MySQL学习笔记

课程传送门

数据库

定义

- 数据库是"按照数据结构来组织、存储和管理数据的仓库"
- 是一个长期存储在计算机内的、有组织的、可共享的、统一管理的大量数据的集合文件
    - 数据库的具体体现，就是磁盘的文件或者内存的一段数据
    - 一种长期存储手段，不主动删除数据不应该丢失
    - 数据库中数据存储是有组织的结构，方便数据读取和修改等
    - 数据库数据属于可共享的，符合真正开发需求

分类

非关系型数据库适用于需要高性能、灵活性和分布式处理的场景
关系型数据库适用于需要强一致性和复杂查询的场景
绝大部分应用采用混合模式，二者结合使用
程序主体数据一般存储在关系型数据库
程序缓存数据和高并发数据存储到非关系型数据库
以关系型数据库为主，非关系型数据库为辅

非关系型数据库

非关系型数据库并没有统一存储结构标准，现常见结构有键值、文档、JSON类型等，对高性能需求设计

- 特点
    - 灵活的数据模型：支持多种数据模型，例如键值对、文档型、列族型和图形数据库
    - 无固定的表结构：不需要预定义固定的表结构，适应非结构化或者半结构化数据
    - 高性能读写：少关系型，注重高性能读写能力，适用于大规模数据和高并发访问模型

关系型数据库

数据按照类别进行存储，每个类型存储到一个容器(表)中，表和表之间可以建立关系，可以进行关联操作，性能相对一般

- 特点
    - 结构化数据模型：数据以表格形式存储，具有固定的结构。例如学生和分数分别存储到不同的表
    - ACID事务：通过强大的事务支持，保证数据的原子性、一致性、隔离性和持久性。例如转账失败、钱不损失
    - 丰富查询语句：支持SQL语句，能够进行复杂的关联数据查询。例如查询学生以及学生的分数
    - 数据一致性：数据的关系和约束确保数据的一致性和完整性。例如存储学生数据，保证身份证号唯一且不为空

关系型数据库存储设计规则遵循E-R模型
E(Entity)：代表实体类别，关系型数据库中一类数据，对应数据库中的一张表存储
R(Relationship)：表和表可以维护某种关系，可以通过关系进行多表操作

数据库的存储单位

- 库：数据库中最大的存储单位，内部存储表
- 表：一类数据和实体的集合，每类数据存储到一张表中，表存储到数据库中
- 列：最小的存储单位，代表一个属性
- 行：一行数据代表一个实体，操作的基本单位，表中的数据按照行来进行存储，一行即为一条记录

数据库管理系统

定义

数据库管理系统(DataBase Management System,DBMS)，指一种操作和管理数据库的大型软件，用户通过数据库管理系统操作数据库中的数据

结构化查询语句(SQL)

分类

SQL包括所有对数据库的操作，主要是由数据定义、数据操纵、数据查询、数据控制、事务控制等SQL语言的使用规定组成

- 数据定义(DDL)：创建和修改盛放数据的容器
- 数据操纵(DML)：表中添加、修改、删除数据
- 数据查询(DQL)：表中数据多条件查询
- 事务控制(TCL)：事务启动、提交和回滚
- 数据控制(DCL)：账号创建、权限控制

基本语句

开启MySQL服务

方式：

- 右键'此电脑'，选择'管理'，找到'服务'中的'MySQL82'(你的MySQL名称)然后右键可以选择'开启'或'停止'
- 以管理员身份运行'命令提示符'，输入'net stop MySQL82'(你的MySQL名称)表示关闭MySQL服务，输入'net start MySQL82'表示开启MySQL服务

连接mysql服务命令(参数不区分前后顺序)

```
mysql -u <username> -p <password> -h <hostname> <databasename>
```

- -u <username>：用于指定你要连接的MySQL数据库的用户名
- -p <password>：表示密码，**后面紧跟密码**，`中间没有空格`，如果你不希望在命令行中显示密码，可以不指定密码，直接-p，然后在提示下手动输入密码
- -h <hostname>：用于指定MySQL服务器的主机名或IP地址，如果MySQL在本地运行，可以用localhost
- -P <port>：用于指定连接MySQL服务器的端口号，默认为3306
- <databasename>：是你要连接的数据库的名称，连接后会默认使用这个数据库

SQL注释

- 单行注释：

```
#注释内容
```

- 单行注释：

```
-- 注释内容 其中--后面的空格必须有
```

- 多行注释：

```
/* 注释内容 */
```

查看版本

```
SELECT version();
```

退出连接

```
EXIT
```

命名规定和规范

标识符命名规定

- 数据库名、表名不得超过30个字符，变量名限制为29个
- 必须只能包含A-Z，a-z，0-9，_共63个字符，而且不能数字开头
- 数据库名、表名、字段名等对象名中间`**不能**`包含空格
- 同一个MySQL软件中，数据库不能同名，同一个库中，表不能重名，同一个表中，字段不能重名
- 必须保证你的字段没有和保留字、数据库系统或常用方法冲突。如果坚持使用，在SQL语句中使用`引起来

标识符命名规范(基于阿里巴巴规范手册)

阿里巴巴的SQL规范建议通常是为了确保数据库操作的效率、安全性和可维护性

- 注释应该清晰、简洁地解释SQL语句的意图、功能和影响
- 库、表、列名应该使用小写字母，并使用下划线(_)或驼峰命名法(不建议写大写字母，容易和软件冲突)
- 库、表、字段名应该简洁明了，具有描述性，反映其所存储数据的含义
- 库名应与对应的程序名一致。如程序名为EcommercePlatform，数据库命名为ecommerce_platform
- 表命名最好是遵循"业务名称_表"的作用，例如alipay_task，force_project
- 列名应遵循"表实体_属性"的作用，例如product_name

数据定义(DDL)

DDL的功能有创建库、定字段、创建表，用于完成库和表的管理

定义

数据定义语言(Data Definition Language)
DDL用于定义和管理数据库的结构，包括库、表、索引、视图等数据库对象的创建、修改和删除
DDL不涉及对数据的操作，而是关注数据库的结构和元数据(容器)

关键字

- CREATE：用于创建数据库、表、索引、视图等
- ALTER：用于修改数据库对象的结构，如修改表结构、添加列、删除列等
- DROP：用于删除数据库对象，如删除表、删除索引等

库管理

创建库

创建库，必须指定库名，字符集和排序方式可选

- 方式1：创建数据库，使用默认的字符集和排序方式

```
CREATE DATABASE 数据库名;
```

- 方式2：判断并创建默认字符集库(推荐)

```
CREATE DATABASE IF NOT EXISTS 数据库名;
```

- 方式3：创建指定字符集库或者排序方式

```
CREATE DATABASE 数据库名 CHARACTER SET 字符集;
CREATE DATABASE 数据库名 COLLATE 排序方式;
```

- 方式4：创建指定字符集和排序方式库

```
CREATE DATABASE 数据库名 CHARACTER SET 字符集 COLLATE 排序方式;
```

字符集就是编码格式，决定了数据如何编码存储
排序方式决定了如何比较和排序存储在数据库中的文本数据
常见字符集(Character Set)：

- utf8：3字节存储一个字符，易出现乱码
- utfmb4：4字节存储一个字符

常见排序方式(Collate)：

- utf8mb4_0900_ai_ci：UTF-8不区分大小写的排序(mysql8+默认排序规则)
- utf8mb4_0900_as_cs：UTF-8的Unicode排序规则，区分大小写

字符集：utf8mb4是一种广泛支持各种语言字符的字符集
排序规则：utf8mb4_0900_ai_ci是一种不区分大小写的排序规则

- 查看默认字符集和排序方式命令

```
SHOW VARIABLES LIKE 'character_set_database';
SHOW VARIABLES LIKE 'collation_database';
```

