第7章 MySQL表定义与完整性约束控制

人工智能学院

古 晶

西安电子科技大学

第7章 MySQL表定义与完整性约束控制

- 7.1 表的基本概念
- 7.2 数据类型
- 7.3 运算符
- 7.4 表的操作
- 7.5 MySQL约束控制
- 7.6 知识点小结
- 本章实验

第7章 MySQL表定义与完整性约束控制

- **表**是数据库中存储数据的基本单位，它由一个或多个字段组成，每个字段需要有对应的**数据类型**。
- MySQL数据库中表的**管理**，包括表的作用、类型、构成、创建、删除和修改等。
- 本章先讲述了表的基本概念，MySQL支持的**数据类型**和**运算符**等一些基础，接着又讲述了表的基本操作，有创建、查看、修改、复制、删除。最后讲述了MySQL的约束控制，如何定义和修改字段的约束条件。

第7章 MySQL表定义与完整性约束控制

- 7.1 表的基本概念
- 7.2 数据类型
- 7.3 运算符
- 7.4 表的操作
- 7.5 MySQL约束控制
- 7.6 知识点小结
- 本章实验

7.1 表的基本概念

- 1.数据库与表之间的关系：数据库是由各种数据表组成的，数据表是数据库中最重要的对象，也是其他对象的基础。
- 2.数据表是用来存储和操作数据的**逻辑结构。**
- 3.数据表是包含了特定**实体**类型数据。
- 4.数据表由**行**（row）和**列**（column）构成的二维网络，其中列是表数据的描述，行是表数据的实例。
- 5.数据表一定先有表结构，再有数据（没有数据，称为空表）。
- 6.数据表至少有一列，可以没有行或者多行。
- 7.数据表名称要求唯一，而且不要包含特殊字符。
- 8.表的操作包括**创建**新表、**修改**表和**删除**表，这些操作都是数据库管理中最基本，也是最重要的操作。

7.1 表的基本概念

1.建表**原则**

（1）应按照一定原则对信息进行分类。

（2）对表进行**规范化设计**，以消除表中存在的冗余，保证一个表只围绕一个主题，并使表容易维护。

2.数据库表的**信息存储**分类原则

（1）每个表应该只包含关于一个主题的信息。

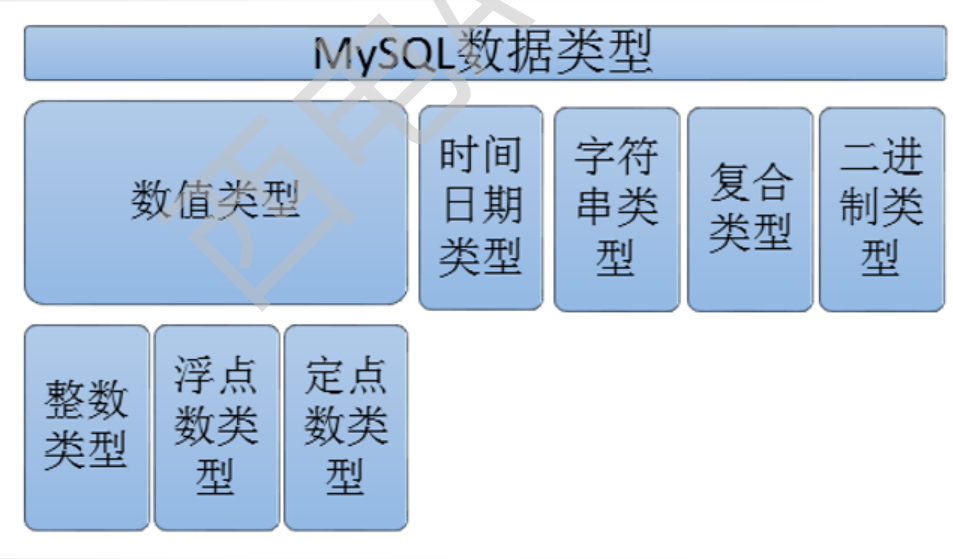
（2）**表中不应包含重复信息。**

第7章 MySQL表定义与完整性约束控制

- 7.1 表的基本概念
- **7.2 数据类型**
- 7.3 运算符
- 7.4 表的操作
- 7.5 MySQL约束控制
- 7.6 知识点小结
- 本章实验

7.2 数据类型

- 为每张表的每个字段**选择合适的数据类型**是数据库设计过程中一个重要的步骤。
- 好处：可以有效地节省数据库的**存储空间**，包括内存和外存，同时也可以提升数据的**计算性能**，节省数据的**检索**时间。
- 常用的数据类型如下：



数值类型

- MySQL支持所有的ANSI/ISO SQL 92数字类型（ANSI，American National Standards Institute，美国国家标准局）。
- 数字分为整数和小数。
  - **整数**：用整数类型表示。
    - INTEGER、SMALLINT、TINYINT、MEDIUMINT和BIGINT
    - **INT**与INTEGER两个整数类型是**同名词**，可以互换。
  - **小数**：
    - **浮点数**类型：单精度浮点数**FLOAT**类型和双精度浮点数**DOUBLE**类型
    - **定点数**类型：DECIMAL类型，**DEC**和DECIMAL这两个定点数类型是同名词。
- 取值范围可以通过：help、?、\h三种方式之一查看，比如help tinyint。

数值类型

- MySQL **浮点型**和**定点型**可以用类型名称后加（M，D）来表示，M表示该值的总共长度，D表示小数点后面的长度，M和D又称为精度和标度，如float(7,4)的 可显示为-999.9999，MySQL保存值时进行四舍五入，如果插入999.00009，则结果为999.0001。
- **所属类型**：
  - ✓ float数值类型用于表示单精度浮点数值
  - ✓ double数值类型用于表示双精度浮点数值
  - ✓ float和double都是浮点型
  - ✓ decimal是定点型
- **不指定精度时显示方式**：
  - ✓ FLOAT和DOUBLE在不指定精度时，默认会按照实际的精度来显示，而DECIMAL在不指定精度时，默认整数为10，小数为0。

