      | YES  |     | NULL    |                |
| Country  | varchar(32)  | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
```

```
mysql> select * FROM USER;
+-----+------------+------------+---------------+------------+-----+-----------+-------------------------+
| Num | Login      | Joined     | Name          | BirthDate  | Sex | Phone     | Email                   |
+-----+------------+------------+---------------+------------+-----+-----------+-------------------------+
|   1 | joao.pinto | 2019-12-01 | João Pinto    | 1976-12-19 | M   |      NULL | azulibranco@fcp.pt      |
|   2 | semedo     | 2019-12-02 | Carlos Semedo | 1985-06-02 | M   | 223774327 | semedo@xpto.com         |
|   3 | maria      | 2019-12-02 | Maria Silva   | 2005-11-17 | F   | 939939939 | maria@xyz.pt            |
|   4 | pedro      | 2019-12-10 | Pedro Costa   | 1982-01-03 | M   |      NULL | pc12345@xpto.com        |
|   5 | mendocas   | 2019-12-11 | Filipa Mendes | 2002-05-03 | F   |      NULL | filipa.mendes@gmail.com |
 … etc …
```

```
mysql> select * FROM USER;
+-----+------------+------------+---------------+------------+-----+-----------+---------+
| Num | Login      | Joined     | Name          | BirthDate  | Sex | Phone     | Country |
+-----+------------+------------+---------------+------------+-----+-----------+---------+
|   1 | joao.pinto | 2019-12-01 | João Pinto    | 1976-12-19 | M   |      NULL | NULL    |
|   2 | semedo     | 2019-12-02 | Carlos Semedo | 1985-06-02 | M   | 223774327 | NULL    |
|   3 | maria      | 2019-12-02 | Maria Silva   | 2005-11-17 | F   | 939939939 | NULL    |
|   4 | pedro      | 2019-12-10 | Pedro Costa   | 1982-01-03 | M   |      NULL | NULL    |
|   5 | mendocas   | 2019-12-11 | Filipa Mendes | 2002-05-03 | F   |      NULL | NULL    |
… etc …
```

— Dados não foram perdidos, além dos de email (o que era pretendido).
— “Rede social” permite agora a definição do país dos utilizadores quando desejável (NULL por omissão; subentende-se que não vamos assumir a nacionalidade dos utilizadores já existentes.).
— Nomes de utilizadores passam a poder ter 256 caracteres.

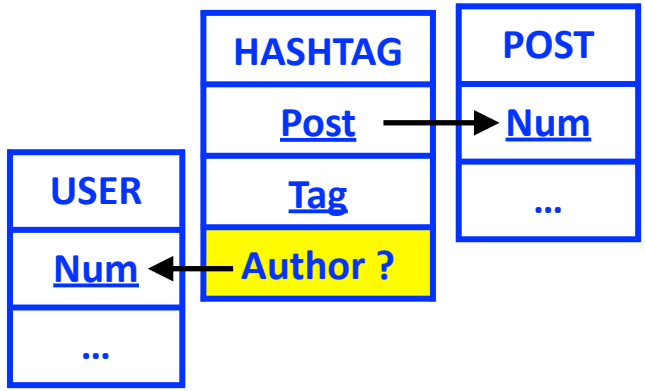
2º exemplo: alteração de chaves

```
/* Obriga a que nº de telefone seja único por utilizador. */
ALTER TABLE USER
ADD UNIQUE(Phone);

/* Adiciona campo para autor de uma "hashtag" e torna-o uma chave
externa para USER(Num). */

ALTER TABLE HASHTAG
ADD  COLUMN Author INT DEFAULT NULL;

ALTER TABLE HASHTAG
ADD FOREIGN KEY(Author) REFERENCES USER(Num);
```



3º exemplo — alterações inválidas

```
/* Dados anteriores não são de tipo compatível. */
ALTER TABLE USER MODIFY COLUMN Name INT NOT NULL;
ERROR 1366 (HY000): Incorrect integer value: 'João Pinto' for column 'Nam
at row 1

/* Só pode haver uma chave primária. */
ALTER TABLE USER ADD PRIMARY KEY(Login);
ERROR 1068 (42000): Multiple primary key defined

/* Quebra integridade referencial */
ALTER TABLE USER  DROP COLUMN(Num);

ERROR 1829 (HY000): Cannot drop column 'Num': needed in a foreign key
constraint 'comment_ibfk_2' of table 'guest.comment'
```