查看和使用库

使用和查看库，包括展示和切换库等命令

- 方式1：查看当前所有库

```
SHOW DATABASES;
```

- 方式2：查看当前使用的库

```
SELECT DATABASE();
```

- 方式3：查看指定库下所有表

```
SHOW TABLES FROM 数据库名;
```

- 方式4：查看创建库的信息

```
SHOW CREATE DATABASE 数据库名;
```

- 方式5：切换库/选中库

```
USE 数据库名;
```

要操作表格和数据之前必须先说明是对哪个数据库进行操作，先USE库

修改库

- 方式1：修改库的编码字符集

```
ALTER DATABASE 数据库名 CHARACTER SET 字符集; #修改字符集gbk utf8
ALTER DATABASE 数据库名 COLLATE 排序方式; #修改排序方式
ALTER DATABASE 数据库名 CHARACTER SET 字符集 COLLATE 排序方式; #修改字符集和排序方式
```

DATABASE不能改名，如果要改名，只能建立一个新库然后把旧库内容复制过去，再删除旧库

删除库

`删除库前要三思`

- 方式1：直接删除库

```
DROP DATABASE 数据库名;
```

- 方式2：判断并删除库(推荐)

```
DROP DATABASE IF EXISTS 数据库名;
```

创建表

核心要素：

- 指定表名
- 指定列名
- 指定列类型

可选要素：

- 指定列约束
- 指定表配置
- 指定表和列的注释(建议都写，很有必要)
- 推荐使用[IF NOT EXISTS]，直接创建可能存在报错

```
CREATE TABLE [IF NOT EXISTS] 表名(
  列名 类型 [列可选约束], #列之间用,进行分割
  列名 类型 [列可选约束] [COMMENT '列可选注释'],
  列名 类型 [列可选约束] [COMMENT '列可选注释'],
  列名 类型 [列可选约束] [COMMENT '列可选注释'],
  列名 类型 [列可选约束] [COMMENT '列可选注释'],
  [列可选约束]
)[表可选约束] [COMMENT '表可选注释'];
```

例：创建一个名为posts的博客文章表，包括一些常见的数据类型、编码设置以及列注释

```
CREATE TABLE IF NOT EXISTS post(
  post_id INT AUTO_INCREMENT PRIMARY KEY, #列之间用,进行分割
  title VARCHAR(255) NOT NULL COMMENT 'The title of the blog post',
  content TEXT NOT NULL COMMENT 'The content of the blog post',
  author_id INT NOT NULL COMMENT 'The ID of the author of the post',
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP COMMENT 'The timestamp when the
post was created',
  CONSTRAINT fk_author FOREIGN KEY (author_id) REFERENCES authors(author_id)
) ENGINE=InnoDB DEFAULT CHARSET = utf8mb4 COMMENT 'Table storing blog posts';
#ENGINE=InnoDB是指定描述它是哪个驱动，CHARSET选择字符集
```

例：创建一个图书表books，判断不存在再创建，并且手动设置books表字符集为utf8mb4，添加表注释内容，同时图书表应该有以下列：

- 图书名称book_name列，类型为varchar(20)，添加注释
- 图书价格book_price列，类型为double(4,1)，添加注释
- 图书数量book_num列，类型为int，添加注释

```
CREATE TABLE IF NOT EXISTS books(
  book_name VARCHAR(20) COMMENT '图书名称',
  book_price DOUBLE(4,1) COMMENT '图书价格',
  book_num INT COMMENT '图书数量'
)CHARSET = utf8mb4 COMMENT '图书表';
```

数据类型

整数

MySQL支持多种数据类型，包括整数、浮点数、定点数、字符串、日期时间等
MySQL支持SQL标准整数类型INTEGER(或INT)和SMALLINT，作为标准的扩展，MySQL还支持整数类型TINYINT、MEDIUMINT和BIGINT
MySQL支持整数类型TINYINT、MEDIUMINT和BIGINT但SQL不支持

类型	存储(字节)	最小值有符号	最小值无符号	最大值有符号	最大值无符号
TINYINT	1	-2^7	0	2^7-1	2^8-1
SMALLINT	2	-2^{15}	0	$2^{15}-1$	$2^{16}-1$
MEDIUMINT	3	-2^{23}	0	$2^{23}-1$	$2^{24}-1$
INT	4	-2^{31}	0	$2^{31}-1$	$2^{32}-1$
BIGINT	8	-2^{63}	0	$2^{63}-1$	$2^{64}-1$

无符号==无负号，整数类型都可以添加`UNSIGNED`修饰符，添加以后对应列数据变成无符号类型，值从0开始

```
stu_age TINYINT UNSIGNED COMMENT '年龄字段，tinyint类型，无符号',
stu_age TINYINT COMMENT '年龄字段，tinyint类型，有符号'
```

浮点数

FLOAT和DOUBLE类型表示近似的数据值

类型	存储(字节)	M(小数+整数位数)	D(小数位数)
FLOAT(M,D)	4	M最大为24	D最大为8
DOUBLE(M,D)	8	M最大为53	D最大为30

浮点数支持`UNSIGNED`修饰，只保留正值范围，没有负值不会影响正值的数据范围
例：

```
stu_height FLOAT(4,1) UNSIGNED COMMENT '身高，保留一位小数，多位会四舍五入',
```

定点数

DECIMAL类型存储精确的数据值，底层直接用十进制进行存储，不会存在浮点误差

类型	存储(字节)	M(小数+整数位数)	D(小数位数)
DECIMAL(M,D)	动态计算	M最大为65	D最大为30

DECIMAL类型的存储空间是可变的，受指定的精度和规模影响，如果D=0，DECIMAL则不包含小数点和小数部分
例：

```
emp_salary DECIMAL(8,1) COMMENT '工资，保留一位小数，多位会四舍五入',
```

字符串

字符串(文本)	特点	长度	长度范围(字符)	存储空间	性能
CHAR(M)	固定长度	M	$0 \leq M \leq 255$	M*4个字节(utf8mb4)	较好
VARCHAR(M)	可变长度	M	MySQL一行数据最多65535字节	(M*4+1)个字节(utf8mb4)	一般

CHAR(M)类型一般需要预先定义字符串长度，如果不指定M，则默认是1
保存数据的实际长度比CHAR类型声明的长度小，则会在右边填充空格以达到声明长度
当MySQL检索CHAR类型数据时，会自动删除尾部空格

VARCHAR(M)类型定义时必须指定长度M，否则报错
MySQL4.0以下版本，VARCHAR(20)：指的是20个**字节**，如果存放UTF8汉字时，只能存6个(每个汉字3字节)
MySQL5.0以上版本，VARCHAR(20)：指的是**20字符**，即不管你是什么编码格式，就最多存20个这种编码格式的字符
检索VARCHAR类型数据时，会保留尾部空格，多出来的1个字节是可以让字符串为空而不需要在尾部插入空格占位
如果使用utf8mb4字符集进行存储，那么一行最多有16383个字符，此时可以把VARCHAR改成TEXT文本类型

例：使用单字符集存储字符时，不是用utf8mb4存储

插入值	CHAR(4)	所需存储	VARCHAR(4)	所需存储
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes

MySQL中一行数据最大65535字节，但TEXT和BLOBs类型的列不受65535字节限制，TEXT不需要指定长度，有固定的大小限制，最大是65535字节，但不占用一行的最大限制空间
使用varchar超过了65535字节限制解决方案：

- 缩小字符大小限制，把M变小
- 更换字符集，使用占字节数更少的字符集进行替换
- 将字符串类型改成TEXT(推荐)

文本类型
在向TEXT类型的字段保存和查询数据时，系统自动按照实际长度存储，不需要预先定义长度

文本字符串类型	特点	长度(字符)	存储范围(字节)	占用存储空间
TINYTEXT	小文本、可变长度	L	$0 \leq x < 2^8$	$L+2$个字节

文本字符串类型	特点	长度(字符)	存储范围(字节)	占用存储空间
TEXT	文本、可变长度	L	$0 \leq x < 2^{16}$	$L + 2$个字节
MEDIUMTEXT	中等文本、可变长度	L	$0 \leq x < 2^{24}$	$L + 3$个字节
LONGTEXT	大文本、可变长度	L	$0 \leq x < 2^{32}$	$L + 4$个字节(最大4G)