数值类型

- 建表时**数字类型的选择**应遵循的原则：

1)选择**最小**的可用类型，如果改字段的值不会超过127，则使用TINYINT比INT效果好。

2)对于完全都是数字的，即**无小数点**时，可以选择整数类型，比如年龄。

3)浮点类型用于可能具有的**小数部分**的数，比如学生成绩。

4)在需要表示**金额**等**货币类型**时优先选择DECIMAL数据类型。

日期时间类型

- MySQL主要支持的**日期**类型：
  （1）DATE表示日期，默认格式为‘YYYY-MM-DD’；
  （2）TIME表示时间，默认格式为‘HH:ii:ss’；
  （3）YEAR表示年份；
  （4）DATATIME与TIMESTAMP是日期和时间的混合类型，默认格式为‘YYYY-MM-DD HH:ii:SS’。
- 从形式上来说，MySQL日期类型的表示方法与**字符串**的表示方法相同（使用单引号括起来）；
- 本质上，MySQL日期类型的数据是一个**数值类型**，可以参与简单的加、减运算。

字符串类型

口字符串类型的数据分为：

- **普通的文本字符串类型**（CHAR和VARCHAR）
  - CHAR类型的长度被固定为创建表所声明的长度，取值在0～255之间；超过存储范围，存储不会成功。
  - VARCHAR类型的值是变长的字符串，取值和CHAR一样。
  - 字符串末尾有空格时，CHAR类型的VALUES会去掉空格，而VARCHAR不会掉空格（ ）；字符串前的空格都不会去掉。
  - 查看字符串在UTF8编码中的长度为3；（LENGTH（ ）函数，后边会讲此函数的应用）
  - 其中，**CHAR**定长字符串，占用空间大，速度快；**VARCHAR**变长字符串，占用空间小，速度慢。

字符串类型

口字符串类型的数据分为：

- **普通的文本字符串类型**（CHAR和VARCHAR）
  - CHAR类型的长度被固定为创建表所声明的长度，取值在0～255之间；超过存储范围，存储不会成功。
  - VARCHAR类型的值是变长的字符串，取值和CHAR一样。
  - 其中，CHAR定长字符串,占用空间大，速度快；
    VARCHAR变长字符串，占用空间小，速度慢
- **文本类型：**TEXT
  - 保存大量文本，只能存储文本字符；**TEXT**不能有默认值
- CHAR、VACHAR、TEXT 3个数据类型存储和检索数据的方式都不一样，数据检索的效率CHAR>VACHAR>TEXT，以后优化数据时，能不用text就不用text。

字符串类型

特殊类型：枚举类型（ENUM）：

- 枚举类型的值是以列表形式显示，多个值之间以**逗号**来分隔。
- 插入值时**只允许从一个集合中取得某一个值**，保存时并不保存它的值，每个值都有一个**区号**，保存的也是这个区号。
- 每个枚举值都有一个索引：列出的元素被分配从1开始的索引值。1 or 2, 取决于枚举成员的数量。空字符串作为错误值的索引值为0。可以使用select语句找出那些被指定无效枚举值的数据行。NULL的索引为NULL。**这里的索引只是指出枚举表里该元素的位置，和表索引不同。**
- 最多可以有65535个不同的元素值（实际限制小于3000）。
- 枚举值不能是0或空字符串（虽然存在特殊情况）
- enum在底层的存储方式是以整型进行存储的，比如这样的字段sex enum('male' , 'female' , 'both' , 'unknow')在查询时where sex='male'和where sex=1是等效的

字符串类型

- **特殊类型：集合类型**（SET）：
  - **SET**是一个字符串对象，可以有零或多个值，其值来自表创建时规定的允许的一列值。
  - 指定包括多个SET成员的SET列值时各成员之间用逗号（‘,’）间隔开。
  - 插入多个数据时，用逗号分隔。
  - MySQL保存值是以二进制的形式保存的。
  - 所占字节：**1|2|3|4|8**，取决于集合成员的数量，最大64个。

字符串类型

- 在**创建表**时，使用**字符串类型**时应遵循以下原则：

1)从**速度**方面考虑，要选择固定的列，可以使用**CHAR**类型。

2)要**节省空间**，使用动态的列，可以使用**VARCHAR**类型。

3)要将列中的内容**限制在一种选择**，可以使用**ENUM**类型。

4)允许在一个列中**有多于一个的条目**，可以使用**SET**类型。

5)如果要搜索的内容**不区分大小写**，可以使用**TEXT**类型。

6)如果要搜索的内容**区分大小写**，可以使用**BLOB**类型。

二进制类型

- MySQL主要支持7种二进制类型：binary、varbinary、bit、tinyblob、blob、mediumblob和longblob。
- 二进制类型的字段主要用于存储由‘0’和‘1’组成的字符串，从某种意义上讲，二进制类型的数据是一种特殊格式的**字符串**。
- **二进制类型与字符串类型的区别**：
  - 字符串类型的数据按**字符**为单位进行存储，因此存在多种字符集、多种字符序；
  - 除了bit数据类型按**位**为单位进行存储，其他二进制类型的数据按**字节**为单位进行存储，仅存在二进制字符集binary。