例：

```
CREATE TABLE 表名(
  tx TEXT
)
```

经验：

- 短文本，固定长度使用char，如性别、手机号
- 短文本，非固定长度使用varchar，如姓名、地址
- 大文本，建议存储到文本文件，使用varchar记录文件地址，不推荐使用TEXT直接存储大文本，存取性能较差

`时间类型`
时间类型就是`特殊格式的字符串`，插入数据的时候需要用''引起来，时间类型如果需要自动赋值，需要手动添加相关设置

类型	名称	字节	日期格式	最小值	最大值
YEAR	年	1	YYYY或YY	1901	2155
TIME	时间	3	HH:MM:SS	-838:59:59	838:59:59
DATE	日期	3	YYYY-MM-DD	1000-01-01	9999-12-03
DATETIME	日期时间	8	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00	9999-12-31 23:59:59
TIMESTAMP	日期时间	4	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00	2038-01-19 03:14:07

YEAR类型赋值00-99对应年限，[00-69]对应[2000-2069]，[70-99]对应[1970-1999]，建议四位年值
默认情况下，时间需要主动赋予默认值和修改值
例：

- 插入默认当前时间和修改自动更新当前时间

```
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

- 插入默认当前时间

```
ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
dt DATETIME DEFAULT CURRENT_TIMESTAMP
```

修改表

- 添加一列[可以指定X字段前或后]

  ```
  ALTER TABLE 表名 ADD 字段名 字段类型 [FIRST|AFTER 字段名];
  ```

- 修改列名

  ```
  ALTER TABLE 表名 CHANGE 原字段名 字段名 新字段类型 [FIRST|AFTER 字段名];
  ```

- 修改列类型

  ```
  ALTER TABLE 表名 MODIFY 字段名 新字段类型 [FIRST|AFTER 字段名];
  ```

- 删除一列

  ```
  ALTER TABLE 表名 DROP 字段名;
  ```

- 修改表名

  ```
  ALTER TABLE 表名 RENAME [TO] 新表名; #TO可以省略
  ```

删除表

- 删除数据表

  ```
  DROP TABLE [IF EXISTS] 数据表1 [,数据表2,···,数据表n];
  ```

- 清空表数据

  ```
  TRUNCATE TABLE 表名;
  ```

删除表和清空表数据都是 `无法回滚` 的

数据操纵(DML)

数据操纵语言(Data Manipulation Language)用于数据库数据插入、删除、更新三种操作，不影响表结构，但是会真正影响数据库数据
DML关键字：

- INSERT：用于数据插入关键字
- UPDATE：用于数据修改关键字
- DELETE：用于数据删除关键字

数据操作的最基本单位是 `行`，按行进行增删改查

插入数据

- 为表的一行所有字段(列)插入数据

```
INSERT INTO 表名 VALUES (value1, value2,...);
```

  值列表中需要为表的每一个字段赋值，并且值的顺序和类型必须和指定的列顺序相同

- 为表的一行指定字段(列)插入数据(推荐)

```
INSERT INTO 表名(列名1,列名2,...) VALUES(value1, value2,...);
```

  值列表中需要为表名后指定的列赋值，并且值的顺序和类型必须和指定的列顺序相同

- 同时插入多条记录

```
INSERT INTO 表名 VALUES(value1, value2,...),...,(value1,value2,...);
或
INSERT INTO 表名(列名1,列名2,...) VALUES(value1,value2,...),...,
(value1,value2,...);
```

日期类型和字符串数据应包含在单引号中，列名不需要加单引号，值需要加

更新数据

- 修改表中所有行数据(全表修改)

```
UPDATE table_name SET column1=value1,column2=value2,...,columnn=valuen;
```

  更新表中所有行指定列的数据

- 修改表中符合条件行的数据(条件修改)

```
UPDATE table_name SET column1=value1,column2=value2,...,columnn=valuen
[WHERE condition];
```

  条件修改只是在后面添加WHERE,WHERE后面指定条件即可

在查询过程中支持对原列数据进行加减乘除运算

删除数据

- 删除表中所有行数据(全表删除)

```
DELETE FROM table_name;
```

  只删除表的数据，不会删除表的相关记录，比TRUNCATE删除的少

- 删除表中符合条件行的数据(条件删除)

```
DELETE FROM table_name [WHERE condition];
```

开发中很少使用全表删除，DELETE和TRUNCATE都会删除表中所有数据，TRUNCATE不仅删除表数据，还会删除数据库id的最大记录值
条件且：**AND**
条件或：**OR**
异或：**XOR**

数据查询(DQL)

数据查询语句(Data Query Language)用于查询数据库数据，不影响库表结构，也不会影响原表数据，会基于原表数据查询出一个虚拟表
DQL关键字：

- SELECT：用于查询数据的关键字

分类：

- 单表查询
- 多表查询

多表查询只是比单表查询多了一个多表数据合并语法，也就是说多表查询是多个单表查询的结果的合并

基础SELECT语法(不指定条件)

- 非表查询

  ```
  SELECT 1;
  SELECT 9/2;
  SELECT VERSION();
  ```

  类似Java控制台输出，直接输出结果

- 指定表

  ```
  SELECT 列名1, 列名2, 列名3 FROM 表名;
  或
  SELECT 表名.*, 表名.列名 FROM 表名;
  ```

  指定表，查询表中全部或某些列，如果是所有列可以用*代替，带有通配符*的应该放在最前面

- 查询列起别名

  ```
  SELECT 列名1 as 别名, 列名2, 列名3 as 别名 FROM 表名;
  或
  SELECT 列名1 别名, 列名2, 列名3 别名 FROM 表名;
  ```

  查询列可以起别名，as可以省列
  起别名的意义主要是简化列名或者对应后期Java数据属性等
  如果别名想要区分大小写，可以添加双引号，如"Name"

- 去除重复行

  ```
  SELECT DISTINCT 列名 [,列名,列名] FROM 表名;
  ```

  指定列值去重复行，可以指定单列或者多列，但是DISTINCT关键字只写一次并且写在最前面

- 查询常数

```
SELECT '公司' as company, 列名, 列名 FROM 表名;
```

SELECT查询还可以对常数进行查询，就是在SELECT查询结果中添加固定的常数列，取值是自己赋予的，不是从表中取出的

ENUM('男','女')：ENUM是枚举类型，表示值不是男就是女
SET('北京','上海','广州',' 深圳')：SET类型表示可以选多个，如'北京'、'上海,深圳'等
NOW()函数可以获取当前时间
IFNULL(列名, 默认值)：如果该列对应的该行的数据为NULL(空)，则给予其设置的默认值
在查询时依然可以进行加减乘除运算

显示表结构

- 使用命令查看表结构

```
DESCRIBE employees;
或者
DESC employees;
```

在没有可视化工具的情况下可以查看表结构，使用命令查看表中的列和列的特征

Field	Type	Null	Key	Default	Extra

各字段含义：

- Field：表示字段名称
- Type：表示字段类型
- Null：表示该列是否可以存储NULL值
- Key：表示该列是否是主键，是否已编辑索引
  - PRI表示该列是表主键的一部分
  - UNI表示该列是UNIQUE索引的一部分
  - MUL表示在列中某个给定值允许多次出现
- Default：表示该列是否有默认值，如果有且没有指定值就赋给默认值
- Extra：表示可以获取的与给定列有关的附加信息，如AUTO_INCREMENT等

过滤数据(条件查询)

- 非全表，添加过滤条件

```
SELECT 字段1,字段2 FROM 表名 WHERE 过滤条件;
```

WHERE添加以后，就不是全表查询，先过滤条件，把符合的行筛选出来，再返回这些行指定的列

FROM：执行参照和查询的表
WHERE：进行表中数据的筛选
SELECT：进行符合条件行的数据类的筛选

分组查询

先将数据行按照某一或多个特性列进行分组，最后查询每组的特性
分组查询的结果只能是分组特性列或聚合函数

```
SELECT 分组列,分组列,... 聚合函数,聚合函数,...
FROM table
[WHERE condition]
[GROUP BY 分组列,分组列,... HAVING 分组后条件]
# where是分组前的筛选条件，having是分组后的筛选条件
# where可以随时出现，having比较一般都是聚合函数，且只能在group by后面使用
# having可以直接使用select后面查询到的列的名称，因为having是在分组后执行，查询结果已经有了，也就是复用列名，而where不能复用，因为是在select查询前执行
```

排序查询

按照某一或多个特性列进行数据排序，不会影响结果条数，只是改变结果顺序

```
SELECT 列,列,... 函数
FROM table
[WHERE condition]
[ORDER BY 排序列 ASC|DESC,排序列 ASC|DESC,...]
# ASC为正序(默认值)，DESC为倒序，多列排序只有第一列相同，第二列才会生效
```

数据切割(分页查询)

将结果进行分页切割，按照指定的区域一段一段的进行展示

```
SELECT 列,列,函数
FROM table
[WHERE condition]
[LIMIT [位置偏移量,] 行数]
```

位置偏移量：可选参数，不写默认是0，代表不偏移
行数：返回的记录条数
LIMIT必须放在整个SELECT语句的最后
LIMIT关键字会影响数据

SELECT语句执行过程

单表查询结构

```
SELECT
FROM
WHERE
AND
GROUP BY
HAVING
ORDER BY
LIMIT
```

关键字的顺序不能颠倒：SELECT...FROM...WHERE...GROUP BY...HAVING...ORDER BY...LIMIT...
SELECT语句的执行顺序：FROM->WHERE->GROUP BY->HAVING->SELECT字段->ORDER BY->LIMIT

标准SQL语句不允许在WHERE子句引用列别名，这个限制是由于在评估WHERE子句时，列的值可能尚未确定
别名可以在查询的选择列表中为列指定不同的名称，可以在GROUP BY,ORDER BY或HAVING子句中使用别名来引用列
MySQL8.0+版本进行了一些优化，WHERE后面依然无法使用SELECT产生的列，经过优化和中间的虚拟表，GROUP BY、ORDER BY和HAVING后面可以使用SELECT产生的列

运算符

算术运算符

主要用于数学计算，其可以连接运算符前后的两个数值或表达式，对数值进行加减乘除取模运算

运算符	描述
%,MOD	模运算符
*	乘法运算符
+	加法运算符
-	减法运算符
/	浮点除法
DIV	整数除法

运算符表达式可以应用在SELECT列位置或者WHERE条件后
乘法如果有浮点数会保留浮点类型，参数都是整数结果为整数，如果出现除以0的情况，通常会返回NULL而不是报错
%和MOD都是关系型数据库的标准用法，二者等价

比较运算符

名字	描述
>	大于运算符
>=	大于等于运算符
<	小于运算符
<=	小于等于运算符
<>,!=	不等于运算符（!=非标准）
=	相等运算符
<=>	NULL安全等于运算符（非标准），如果值是NULL则是true
IS NULL	NULL值测试
IS NOT NULL	NOT NULL值测试
BETWEEN .. AND ..	值是否在范围内

名字	描述
NOT BETWEEN .. AND ..	值是否不在范围内
IN()	值是否在一组值中
NOT IN()	值是否不在一组值中
LIKE	简单的模式匹配(模糊等于)
NOT LIKE	否定简单的模式匹配

比较运算符的结果为1、0、NULL，1代表true，0和NULL代表false，适用于数字和字符串
根据需要，字符串会自动转化为数字，数字也会自动转化为字符串，即'1'=1为true
<=>在=的基础上，添加了NULL判断
如果=两边都是字符串，就按照字符串比较，不会转化成数字
字符串是按照ASCII码编码顺序，BETWEEN AND语句固定第一个是min，第二个是max，不会自动转化，有字符串和数字会自动值转化
NOT IN和IN用法一样，只是范围相反
LIKE运算符主要用来匹配字符串，适用于模糊匹配，如果成功返回1，否则返回0，如果给定的值或者匹配条件为NULL，则返回结果为NULL
如果要查询的集合是一个set，那么可以使用LIKE模糊匹配来查询包含某个元素的结果，或者也可以直接调用FIND_IN_SET('值',列名)函数查询值是否出现，如果出现函数返回1，否则返回0
LIKE运算符通常使用如下通配符：

- %：匹配0个或多个字符
- _：只匹配一个字符

多列一次对比