选择合适的数据类型

□ 选择数据类型的原则：

(1)在符合应用要求（取值范围、精度）的前提下，尽量使用“**短**”数据类型。

(2)数据类型越**简单**越好。

(3)尽量采用**精确小数类型**（例如decimal），而不采用浮点数类型。

(4)在MySQL中，应该用**内置的**日期和时间数据类型，而不是用字符串来存储日期和时间。

(5)尽量**避免**NULL字段，建议将字段指定为NOT NULL约束。

选择合适的数据类型

- 1.**整数**类型和**浮点数**类型

如果要表示小数只能用浮点数类型，整数类型不能表示小数。浮点类型DOUBLE精度比FLOAT 类型高，如果需要精确10位以上，就应该选择DOUBLE类型。

- 2.**浮点数**类型和**定点数**类型

对于精度要求较高的时候需要使用定点数存，因为定点数内部是以字符串形式存储的。

- 3.**日期**和**时间**类型

YEAR只保存年份，占用空间小。其它和日期时间有关的可以通过整型保存时间，方便计算。

选择合适的数据类型

- 4.**CHAR类型**和**VARCHAR类型**和**TEXT类型**
  - ✓ CHAR定长字符串，占用空间大，速度快。VARCHAR变长字符串，占用空间小，速度慢。 TEXT类型是一种特殊的字符串类型。只能保存字符数据，而且不能有默认值。它们3个存储和检索数据的方式都不一样。
  - ✓ **数据检索的效率 CHAR>VAARCHAR>TEXT**。
  - ✓ CHAR在保存的时候，后面会用空格填充到指定的长度，在检索的时候后面的空格去掉VARCHAR在保存的时候，不进行填充。当值保存和检索时尾部的空格人保留。


第7章 MySQL表定义与完整性约束控制

- 7.1 表的基本概念
- 7.2 数据类型
- **7.3 运算符**
- 7.4 表的操作
- 7.5 MySQL约束控制
- 7.6 知识点小结
- 本章实验

7.3 运算符

- 运算符是用来连接表达式中各个操作数据的符号，其作用是用来指明对操作数所进行的运算。
- MySQL数据库支持运算符的使用，通过运算符可以更加灵活的操作数据表中的数据。
- MySQL主要支持**算术运算符**、**比较运算符**、**逻辑运算符**和位运算符四种类型。

算术运算符

- MySQL数据库支持的**算术运算符**包括**加**、**减**、**乘**、**除**和**取余**运算。他们是最常用的、最简单的一类运算符。

运算符	作用
+	加法，返回相加后的值
-	减法 ，返回相减后的值
*	乘法，返回相乘后的值
/，DIV	取整，返回相除后的商
%,MOD	取余，返回相除后的余数

比较运算符

- MySQL数据库允许用户对表达式的左边操作数和右边操作数进行**比较**，比较结果**为真返回1**，**为假返回0**，**不确定返回NULL**。

运算符	作用
=	等于
<>或!=	不等于
<=>	NULL 安全的等于
<	小于
<=	小于等于
>	大于
>=	大于等于
BETWEEN min AND max	在 min 和 max 之间
IN (value1，value2，……)	存在于集合(value1，value2，……)中
IS NULL	为 NULL
IS NOT NULL	不为 NULL
LIKE	通配符匹配
REGEXP 或 RLIKE	正则表达式匹配

逻辑运算符

- **逻辑运算符**，也称为布尔运算符，判断表达式的真假。

运算符	作用
NOT 或 !	逻辑非
AND 或 &	逻辑与
OR 或 ||	逻辑或
XOR	逻辑异或

- 7.1 表的基本概念
- 7.2 数据类型
- 7.3 运算符
- **7.4 表的操作**
- 7.5 MySQL约束控制
- 7.6 知识点小结
- 本章实验

表的操作

- 表是数据库中最为重要的数据库对象。
- 创建表前，需要根据数据库设计的结果确定**表名**、**字段名**及**数据类型**、**约束**等信息。
- 还要为每张表选择一个合适的**存储引擎**。
- 表的操作：
  - **创建**表，**查看**表，**修改**表，**复制**表，**删除**表
- 表管理中的注意事项。

创建表

- 在同一个数据库中，**表名不能有重名**。
- 创建数据表可使用CREATE TABLE命令。
- **语法格式**：

```
CREATE  TABLE [IF NOT EXISTS] 表名(
字段名称 字段类型 [完整性约束条件]
… )
ENGINE=引擎名称 CHARSET='编码方式' ;
```

- **说明**：

1.语法格式中“[]”表示可选的。

创建表

口 说明：

2. IF NOT EXISTS：如果数据库中已经存在某个表，再来创建一个同名的表，这时会出现错误，为了避免错误信息，可以在创建表的前面加上这个判断，只有该表目前不存在时才执行CREATE TABLE操作。

3.表名：要创建的表名。

4.字段的定义。包括**指定字段名、数据类型、是否允许空值，指定默认值、主键约束、唯一性约束、注释字段名、是否为外键，以及字段类型的属性**等。

创建表

- **说明：**
  - 完整性约束条件：
    - ✓ **PRIMARY KEY**主键（唯一来标识的，每一个表都一个，自动非空）
    - ✓ **AUTO_INCREMENT**自增长
    - ✓ **FOREIGN KEY**外键
    - ✓ **NOT NULL**非空
    - ✓ **UNIQUE KEY**唯一
    - ✓ **DEFAULT**默认值

创建表

- 设置表的**存储引擎**、**默认字符**集以及**压缩类型**：

  (1)向create table语句末尾添加engine选项，即设置该表的存储引擎

  语法格式：**Engine**=存储引擎类型

  (2) 向create table语句末尾添加default charset选项，即设置该表的字符集

  语法格式：**Default charset**=字符集类型

  (3) 如果希望压缩索引中的关键字，使索引关键字占用更少的存储空间，可以通过设置pack_keys选项实现（注意：该选项仅对MyISAM存储引擎的表有效）

  语法格式：**Pack_keys**=压缩类型

创建表的示例

例：创建一个员工表，其中，字段要求：编号（id）、用户名（username）、年龄（age）、性别（sex）、邮箱（email）、地址（addr）、生日（birth）、薪水（salary）、电话（tel）、是否结婚（married）。

Field	Type	Null	Key	Default	Extra
id	smallint(6)	YES		NULL	
username	varchar(20)	YES		NULL	
age	tinyint(4)	YES		NULL	
sex	enum('男','女','保密')	YES		NULL	
email	varchar(50)	YES		NULL	
addr	varchar(200)	YES		NULL	
birth	year(4)	YES		NULL	
salary	float(8,2)	YES		NULL	
tel	int(11)	YES		NULL	
married	tinyint(1)	YES		NULL	