```
(a,b)=(x,y)等同于(a=x) AND (b=y)
(a,b)<=>(x,y)等同于(a<=>x) AND (b<=>y)
(a,b)<>(x,y)等同于(a<>x) AND (b<>y)
```

逻辑运算符

逻辑运算符主要用来判断表达式的真假，在MySQL中，逻辑运算符的返回结果为1、0或者NULL

逻辑运算符	描述
AND,&&	逻辑与
NOT,!	否定值
OR,||	逻辑或
XOR	逻辑异或

MySQL将任何非0、非NULL值计算为True，反之为False
逻辑运算符主要做多条件结果运算

运算符优先级

编号越大优先级越高，优先级高先运算

优先级	运算符
1	:=,=(赋值)
2	||,OR,XOR
3	&&,AND
4	NOT
5	BETWEEN,CASE,WHEN,THEN,ELSE
6	=(比较运算符),<=>,>=,>,<=,<,<>,!=,IS,LIKE,REGEXP,IN
7	|
8	&
9	<<,>>
10	-,+
11	*,/,DIV,%,MOD
12	^
13	-(负号),~(按位取反)
14	!

在实际使用中如果不确定优先级，可以使用`括号`来明确运算顺序，括号优先级最高

单行函数与多行函数

SQL函数分为内置函数和自定义函数

在内置函数中分为单行函数和多行函数
单行函数：对一行中的某列操作函数，返回结果是单一值

- 数字函数
- 日期函数
- 字符函数
- 流程函数
- 信息函数
- 时间函数

多行函数：对多行中的某列操作函数，返回结果是单一值

- 聚合函数

数值函数

函数	用法
`ABS(x)`	返回x的绝对值
SIGN(x)	返回x的符号，正数返回1，负数返回-1，0返回0
PI()	返回圆周率的值
`CEIL(x)`, `CEILING(x)`	返回大于或大于等于某个值的最小整数
`FLOOR(x)`	返回小于或小于等于某个值的最大整数
LEAST(e1,e2,...)	返回列表中的最小值
GREATEST(e1,e2,...)	返回列表中的最大值
MOD(x,y)	返回x模y的结果
RAND()	返回[0,1)的随机值
RAND(x)	返回[0,1)的随机值，其中x的值作为种子值，相同的x值会产生相同的随机数
`ROUND(x)`	返回一个对x的值进行四舍五入后，最接近于x的整数
`ROUND(x,y)`	返回一个对x的值进行四舍五入后最接近于x的值，并保留到小数点后y位
`TRUNCATE(x,y)`	返回数字x截断后y位小数的结果
SQRT(x)	返回x的平方根，当x的值为负数时，返回NULL

字符串函数

函数	用法
`CHAR_LENGTH(s)`	返回字符串s的字符数，作用与CHARACTER_LENGTH(s)相同
`LENGTH(s)`	返回字符串s的字节数，和字符集有关
`CONCAT(s1,s2,...,sn)`	连接s1,s2,..,sn为一个字符串
INSERT(str,idx,len,replacestr)	将字符串str从第idx位置开始，len个字符长的子串替换为字符串replacestr
REPLACE(str,a,b)	用字符串b替换字符串str中所有出现的字符串a
UPPER(s)或UCASE(s)	将字符串s的所有字母转换为大写字母
LOWER(s)或LCASE(s)	将字符串s的所有字母转换为小写字母
LEFT(str,n)	返回字符串str最左边的n个字符
RIGHT(str,n)	返回字符串str最右边的n个字符

函数	用法
TRIM(s)	去掉字符串s开始与结尾的空格
`SUBSTR(s,index,len)`	返回从字符串s的index位置取len个字符，作用与SUBSTRING(s,n,len)、MID(s,n,len)相同
`FIND_IN_SET(s1,s2)`	返回字符串s1在字符串s2中出现的位置。其中字符串s2是一个以逗号分隔的字符串
REVERSE(s)	返回s反转后的字符串
NULLIF(value1,value2)	比较两个字符串，如果value1和value2相等，则返回NULL，否则返回value1

时间函数

函数	用法
`CURDATE()`,CURRENT_DATE()	返回当前日期，只包含年月日
CURTIME(),CURRENT_TIME()	返回当前时间，只包含时分秒
`NOW()`,SYSDATE()	返回当前系统日期和时间
UTC_DATE()	返回UTC(世界标准时间)日期，不考虑时区
UTC_TIME()	返回UTC(世界标准时间)时间，不考虑时区
`YEAR(date),MONTH(date),DAY(date)`	返回具体的日期值
`HOUR(time),MINUTE(time),SECOND(time)`	返回具体的时间值
`MONTHNAME(date)`	返回月份：January,...
`DAYNAME(date)`	返回星期几：MONDAY,TUESDAY...
`WEEKDAY(date)`	返回周几，周1是0，周2是1...
`QUARTER(date)`	返回日期对应的季度，范围是[1,4]
`WEEK(date),WEEKOFYEAR(date)`	返回一年中的第几周
`DAYOFYEAR(date)`	返回日期是一年中的第几天
`DAYOFMONTH(date)`	返回日期位于所在月份的第几天
`DAYOFWEEK(date)`	返回日期是一周的第几天，周日是1，周一是2，...，周六是7
`DATE_ADD(date,INTERVAL expr type),ADDDATE(date,INTERVAL expr type)`	返回与给定日期时间相差INTERVAL时间段的日期时间，INTERVAL是关键字，expr是相差时间，type是年月日的英文大写

函数	用法
`DATE_SUB(date,INTERVAL expr type),SUBDATE(date,INTERVAL expr type)`	返回与date相差INTERVAL时间间隔的日期，INTERVAL是关键字，expr是相差时间，type是年月日的英文大写
ADDTIME(time1,time2)	返回time1加上time2的时间，time2是一个数字时代表的是秒，可以为负数
`SUBTIME(time1,time2)`	返回time1减去time2的时间，time2是一个数字时代表的是秒，可以为负数
`DATEDIFF(date1,date2)`	返回date1-date2的日期间隔天数
TIMEDIFF(time1,time2)	返回time1-time2的时间间隔
FROM_DAYS(N)	返回从0000年1月1日起，N天后的日期
TO_DAYS(date)	返回从date距离0000年1月1日的天数
LAST_DAY(date)	返回date所在月份的最后一天的日期
MAKEDATE(year,n)	针对给定年份与所在年份中的天数返回一个日期
MAKETIME(hour,minute,second)	将给定的小时、分钟、秒组成的时间返回
DATE_FORMAT(date,fmt)	按照字符串fmt格式化日期date值，date必须是标准的日期格式
TIME_FORMAT(time,fmt)	按照字符串fmt格式化时间time值，time必须是标准的时间格式
STR_TO_DATE(str,fmt)	按照字符串fmt对str进行解析，解析为一个标准的日期格式

格式符	说明	格式符	说明
`%Y`	4位数字表示年份	%y	表示两位数字的年份
%M	月名表示月份(January...)	`%m`	两位数字表示月份(01,02,...)
%b	缩写的月名(Jan,Feb,...)	%c	数字表示月份(1,2,3,...)
%D	英文后缀表示月中的天数(1st,2nd,3rd,...)	`%d`	两位数字表示月中的天数(01,02,...)
%e	数字形式表示月中的天数(1,2,3,4,5)		
`%H`	两位数字表示小时，24小时制(01,02,...)	%h和%l	两位数字表示小时，12小时制(01,02,...)
%k	数字形式的小时，24小时制(1,2,3,...)	%l	数字形式表示小时，12小时制(1,2,3,4,...)

格式符	说明	格式符	说明
%i	两位数字表示分钟(00,01,02,...)	%S 和%s	两位数字表示秒(00,01,02...)
%W	一周中的星期名称(Sunday,...)	%a	一周中的星期缩写(Sun,Mon,Tue,..)
%w	以数字表示周中的天数 (0=Sunday,1=Monday,...)		
%j	以3位数字表示年中的天数 (001,002,...)	%U	以数字表示年中的第几周，(1,2,3...其中Sunday为周中的第一天)
%u	以数字表示年中的第几周(1,2,3,...其中Monday为周中的第一天)		
%T	24小时制	%r	12小时制
%p	AM或PM	%%	表示%

流程函数

流程处理函数可以根据不同的条件，执行不同的处理流程，可以在SQL语句中实现不同的条件选择，MySQL中的流程处理函数主要包括IF()、IFNULL()和CASE()函数

- IF函数是一种条件函数，用于在SQL查询中执行基本的条件判断

```
IF(conditin,true_value,false_value)
# 当condition成立时，返回true_value，否则返回false_value
```

- IFNULL函数是MySQL中的一个函数，用于处理NULL值

```
IFNULL(column,null_value)
# 当指定列column值为null，取null_value的值作为结果
```

- CASE表达式用于实现多条件判断，并根据条件的结果返回不同的值

```
# 格式1
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  ...
  ELSE default_result
END [AS alias_name]
# 这种形式中，WHEN子句后面跟着一个条件，而不是一个具体的值
# 格式2
CASE expr
  WHEN value1 THEN result1
  WHEN value2 THEN result2
  ...
  ELSE default result
END [AS alias_name]
```

```
# 这种形式中，expr是要比较的表达式或列名，然后逐个与WHEN子句中的值进行比较
# [AS alias_name]可写可不写，当你查询的结果作为一个新的列，可以在alias_name处改成新的列名
```

聚合函数

聚合函数作用于全部或分组数据进行统计和计算，最终返回一条结果

函数	用法
AVG(列名)	计算某一列的平均值(数值类型)
SUM(列名)	计算某一列的和(数值类型)
MIN(列名)	计算某一列的最小值(任意类型)
MAX(列名)	计算某一列的最大值(任意类型)
COUNT(列名/*/1)	计算某一列或行的记录数(任意类型)，如果括号里是列名，则返回列中非空值出现的次数，如果是*或者1则返回总条数

聚合函数不能嵌套使用，不能出现类似于AVG(SUM(字段名称))形式的调用

数据库约束

约束概念

表级别的规定，数据的限制语法

约束作用

确保表数据的准确性、可靠性和正确性

添加时机

- 创建表时直接添加
- 创建表之后，通过ALTER TABLE语句添加

约束分类

- 域(列)级约束
  此类约束只对当前列值有效，如某列不能是NULL
  - 非空约束：NOT NULL，列非空约束
  - 默认值约束：DEFAULT，某列默认值
  - 检查约束：CHECK
- 实体(行)级约束
  此类约束需要对比同一表中其他行数据才有效果，如某列值必须唯一
  - 主键约束：PRIMARY KEY，主键唯一且不为空约束
  - 唯一约束：UNIQUE，限制某一列值表中唯一
  - 自增长约束：AUTO_INCREMENT，数字类型字段插入数据自增长约束
- 引用(多表)级约束
  此类约束需要对比其他表的列才有效果，如分数表中引用学生表学号
  - 参照(外键)约束：FOREIGN KEY，限制表中某一列，正确引用其他表的数据值

非空约束

作用

限定某个字段/列的值不为空

关键字

```
NOT NULL
```

特点

- 默认：所有类型列默认都可以为NULL，包括数字类型
- 列上添加：非空约束只能添加到列上
- 多次使用：一个表可以有很多列进行非空限定
- 空值判断：空字符串不算NULL，0也不算NULL

添加

- 建表时添加

```
CREATE TABLE 表名称(
  字段名 数据类型,
  字段名 数据类型 NOT NULL,
  字段名 数据类型 NOT NULL
);
```

- 建表后修改

```
ALTER TABLE 表名 MODIFY 字段名 数据类型 NOT NULL;
```

删除

```
ALTER TABLE 表名 MODIFY 字段名 数据类型 NULL;
```

```
ALTER TABLE 表名 MODIFY 字段名 数据类型;
```

不写NULL默认可以为NULL

默认值约束

作用

限定某个字段/某列的添加默认值

关键字

```
DEFAULT 默认值
```

特点

- 位置约定：默认值约束不能添加到唯一或者主键上，其他列都可以
- 生效时间：当插入数据时，没有显示赋值，赋予默认值，如果没有非空约束，插入NULL时不会赋予默认值，而是赋予NULL
- 细节特点：添加约束时，DEFAULT 默认值，默认值对应正确数据类型

添加

- 建表时添加

```
CREATE TABLE 表名(
  字段名 数据类型 DEFAULT 默认值,
  字段名 数据类型 NOT NULL DEFAULT 默认值
);
```

- 建表后修改

```
ALTER TABLE 表名 MODIFY 字段名 数据类型 DEFAULT 默认值;
# 保留非空约束写法：
ALTER TABLE 表名 MODIFY 字段名 数据类型 DEFAULT 默认值 NOT NULL;
```

删除

```
# 删除默认值约束，也不保留非空约束
ALTER TABLE 表名 MODIFY 字段名 数据类型;
#删除默认值约束，保留非空约束
ALTER TABLE 表名 MODIFY 字段名 数据类型 NOT NULL;
```

检查约束

作用

检查某个字段的值是否符合要求，一般指的是值的范围

关键字

```
CHECK (表达式)
```

特点

- 新特性：5.7版本不支持CHECK约束，8+版本才支持
- 万能约束：CHECK (表达式)，可以自定义表达式，变成任何约束
- 不推荐：不推荐使用CHECK约束，进行数据检查建议使用程序级限制

添加

- 建表时添加

```
CREATE TABLE 表名(
  字段名 数据类型,
  CHECK (表达式), # CHECK约束属于表级别，不能添加到列后
  字段名 数据类型 NOT NULL DEFAULT 默认值
);
```

- 建表后修改

```
ALTER TABLE 表名 ADD CONSTRAINT 约束名 CHECK (表达式);
```

删除

```
ALTER TABLE 表名 DROP CONSTRAINT 约束名;
```

查看约束

```
SELECT *
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_SCHEMA = '你要查询的数据库名字'
AND TABLE_NAME = '你要查询的表名字';
```

INFORMATION_SCHEMA.TABLE_CONSTRAINTS是数据库默认存放约束的一个库

唯一约束

作用

限定某个字段或者组合字段在表中的数据唯一

关键字

```
UNIQUE
```

特点

- 约束数量：同一个表可以有多个唯一约束
- 空值处理：唯一约束允许列值为空，即允许为NULL
- 约束名称：在创建唯一约束的时候，如果不给唯一约束命名，就默认和列名相同

添加

- 建表时添加

```
CREATE TABLE 表名(
  字段名 数据类型 UNIQUE
);

CREATE TABLE 表名(
  字段名 数据类型 UNIQUE KEY
);

CREATE TABLE 表名(
  字段名 数据类型,
  [CONSTRAINT 约束名] UNIQUE KEY(字段名,...)
);
```

- 建表后修改

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE(列名,列名,...);
```

删除

```
ALTER TABLE 表名 DROP CONSTRAINT constraint_name;
```

主键

在任何情况下，确保表中的行数据至少有一列是不重复的且不为NULL，那么这一列成为主键或主键列

分类

- 自定义主键：人为创建的一列，专门用来做主键，它的使命就是保证行数据不重复，如学号、身份证号
- 自然主键：不是人为创建的列，数据实体自带的属性列，当前环境下此属性列唯一且不为空，可以做主键，如DNA序列

`推荐使用自定义主键`

细节说明

- 主键数量：每个表只能有“一个”主键
- 单一和复合：主键可以由单个或多个列构成
- 主键列类型：可以是任意类型，只要唯一且不为NULL即可
- 主键命名：主键一般采用identify(标识)单词缩写xx_id等，没有强制要求
- 主键索引：创建主键约束时，系统默认会在所在的列或列组合上创建对应的主键索引(能够根据主键查询的就根据主键查询，效率更高)，如果删除了主键约束，主键约束对应的索引就自动删除，主键索引固定命令：PRIMARY

主键约束

主键约束是对主键的一种保护限制和约束
没有主键约束的主键也是主键，但是数据安全性得不到保证

添加

- 建表时添加

```
CREATE TABLE 表名(
  字段名 数据类型 PRIMARY KEY # 列级模式
);

CREATE TABLE 表名(
  字段名 数据类型,
  [CONSTRAINT 约束名] PRIMARY KEY (字段名,...) # 表级模式
);
```

- 建表后修改

```
ALTER TABLE 表名 ADD PRIMARY KEY (字段名,字段名,...);
# 主键是一个或多个字段组成的
```

删除

删除主键约束不需要指定主键名，因为一个表只有唯一的主键，删除主键约束后非空还在，唯一消失，也就是说删除后主键不为空，但可能重复

```
ALTER TABLE 表名 DROP PRIMARY KEY;
```

自增长约束

作用

限定某个整数类型字段，插入数据不显式维护，值自动增长
自增长每次都是从最大值的基础上进行增长，就算删除，也不会回退，如果你指定的值远大于以前自增长的值，那么再进行插入时从你指定的值开始自增长，中间没有出现的值全部空过去

关键字

```
AUTO_INCREMENT
```

特点

- 添加位置：只能添加到键值列(主键列和唯一列)，普通列不可以
- 约束数量：每一张表只能有一个自增长约束
- 数据类型：增加自增长约束的列必须是整数类型
- 特殊情况：如果给自增长字段设置0或者NULL，列数据会自增长赋值，如果设置的是非0或非空数据，那么将设置你指定的值

添加

- 建表时添加

  ```
  CREATE TABLE 表名(
    字段名 数据类型 PRIMARY KEY AUTO_INCREMENT
  );

  CREATE TABLE 表名(
    字段名 数据类型 UNIQUE KEY AUTO_INCREMENT
  );
  ```

- 建表后修改

  ```
  ALTER TABLE 表名 MODIFY 字段名 数据类型 AUTO_INCREMENT;
  # 主键列和唯一列不需要再指定，不会被覆盖
  ```

删除

```
ALTER TABLE 表名 MODIFY 字段名 数据类型;
# 去掉AUTO_INCREMENT相当于删除，主键列和唯一列不需要再指定，不会被覆盖
```

外键(参照|引用)约束

- 外键：引用或参照其他表主键列值的列，称为外键，外键的值范围应当对应引用的主键的值范围
- 外键约束：外键应该引用主键的值，如果不添加约束，可能会出现错误数据，
- 外键数量：每个表可以包含多个外键
- 外键跨表：外键是跨表引用其他表的主键，被引用表为主表(学生表)，外键表为子表(分数表)，创建时先创建主表，再创建子表
- 外键类型：外键类型不能是任意类型，应该和主键类型对应，尽量命名相同
- 主外键关系：关系型数据库，关系指的是主外键关系，有主外键的两张表可以水平联查
- 其他影响：存在主外键关系(外键约束)，删除主表数据时，可能会因为子表引用而删除失败，需要先删除子表的所有引用数据再删除主表数据，若没有引用则可以直接删除

添加

- 建表时添加

```
CREATE TABLE 主表名称(
  字段名 数据类型 PRIMARY KEY
);
CREATE TABLE 子表名称(
  字段名 数据类型 PRIMARY KEY,
  [CONSTRAINT 外键约束名称] FOREIGN KEY(外键名) REFERENCES 主表名(主键名) [ON
UPDATE ..][ON DELETE ..]
);
```

- 建表后修改

```
ALTER TABLE 子表名 ADD [CONSTRAINT 约束名] FOREIGN KEY(外键名) REFERENCES 主表名
(引用字段名) [ON UPDATE ..][ON DELETE ..];
```

删除

```
# 第一步先查看约束名和删除外键约束
SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS WHERE table_name='表名';
ALTER TABLE 子表名 DROP FOREIGN KEY 外键约束名;
# 第二步查看索引名和删除索引(只能手动删除)
SHOW INDEX FROM 表名;
ALTER TABLE 子表名 DROP INDEX 索引名;
```