创建表的示例

例：创建一个员工表，其中，字段要求：编号（id）、用户名（username）、年龄（age）、性别（sex）、邮箱（email）、地址（addr）、生日（birth）、薪水（salary）、电话（tel）、是否结婚（married）。

其中，COMMENT为表字段的注释
注：
在SQL语句中注释可以通过“#”和“--”两种符号进行。

```
CREATE TABLE IF NOT EXISTS
`employee`(
    id SMALLINT,
    username VARCHAR(20),
    age TINYINT,
    sex ENUM('男','女','保密'),
    email VARCHAR(50),
    addr VARCHAR(200),
    birth YEAR,
    salary FLOAT(8,2),
    tel INT,
    married TINYINT(1) COMMENT '0
代表未结婚，1代表结婚'
    )ENGINE=INNODB
CHARSET=UTF8;
```

创建表小练习

练一练：创建两个表：
1.新闻分类表，其中字段要求：编号、分类名称、分类描述。
2.新闻表，其中字段要求：编号、新闻标题、内容、发布时间、点击量、是否置顶、所属分类、发布人。

```
CREATE TABLE IF NOT EXISTS cms_cate(
    id TINYINT,
    cateName VARCHAR(50),
    cateDesc VARCHAR(200)
    )ENGINE=MyISAM CHARSET=UTF8;

CREATE TABLE IF NOT EXISTS cms_news(
    id INT,
    title VARCHAR(50),
    content TEXT,
    pubTime INT,
    clickNum INT,
    isTop TINYINT(1) COMMENT '0代表不置顶，
1代表置顶 ',
    cId tinyint
    );
```

查看表

1)显示表的**名称**

可以使用SHOW TABLES语句来显示指定数据库中存放的所有表名

语法格式： **SHOW TABLES**；

2)显示表的**结构**

查看表结构有简单查询和详细查询，可使用DESCRIBE/DESC语句和SHOW CREATE TABLE语句

语法格式：

- DESCRIBE 表名;
- DESC 表名;
- **SHOW CREATE TABLE 表名**;

示例：查看emploee的表结构。

```
DESCRIBE employee;
DESC employee;
SHOW CREATE TABLE employee;
```

修改表

ALTER TABLE用于**更改原有的结构**。**语法格式：**

```
ALTER [ IGNORE] TABLE table_name
      alter_specification [, alter_specification]
ADD [COLUMN] column_definition[ FIRST | AFTER col_name]       //添加字段
| ALTER [COLUMN] col_name {SET DEFAULT LITERAL | DROP DEFAULT} //修改字段
| CHANGE [COLUMN] old_col_name column_definition [FIRST| AFTER col_name] //重命名字段
| MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]//修改字段
| DROP [COLUMN] col_name                                     //删除列
| RENAME [AS| TO] new_table_name                             //对表重命名
| ORDER BY col_name                                         //按字段排序
| CONVERT TO CHARACTER SET character_name [COLLATE collation_name] //将字段集转化为二进制
| [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name    //修改字符集
```

修改表—重命名

- **方法一：**语法格式： **ALTER TABLE 表名 RENAME [TO|AS] 新表名；**

  注：TO 或AS都可以，也以省略掉

  例如：

  - ✓ ALTER TABLE employee RENAME TO reg_user;
  - ✓ ALTER TABLE reg_user RENAME AS employee ;
  - ✓ ALTER TABLE employee RENAME reg_user;

- **方法二：**语法格式：**RENAME TABLE 表名 TO 新表名；**

  注：这里面的TO不可以省略

  例如：

  RENAME TABLE reg_user TO employee ;

- 每一次的更名操作，可以通过**SHOW TABLES;**命令查看

修改表—添加字段

- **语法格式：ALTER TABLE tbl_name ADD 字段名称 字段类型 [完整性约束条件] [FIRST|AFTER 字段名称]；**

✓ **在employee 表中添加一个身份证号码的字段。**

```
ALTER TABLE employee ADD card CHAR(18);
```

✓ **添加带上完整性约束条件的记录，默认添加在最后面。**

```
ALTER TABLE employee ADD testcol1 VARCHAR(100) NOT NULL UNIQUE;
```

表结构的修改可以通过**DESC 表名;**查看

✓ **来指定子段的添加位置。**

```
ALTER TABLE employee ADD testcol3 INT NOT NULL DEFAULT 100 AFTER username;
```

✓ **在一个表中，一次添加多个字段。**

```
ALTER TABLE employee
    ADD testcol4 INT NOT NULL DEFAULT 123 AFTER password,
    ADD testctestcol6 ol5 FLOAT(6,2) FIRST,
    ADD SET('A','B','C');
```

修改表—删除字段

□ 语法格式：ALTER TABLE tbl_name DROP 字段名称；

□示例

✓ 删除testcol1字段：

```
ALTER TABLE vipvipuser10 DROP testcol1;
```

✓ 一次删除多个字段：

```
ALTER TABLE vipvipuser10
    DROP testcol2,DROP testcol3,DROP testcol4,DROP testcol5,
DROP testcol6;
```

✓ 添加test字段删除addr字段：

```
ALTER TABLE vipvipuser10
    ADD testcol INT UNSIGNED NOT NULL DEFAULT 10 AFTER sex,
    DROP addr;
```

修改表—修改字段属性

□ 语法格式：ALTER TABLE tbl_name MODIFY 字段名称 字段类型 [完整性约束条件] [FIRST|AFTER 字段名称]；

□示例

✓将email VARCHAR(200)，先选中字段在进行修改

ALTER TABLE vipuser10 MODIFY email VARCHAR(200);

✓将card字段移动到testcol字段之后

ALTER TABLE vipuser10 MODIFY card CHAR(18) AFTER testcol;

✓将testcol字段修改为CHAR(32) NOT NULL DEFAUL ‘123’移动到第一个位置

ALTER TABLE vipuser10 MODIFY testcol CHAR(32) NOT NULL DEFAULT '123' FIRST;