约束等级设计

CASCADE	在父表上UPDATE/DELETE记录时，同步UPDATE/DELETE子表的匹配记录
SET NULL	在父表上UPDATE/DELETE记录时，将子表上匹配的记录的列设为空，但是要注意子表的外键列不能为非空
NO ACTION	如果子表中有匹配的记录，则不允许父表对应的键进行UPDATE/DELETE操作
RESTRICT(默认)	同NO ACTION,都是立即检查外键约束

SET DEFAULT	父表有变更时，子表将外键列设为一个默认值，但是有些驱动不识别

最好采用的是:ON UPDATE CASCADE ON DELETE RESTRICT的方式

例

当主表中的行被删除或更新时，子表中的相关行也被删除或更新

```
ALTER TABLE 子表名 ADD CONSTRAINT 外键约束名 FOREIGN KEY(子表外键列名) REFERENCES 主表名(主表主键列名) ON UPDATE CASCADE ON DELETE CASCADE;
```

约束常见面试题

- 建和不建外键约束有什么区别
  - 建外键约束你的操作(创建表、删除表、添加、修改、删除)会受到限制，从语法层面受到限制，如在员工表中不可能添加一个员工信息，它的部门的值在部门表中找不到。不建外键约束，操作不会受到限制。要保证数据的引用完整性只能靠程序员的自觉，或者在代码中进行限定，如可以在员工表中添加一个员工信息，而它的部门指定为一个完全不存在的部门
- 建和不建外键约束对查询有什么影响
  - 在语法层面没有影响，添加约束可能会影响查询速度和效率
- 表中字段为什么不想要NULL的值
  - 不好比较，NULL是一种特殊值，比较时只能用专门的IS NULL或IS NOT NULL来比较，碰到运算符通常返回NULL
  - 效率不高，影响提高索引效果，我们往往在建表时设置NOT NULL DEFAULT ''或DEFAULT 0(建立好非空约束)
- 带AUTO_INCREMENT约束的字段值是从1开始的吗
  - 在MySQL中，默认AUTO_INCREMENT的初始值是1，每新增一条记录，字段值自动加1，设置自增约束的时候，还可以指定第一条插入记录的自增字段的值，这样新插入的记录的自增字段的值从初始值(如果没有最大值从1开始，有最大值从最大值+1开始)开始递增。添加主键约束时，往往需要设置字段自动递增属性。此外可以在创建表的时候指定自增长起始值
- 是不是每个表都可以任意选择存储引擎
  - 外键约束(FOREIGN KEY)不能跨引擎使用。MySQL中支持多种存储引擎，每一个表都可以指定一个不同的存储引擎，需要注意的是：外键约束是用来保证数据的参照完整性的，如果表之间需要关联外键，却指定了不同的存储引擎，那么这些表之间不能创建外键约束。
- 请问你会不会创建数据库时给你的表添加完备的约束
  - 不会，一般情况下，只添加一些单表的约束(实体约束和域约束)，不会添加外键和级联操作。根据阿里开发规范，强制不得使用外键和级联，一切外键概念必须在应用层解决。外键与级联更新适用于单机低并发，不适合分布式、高并发集群，级联更新是强阻塞，存在数据库更新风暴的风险，外键影响数据库的插入操作。如学生表的id是主键，成绩表的id为外键，如果更新学生表的id，同时触发成绩表id的更新，即为级联更新

数据库多表关系

拆表优势

- 提高查询效率：分表存储不同数据类型可以根据数据特征和查询模式进行优化
- 防止数据冗余：将不同数据类型存储在不同的表中可以避免数据冗余
- 更方便进行数据管理和维护：分表存储不同数据类型可以简化数据管理和维护工作

多表关系概念

- 关系型数据库中，数据都是按照类别存储到对应的表结构
- 表和表之间可以通过`主外键`建立表(数据)之间关系
- 拆表存储可以减少数据冗余，并提高数据查询和操作的效率
- 因为拆表存储的原因，我们不仅要掌握`单表的查询语法`，还需要掌握`多表关联查询语法`
- 关系型数据库需要`提前创建库表结构`，需要`通过分析数据关系在建表添加合理约束`

多表关系

多：实际上就是两两关系
表：实际上就是数据之间的关联

- 一对一：两个表之间的每行数据都是唯一的对应关系，如一个员工对应一个员工档案
    - 特点：一条员工数据至多对应一条档案数据，一条档案数据至多对应一条员工数据
- 一对多：一个表关联另一个表的多行数据，反方向只关联一个数据，如一个作者对应多篇文章
    - 特点：一条作者数据对应多条文章数据，一条文章数据至多对应一条作者数据
- 多对多：两个表中的记录都可以与对方表中的多个记录相关联，如学生和课程之间，一个学生可以选多门课，一门课可以被多个学生选择
    - 特点：多对多关系需要创建中间表，存储多个外键来建立两张表的关联关系

一对一表关系

- 解决冗余：一对一并不能解决数据冗余问题
- 存在意义：一张表有冷热数据，将表分成两张表存储，一张存热数据，一张存冷数据
- 关系维护：子表外键对应的数据不能重复(外键唯一)，子表主外键进行融合，两张表共用同一个主键或者子表外键添加唯一约束

例

主表：员工表

```
CREATE TABLE emp(
  e_id INT PRIMARY KEY AUTO_INCREMENT,
  e_name VARCHAR(20) NOT NULL,
  e_age INT DEFAULT 18,
  e_gender CHAR DEFAULT '男'
);
```

子表：档案表(唯一外键)

```
CREATE TABLE profile(
  p_id INT PRIMARY KEY AUTO_INCREMENT,
  p_address VARCHAR(100) NOT NULL,
  p_level INT DEFAULT 10,
  e_id INT UNIQUE, # 唯一外键
  CONSTRAINT s_p_1 FOREIGN KEY(e_id) REFERENCES emp(e_id)
);
```

子表：档案表(共用主键)

```
CREATE TABLE profile(
  e_id INT PRIMARY KEY,
  p_address VARCHAR(100) NOT NULL,
  p_level INT DEFAULT 10,
  CONSTRAINT s_p_2 FOREIGN KEY(e_id) REFERENCES emp(e_id)
);
```

一对多表关系

- 解决冗余：一对多可以解决数据冗余问题
- 存在意义：一个实体对应多个子元素，通过分表解决数据冗余问题和提高数据操作效率
- 关系维护：子表外键对应的数据可以重复，外键不唯一

例

主表：作者表

```
CREATE TABLE author(
  a_id INT PRIMARY KEY AUTO_INCREMENT,
  a_name VARCHAR(20) NOT NULL,
  a_age INT DEFAULT 18,
  a_gender CHAR DEFAULT '男'
);
```

子表：文章表(外键)

```
CREATE TABLE blog(
  b_id INT PRIMARY KEY AUTO_INCREMENT,
  b_title VARCHAR(100) NOT NULL,
  b_content VARCHAR(600) NOT NULL,
  a_id INT, # 外键
  CONSTRAINT a_b FOREIGN KEY(a_id) REFERENCES author(a_id)
);
```

多对多关系

- 特殊情况：多对多需要创建中间表建立数据之间的关联
- 关系维护：中间表需要包含两个外键，主表数据之间间接关联

例

主表：学生表

```
CREATE TABLE student(
  s_id INT PRIMARY KEY AUTO_INCREMENT,
  s_name VARCHAR(20) NOT NULL,
  s_age INT DEFAULT 18
);
```

中间表：stu_course

```
CREATE TABLE stu_course(
  sc_id INT PRIMARY KEY AUTO_INCREMENT,
  s_id INT,
  c_id INT,
  FOREIGN KEY(s_id) REFERENCES student(s_id),
  FOREIGN KEY(c_id) REFERENCES course(c_id)
);
```

主表：课程表

```
CREATE TABLE course(
  c_id INT PRIMARY KEY AUTO_INCREMENT,
  c_name VARCHAR(10) NOT NULL,
  c_teacher VARCHAR(10)
);
```