修改表—修改字段名称

口 语法格式：ALTER TABLE tbl_name CHANGE 旧字段名称 新字段名称 字段类型 [完整性约束条件] [FIRST|AFTER 字段名称]；

口示例

✓ 将testcol字段改为testcol1：

```
ALTER TABLE vipuser10 CHANGE testcol testcol1 CHAR(32) NOT NULL DEFAULT '123';
```

✓ 改变testcol1的字段的类型：

```
ALTER TABLE vipuser10 CHANGE testcol1 testcol VARCHAR(200) NOT NULL AFTER username;
```

✓ 修改字段类型和字段属性，也能完成modify：

```
ALTER TABLE vipuser10 CHANGE testcol testcol INT;
```

修改表—添加和删除默认值

口 添加默认值语法格式：ALTER TABLE tbl_name
ALTER 字段名称 SET DEFAULT 默认值；

口示例

✓ 创建表vipuser11：

```
CREATE TABLE IF NOT EXISTS vipuser11(
    id TINYINT UNSIGNED KEY AUTO_INCREMENT ,
    username VARCHAR(20) NOT NULL UNIQUE,
    age TINYINT UNSIGNED
    );
```

✓ 选中字段添加默认值：

```
ALTER TABLE vipuser11 ALTER age SET DEFAULT 18;
```

修改表—添加和删除默认值

□ 删除默认值语法格式：ALTER TABLE tbl_name
ALTER 字段名称 DROP DEFAULT；

□示例

✓ 添加一个字段：

ALTER TABLE vipuser11 ADD email VARCHAR(50) ;

✓ 给email添加默认值：

ALTER TABLE vipuser11 ALTER email SET DEFAULT 'lihui@cau.edu.cn';

✓ 删除默认值，先选中这段，再删除字段：

ALTER TABLE vipuser11 ALTER age DROP DEFAULT;

✓ 删除email默认值：

ALTER TABLE vipuser11 ALTER age DROP DEFAULT;

修改表—添加和删除主键

口 添加主键语法格式：ALTER TABLE tbl_name ADD [CONSTRAINT [symbol]] PRIMARY KEY[index_type] (字段名称,...)；

口示例

✓ 添加一个表：**CREATE TABLE IF NOT EXISTS t_test12(id INT);**

✓ 添加主键：**ALTER TABLE t_test12 ADD PRIMARY KEY(id);**

✓ 创建一个表：

```
CREATE TABLE IF NOT EXISTS t_test13(
    id INT,
    card CHAR(18),
    username VARCHAR(20) NOT NULL
    );
```

✓ 添加一个复合主键：

ALTER TABLE t_test13 ADD PRIMARY KEY(id,card);

修改表—添加和删除主键

口 删除主键语法格式：ALTER TABLE tbl_name DROP PRIMARY KEY；

口示例

✓ 删除t_test12表的主键：

```
ALTER TABLE t_test12 DROP PRIMARY KEY;
```

✓ 删除t_test13的复合主键：

```
ALTER TABLE t_test13 DROP PRIMARY KEY;
```

修改表—添加和删除唯一

- **添加唯一语法格式：ALTER TABLE tbl_name ADD [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY] [索引名称](字段名称,...)；**
- **示例**
  - ✓ 选中表vipuser12把username添加唯一约束：

```
ALTER TABLE vipuser12 ADD UNIQUE(username);
```

- **删除唯一的语法格式：ALTER TABLE tbl_name DROP {INDEX|KEY} index_name**
  - ✓ 删除vipuser12 表username主键：

```
ALTER TABLE vipuser12 DROP INDEX username;
```

修改表—修改表的存储引擎

- **添加唯一语法格式：ALTER TABLE tbl_name ENGINE=存储引擎名称；**
- **示例**
  - ✓ 修改表的存储引擎为myisam：

```
ALTER TABLE vipuser12 ENGINE=MyISAM;
```

  - ✓ 再变回来：

```
ALTER TABLE vipuser12 ENGINE=INNODB;
```

修改表—设置自增长的值

- 设置自增长的值的语法格式：

ALTER TABLE tbl_name AUTO_INCREMNET=值

- 示例：

ALTER TABLE vipuser12 AUTO_INCREMENT=100;

复制表

- 可以通过CREATE TABLE 命令**复制表的结构和数据。**
- **语法格式**

```
create [temporary] table [if not exists] table_name
[ () like old_table_name [] ]
| [AS (select_statement)];
```

复制表

- 比如，CREATE TABLE T_A LIKE T_B；此种方式在将表T_B复制到T_A时候会将表T_B完整的**字段结构**和**索引**复制到表T_A中来。
- 而CREATE TABLE T_A AS SELECT sn,sname,sage FROM T_B；此种方式只会将表T_B的字段结构复制到表T_A中来，但**不会**复制表T_B中的索引到表T_A中来。这种方式比较灵活可以在复制原表表结构的同时指定要复制哪些字段，并且自身复制表也可以根据需要增加字段结构。
- 两种方式在复制表的时候**均不会复制权限对表的设置**。比如说原本对表B做了权限设置，复制后，表A不具备类似于表B的权限。

删除表

- **删除表语法格式：DROP TABLE [IF EXISTS] table_name [,table_name] …;**
- **示例：**
  - ✓ 删除vipuser12：

    DROP TABLE vipuser12;
  - ✓ 查看数据表：

    SHOW TABLES;
  - ✓ 删除多张表：

    DROP TABLE IF EXISTS vipuser11,vipuser10, vipuser9;

注：删除不存在的表，会产生一条警告，查看警告命令：

SHOW WARNINGS;

表管理中的注意事项

（1）**关于空值（NULL）的说明**

空值通常用于表示**未知**、不**可用**或**将在以后添加的数据**，切不可将它与数字0或字符类型的空字符混为一谈。

（2）**关于列的标志(IDENTITY)属性**

任何表都可以创建一个包含系统所生成序号值的标志列。该序号值唯一标志表中的一列，且可以作为键值。每个表中只能有一个列设置为标志属性，并且该列只能是DECIMAL，INT，NUMERIC，SMALLINT，BIGINT或TINYINT数据类型的。

表管理中的注意事项

（3）**关于列类型的隐含改变**

在MySQL中，存在以下一些情形，系统会隐含地改变在CREATE TABEL语句或ALTER TBALE语句中所指定的列类型。

1）长度**小于4**的VARCHAR类型会被改变为CHAR类型。

2）由于只要一个表中存在着任何可变长度的列，都会使表中整个数据列成为变长的，因此当一张表含有任何变长的列时，例如VARCHAR，TEXT，BLOB类型的列，该表中所有大于 3 个字符的其他CHAR类型列被改变为VARCHAR类型列，而这不影响用户如何使用这些列。


第7章 MySQL表定义与完整性约束控制

- 7.1 表的基本概念
- 7.2 数据类型
- 7.3 运算符
- 7.4 表的操作
- **7.5 MySQL约束控制**
- 7.6 知识点小结
- 本章实验

7.5 MySQL约束控制

- MySQL约束控制:
  - **数据完整性约束**
  - **字段的约束**
  - **删除约束**

数据完整性约束

- 各种**完整性约束**是作为数据库关系模式定义的一部分，可通过CREATE TABLE或ALTER TABLE语句来定义。
- 一旦定义了完整性约束，MySQL服务器会**随时检测处于更新状态的数据库内容是否符合相关的完整性约束，从而保证数据的一致性与正确性。**如此，既能有效地防止对数据库的意外破坏，又能提高完整性检测的效率，还能减轻数据库编程人员的工作负担。
- 数据的完整性总体来说可分为以下4类，即**实体完整性、参照完整性、域完整性和用户自定义完整性。**

数据完整性约束

- **实体完整性**：实体的完整性强制表的标识符列或**主键的完整性**（通过约束，唯一约束，主键约束或标识列属性）。
- **参照完整性**:在删除和输入记录时，引用完整性保持表之间已定义的关系，引用完整性确保键值在所有表中一致。这样的一致性要求不能引用不存在的值。如果一个键值更改了，那么在整个数据库中，对该键值的引用要进行一致的更改。
- **域完整性**:限制类型（数据类型），格式（检查约束和规则），可能值范围（**外键约束**，**检查约束**，默认值定义，非空约束和规则）。
- **用户自定义完整性**:用户自己定义的业务规则。
- 在MySQL数据库中**不支持检查约束。可以在语句中对字段添加检查约束，不会报错，但该约束不起作用。**

字段的约束

- 设计数据库时，可以对数据库表中的一些字段设置约束条件，由数据库管理系统（例如MySQL）自动检测输入的数据是否满足约束条件，不满足约束条件的数据，数据库管理系统拒绝录入。
- MySQL支持的**常用约束条件有7种**：**主键**（primary key）约束、**外键**（foreign key）约束、**非空**(not NULL)约束、**唯一性**（unique）约束、**默认值**（default)约束、**自增约束**（auto_increment）以及**检查**(check)约束。
- 其中，**检查(check)约束需要借助触发器或者MySQL复合数据类型实现。**

主键（primary key）约束

- 设计数据库时，建议为所有的数据库表都定义一个主键，用于保证数据库表中记录的**唯一性**。一张表中只允许设置一个主键，当然这个主键可以是一个字段，也可以是一个字段组（不建议使用复合主键）。
- 在录入数据的过程中，必须在所有主键字段中输入数据，即**任何主键字段的值不允许为NULL**。
- 可以在创建表的时候创建主键，也可以对表已有的主键进行修改或者增加新的主键。
- 设置主键通常有两种方式：**表级**完整性约束和**列级**完整性约束。

  (1)如果一个表的主键是单个字段ID

  如果用表级完整性约束，就是用PRIMARY KEY命令单独设置主键为ID列。

  语法规则: PRIMARY KEY(字段名)

主键（primary key）约束

(1)如果一个表的主键是单个字段ID

如果用**列级**完整性约束，就是直接在该字段的数据类型或者其他约束条件后加上“PRIMARY KEY”关键字，即可将该字段设置为主键约束

语法规则: 字段名 数据类型[其他约束条件] PRIMARY KEY

(2)如果一个表的主键是多个字段的组合（例如，字段名1与字段名2共同组成主键），定义完所有的字段后，使用下面的语法规则设置复合主键。

语法规则: PRIMARY KEY（字段名1，字段名2）

- **修改主键**：

  利用**ALTER TABLE**

主键（primary key）约束创建示例

创建单字段主键：

```
CREATE TABLE IF NOT
EXISTS VIPuser1(
    id INT PRIMARY KEY,
    username VARCHAR(20)
    );
```

创建多字段主键：

```
CREATE TABLE IF NOT EXISTS
VIPuser2(
    id INT,
    username VARCHAR(20),
    card CHAR(18),
    PRIMARY KEY(id,card)
    );
```

省略PRIMARY也可以创建主键：

```
CREATE TABLE IF NOT EXISTS user4(
    id INT,
    username VARCHAR(20) KEY
 );
```

外键约束（FOREIGN KEY）约束

- 1.**外键**是表的一个**特殊字段**。
  - ✓ 被参照的表是主表，外键所在字段的表为子表。
  - ✓ 设置外键的原则需要记住，就是依赖于数据库中已存在的表的主键。
  - ✓ 外键的作用是建立该表与其父表的关联关系。
  - ✓ 父表中对记录做操作时，子表中与之对应的信息也应有相应的改变。
- 2.外键的作用保持**数据的一致性和完整性**。
- 3.可以实现**一对一或一对多的关系**。

外键约束（FOREIGN KEY）约束

注意：

1.父表和子表必须使用**相同的存储引擎**，而且禁止使用临时表。

2.数据表的存储引擎只能为**InnoDB**。

3.外键列和参照列必须具有**相似的数据类型**。其中数字的长度或是否有符号位必须相同；而字符的长度则可以不同。

4.外键列和参照列必须**创建索引**。如果外键列不存在索引的话，MySQL将自动创建索引。

外键约束（FOREIGN KEY）约束

- 外键约束的**参照操作**:

1.**CASCADE**

从父表删除或更新且自动删除或更新子表中**匹配的行**。

2.**SET NULL**

从父表删除或更新行，并设置子表中的外键列为**NULL**。如果使用该选项，必须保证子表列没有指定NOT NULL。

3.**RESTRICT**

拒绝对父表的删除或更新操作。

4.**NO ACTION**

标准SQL的关键字，在MySQL中与**RESTRICT**相同。

外键约束（FOREIGN KEY）约束

- **设置外键的两种方式**：
  - 在**表级**完整性下定义外键约束**语法格式**：

    FOREIGN KEY（表A的字段名列表）REFERENCES表B（字段名列表）
    [ON DELETE {CASCADE| RESTRICT |SET NULL | NO ACTION}]
    [ON UPDATE {CASCADE| RESTRICT |SET NULL | NO ACTION}]

    级联选项有4种取值，其意义如下：

    1）**cascade**:父表记录的删除(delete)或者修改(update)操作，会自动删除或修改子表中与之对应的记录。

    2） **set null**：父表记录的删除(delete)或者修改(update)操作，会将子表中与之对应记录的外键值自动设置为null.值。

    3）**no action**:父表记录的删除(delete)或修改(update)操作，如果子表存在与之对应的记录，那么删除或修改操作将失败。

    4）**restrict**:与no action功能相同，且为级联选项的默认值。

外键约束（FOREIGN KEY）约束

- 如果表已经建好，那么可以通过**alter table**命令添加
  语法如下:

  alter table table_name
  add [constraint 外键名] foregin key [id] (index_col_name, ……)
  references table_name(index_col_name, ……)
  [on delete {cascade| restrict |set null | no action}]
  [on update {cascade| restrict |set null | no action}]
- 如果表还没建立，那么可以在**create table**中指定。

➢ 另一种是在**列级**完整性下定义外键约束
  - 在列级完整性上定义外键约束，就是直接在列的后面添加**references**命令。

外键约束（FOREIGN KEY）约束

口 示例:

✓ 创建员工表指定外键名称：

```
CREATE TABLE IF NOT EXISTS employee(
    id SMALLINT UNSIGNED AUTO_INCREMENT KEY,
    username VARCHAR(20) NOT NULL UNIQUE,
    depId TINYINT UNSIGNED,
    CONSTRAINT emp_fk_dep FOREIGN KEY(depId)
REFERENCES department(id)
    )ENGINE=INNODB;
```

✓ 给employee 表增加外键：

```
ALTER TABLE employee ADD CONSTRAINT
emp_fk_dep FOREIGN KEY(depId) REFERENCES
department(id);
```

外键约束（FOREIGN KEY）约束

示例创建级联外键:

- ✓ 删除外键的级联同时删除子表：

```
CREATE TABLE IF NOT EXISTS department(
    id TINYINT UNSIGNED AUTO_INCREMENT KEY,
    depName VARCHAR(20) NOT NULL UNIQUE
    )ENGINE=INNODB;
```

- ✓ 创建员工表employee(子表)：

```
CREATE TABLE IF NOT EXISTS employee(
    id SMALLINT UNSIGNED AUTO_INCREMENT KEY,
    username VARCHAR(20) NOT NULL UNIQUE,
    depId TINYINT UNSIGNED,
    FOREIGN KEY(depId) REFERENCES department(id)
ON DELETE CASCADE
    )ENGINE=INNODB;
```

外键约束（FOREIGN KEY）约束

- **表级完整性约束**和**列级完整性约束**都是在**CREATE TABLE**语句中定义。
- 另外一种方式：就是使用**完整性约束命名字句CONSTRAINT**,用来对完整性约束条件命名，从而可以灵活的增加、删除一个完整性约束条件。
- **完整性约束命名字句格式：**

  constraint <完整性约束条件名> [PRIMARY KEY 短语| FOREIGN KEY 短语 | CHECK 短语]
- 创建表时，建议**先创建父表，然后再创建子表**，并且建议子表的外键字段与父表的主键字的数据类型（包括长度）相似或者可以相互转换（**建议外键字段与主键字数据类型相同**）。

非空约束

- 被标识了非空的就不能有空值（null），在插入记录时不能有空否则会报错。没有标志可以为空默认值为NULL。
- **非空一般与默认值使用。**
- 如果某个字段满足非空约束的要求（例如学生的姓名不能取NULL值），则可以向该字段添加非空约束。
- 若设置某个字段的非空约束，直接在该字段的数据类型后加上“NOT NULL”关键字即可。
- 非空约束限制该字段的内容**不能为空**，但**可以是空白**。
- **语法规则**: 字段名 数据类型 NOT NULL

非空约束示例

- ✓ 创建一个表：

```
CREATE TABLE IF NOT EXISTS VIPuser7(
    id INT UNSIGNED KEY AUTO_INCREMENT,
    username VARCHAR(20) NOT NULL,
    password CHAR(32) NOT NULL,
    age TINYINT UNSIGNED
    );
```

- ✓ 查看表结构：

```
DESC VIPuser7;
```

默认约束

- **默认值一般与非空约束使用**，在插入记录没有给字段赋值时，就使用默认值。根据具体问题具体分析给那些字段添加默认值。
- 如果某个字段满足默认值约束要求，可以向该字段添加默认值约束，例如，可以将课程course表的人数上限up_limit字段设置默认值60。
- 若设置某个字段的默认值约束，直接在该字段数据类型及约束条件后加上“**DEFAULT** 默认值”即可。
- **语法规则**: 字段名 数据类型[其他约束条件] DEFAULT默认值

默认约束示例

- ✓ 创建表，带有默认约束：

```
CREATE TABLE IF NOT EXISTS VIPuser8(
    id INT UNSIGNED KEY AUTO_INCREMENT,
    username VARCHAR(20) NOT NULL,
    password CHAR(32) NOT NULL,
    age TINYINT UNSIGNED DEFAULT 18,
    addr VARCHAR(50) NOT NULL DEFAULT '北京',
    sex ENUM('男','女','保密') NOT NULL DEFAULT '男'
    );
```

- ✓ 查看表结构： DESC VIPuser8;

唯一约束

- 唯一性约束就是唯一性索引，一个表中只有一个主键，**一个表中可以有多个唯一**。被标志了唯一字段的值不允许出现重复，但是有一个特例null不算重复的值。
- **唯一性约束**可以通过**UNIQUE KEY**来实现，Key可以省略。
- 与主键约束不同，一张表中可以存在**多个唯一性约束**，并且满足唯一性约束的字段可以取NULL值。
- 若设置某个字段为唯一性约束，直接在该字段数据类型后加上“UNIQUE”关键字即可。**语法规则:** 字段名 数据类型 UNIQUE
- 如果某个字段存在多种约束条件，约束条件的顺序是**随意**的。
- 唯一性约束实质上是通过唯一性索引实现的，因此唯一性约束的字段一旦创建，那么该字段将自动创建唯一性索引。如果要删除唯一性约束，只需删除对应的唯一性索引即可。

唯一约束示例

- ✓ 创建唯一性约束：

```
CREATE TABLE IF NOT EXISTS VIPuser9(
    id TINYINT UNSIGNED KEY AUTO_INCREMENT,
    username VARCHAR(20) NOT NULL UNIQUE,
    card CHAR(18) UNIQUE
    );
```

- ✓ 查看表结构

```
DESC VIPuser9;
```

自增约束

- **AUTO_INCREMENT**是MySQL唯一扩展的完整性约束，当为数据库表中插入新记录时，字段上的值会自动生成唯一的ID。
- 在具体设置AUTO_INCREMENT约束时，**一个数据库表中只能有一个字段**使用该约束，该字段的数据类型**必须是整型类型**。
- 由于设置auto_increment约束后的字段会生成唯一的ID，所以该字段也经常会设置为PK主键。
- 从1开始增长的每次加1。**自增长要配合主键使用**，被标志为自增长的一定是主键，但是是主键不一定是自增长，它们可以调换位置，而且自增长只对整数、整数列有效果
- MySQL中通过SQL语句的**AUTO_INCREMENT**来实现：

```
CREATE TABLE table_name(
    属性名 数据类型 AUTO_INCREMENT,
……
);
```

自增约束示例

```
CREATE TABLE IF NOT EXISTS VIPuser5(
    id SMALLINT KEY AUTO_INCREMENT,
    username VARCHAR(20)
    );
```

在建表是指定自增长值100，如果开始不知道会从100开始

```
CREATE TABLE IF NOT EXISTS VIPuser6(
     id SMALLINT KEY AUTO_INCREMENT,
     username VARCHAR(20)
     )AUTO_INCREMENT=100;
```

检查约束和删除约束

- **检查约束**：
  - 检查约束是用来检查数据表中字段值的有效性的一个手段，例如，学生信息表中的年龄字段是没有负数的，并且数值也是有限制的，当前大学生的年龄一般在15~30岁之间。
  - 其中，前面讲述的默认值约束和非空约束可以看作是特殊的检查约束。
  - 在创建表时设置列的检查约束有两种：设置**列级**约束和**表级**约束。
- **删除约束**：
  - 在MySQL 数据库中，一个字段的所有约束都可以用 **alter table** 命令删除。

综合应用示例

✓ 创建一个注册用户表：

```
CREATE TABLE IF NOT EXISTS reg_user(
    id SMALLINT UNSIGNED KEY AUTO_INCREMENT,
    username VARCHAR(20) NOT NULL UNIQUE,
    password CHAR(32) NOT NULL,
    email VARCHAR(50) NOT NULL DEFAULT
'caulihui@cau.edu.cn',
    age TINYINT UNSIGNED DEFAULT 18,
    sex ENUM('男','女','保密') DEFAULT '保密',
    addr VARCHAR(200) NOT NULL DEFAULT '北京',
    salary FLOAT(6,2),
    regTime INT UNSIGNED,
    face CHAR(100) NOT NULL DEFAULT 'default.jpg'
    );
```


第7章 MySQL表定义与完整性约束控制

- 7.1 表的基本概念
- 7.2 数据类型
- 7.3 运算符
- 7.4 表的操作
- 7.5 MySQL约束控制
- **7.6 知识点小结**
- 本章实验

7.6 知识点小结

本章知识小结：

- 表的基本概念
- MySQL支持的数据类型和运算符
- 表的基本操作，有创建、查看、修改、复制、删除
- MySQL的约束控制，如何定义和修改字段的约束条件。

对数据库表的常用操作：

- **查看**表结构：DESC 表名
- **创建**表：CREATE TABLE 表名 (列名 列类型，列名2 列类型2，...);
- **修改**表：
  - ✓ 修改**表名**：ALTER TABLE 表名 RENAME TO 新表明;
  - ✓ **添加列**：ALTER TABLE 表名 ADD 列名 列类型;
  - ✓ 修改**列类型**：ALTER TABLE 表名 MODIFY 列名 列类型;
  - ✓ 修改**列名**：ALTER TABLE 表名 CHANGE 列名 新列名 列类型;
  - ✓ **删除列**：ALTER TABLE 表名 DROP 列名;
- **删除**表：DROP 表名;
- **查询**表：SHOW TABLES;


第7章MySQL表定义与完整性约束控制

- 7.1 表的基本概念
- 7.2 数据类型
- 7.3 运算符
- 7.4 表的操作
- 7.5 MySQL约束控制
- 7.6 知识点小结
- **本章实验**

本章实验

口 实验内容：

（3）在学生信息数据库中创建学生表，包括字段：学号、姓名、性别、出生日期、专业号、班级、电话，要求学号、姓名非空；

（4）在学生基础信息表中的一列身份证号添加到学生表中，并将学生表中的出生日期列删除。

Thanks !

See you