多表查询语法

多表查询的重点是将多张表数据利用`多表查询语法`合并成单张虚拟表，然后进行条件和高级查询处理等
按照多表结果合并的方向分为**水平合并语法**和**垂直合并语法**

- 垂直合并语法(使用较少)
    - 语法:union/union all
    - 作用：将多个表结果汇总
    - 主外键：不要求表之间有主外键关系，只要求列数和类型对应
- 水平合并语法(使用较多)
    - 语法：连接查询
    - 作用：将多张表结果水平整理
    - 主外键：要求表之间有主外键关系

垂直合并语法

语法

```
UNION # 合并记录同时去掉重复数据
UNION ALL # 合并记录且不去掉重复数据
```

```
# 垂直合并表a和表b的数据，去重复数据
SELECT aid, aname FROM a
UNION
SELECT bid, bname FROM b;

# 垂直合并表a和表b的数据，不去重复数据
SELECT aid, aname FROM a
UNION ALL
SELECT bid, bname FROM b;

# 如果后面还有数据直接在后面接着加 UNION/UNION ALL SELECT....即可
```

细节

- 实现要求：只要求合并的结果集之间的列数和对应列的数据类型相同即可
- 主外键要求：UNION只是结果集垂直汇总，不涉及行数据水平连接，不要求有主外键关系
- 重复数据认定：一行中的所有列值都相同，认定为重复行

场景

如果想要汇总来自不同表集的数据时，可以使用垂直合并语法

- 合并多张相似表的数据
- 统计不同源的数据

水平合并语法

语法

- 内连接

  ```
  FROM 表1 [INNER] JOIN 表2 ON 表1.主键 = 表2.外键; # (关系型数据库通用标准语法)
  FROM 表1, 表2 WHERE 表1.主键 = 表2.外键; # (MySQL支持的语法)
  ```

- 外连接

  ```
  FROM 表1 LEFT或RIGHT [OUTER] JOIN 表2 ON 表1.主键 = 表2.外键;
  ```

- 自然连接

  ```
  FROM 表1 NATURAL [LEFT或RIGHT] JOIN 表2;
  ```

细节

- 核心要求：水平连接是行和行数据连接，要求两个表必须有主外键关系
- 正确连接：为了正确的将行数据之间连接，水平连接需要额外判定主外键相等
- 拆表产物：因为关系型数据库特点，数据进行拆表存储，所以水平连接语法非常重要
- 语法理解：只需要理解内外连接区别即可

内连接查询语法

内连接(inner join)是一种用于从两个或多个表中检索数据的查询方式，内连接会根据指定的条件，将两个表中满足条件的行进行匹配，并返回匹配成功的行

语法

```
FROM 表1 [INNER] JOIN 表2 ON 表1.主键 = 表2.外键; # (关系型数据库通用标准语法)
FROM 表1, 表2 WHERE 表1.主键 = 表2.外键; # (MySQL支持的语法)

# 如果有N张表合并，至少要有N-1对主外键关系
FROM 表1 [INNER] JOIN 表2 ON 表1.主键 = 表2.外键
         [INNER] JOIN 表3 ON 主键 = 外键
         [INNER] JOIN 表4 ON 主键 = 外键
         ...
         WHERE 其他条件;

FROM 表1, 表2, ... WHERE 表1.主键 = 表2.外键 AND ...(N-1对主外键关系) AND 其他条件;
```

细节

- 以上两种语法功能和效果相同，推荐使用标准语法
    - 两者效果一致，内连接的特点就是两个表必须满足主外键相等，方可返回数据
- 为了避免错误的数据连接，连接查询必须添加主外键相等条件
    - 连接查询就是将所有行数据相互拼接一次，如果不添加主外键相等条件，相当于两个表数据进行笛卡尔乘积，一定存在正确数据，但不一定所有数据都是正确的
- 多表查询要考虑不同表存在相同字段名的问题
    - 多表中很大概率存在相同名称的字段，最好使用表名.列名，如果表名较长可以给表名起别名，方便拼写，语法：FROM 表名 别名... 或FROM 表名 as 别名...，这样就可以使用表别名.列名进行查询

外连接查询语法

外连接(outer join)是一种用于从两个或多个表中检索数据的查询方式，与内连接不同的是，外连接会返回所有符合条件的行，同时也返回未匹配的行，外连接分为左外连接和右外连接
在开发过程中绝大部分情况都需要用到逻辑主表，外连接使用率更高

语法

```
SELECT * FROM 表1 LEFT [OUTER] JOIN 表2 ON 表1.主键 = 表2.外键;# 表1为逻辑主表
SELECT * FROM 表1 RIGHT [OUTER] JOIN 表2 ON 表1.主键 = 表2.外键;# 表2为逻辑主表
```

细节

- 内外连接区别
    - 内连接：只有满足匹配条件的行才会返回，两个表必须存在主外键，且主外键值相等
    - 外连接：可以通过左和右定义逻辑主表，逻辑主表的数据一定会返回
- 外连接连续性：建议将逻辑主表放在表1的位置，这样无论后面还有多少表都可以直接使用LEFT JOIN把表1定义为它们共同的逻辑主表

内外连接总结

- 内连接需要添加主外键判定，去除笛卡尔积问题
- 内连接属于`公平`查询语法，双方都满足主外键相等方可查询
- 外连接属于`不公平`查询语法，可以指定一方为逻辑主表，逻辑主表的数据一定会查询出来
- 一般情况下主外键的条件添加到ON后，其他条件仍然使用WHERE
- 如果有逻辑主表，建议放在最左侧，后续可以全部使用左外连接
- 多表联合查询，关联外键条件个数为N-1

多表查询关键字顺序

```
SELECT ...
FROM ...
LEFT或RIGHT JOIN ON
WHERE AND
GROUP BY ...
HAVING
ORDER BY ... ASC或DESC
LIMIT ...
```

自然连接查询语法

自然连接(natural join)是一种内连接和外连接的升级版，会自动找到两个表中相同的列名判定相等，可以省列ON 主 = 外语法，但除了主外键的其他列名相等它也会判定，较为繁琐，不推荐使用

语法

```
SELECT * FROM 表1 NATURAL JOIN 表2;# 自然内连接
SELECT * FROM 表1 NATURAL LEFT JOIN 表2;# 自然左外连接
SELECT * FROM 表1 NATURAL RIGHT JOIN 表2;# 自然右外连接
```

细节

- 自然连接就是内外连接合并版本，使用自然连接要求两个表主外键命名相同，除了主外键以外的列命名不同
- 自然连接可以指定判定哪些列，如SELECT * FROM 表1 JOIN 表2 USING(列名); 使用USING指定数据表里的`同名字段`进行等值连接

自连接查询语法

自连接是指数据库中，一个表与其自身进行连接的操作，它在查询过程中使用相同表的不同列名来表示两个不同的实例，然后通过连接条件将这两个实例进行连接。**自连接只是单表查询的一种使用场景**，自连接仍然使用内外连接语法实现。

例

一个员工表中有员工编号、员工姓名和其领导编号，如果想查询员工的领导的姓名，需要将这张表进行自连接，把领导编号与员工编号进行对应，然后再返回出领导编号下的员工姓名是谁

```
SELECT 表1.员工编号, 表1.员工姓名, 表1.领导编号, 表2.领导姓名 FROM 表1 LEFT JOIN 表2(这里的表2和表1是同一张表) ON 表1.领导编号 = 表2.员工编号
```

查询编号等于5的员工的编号、姓名、领导编号、领导姓名、领导的领导的编号和姓名

```
# 其中表1，表2，表3为同一张表
SELECT 表1.员工编号,表1.员工姓名,表1.领导编号,表2.领导姓名,
表2.领导编号,表3.领导姓名 FROM 表1 LEFT JOIN 表2 ON 表1.领导编号 = 表2.员工编号
LEFT JOIN 表3 ON 表2.领导编号 = 表3.员工编号
WHERE 表1.员工编号 = 5;
```

子查询语法

子查询是指在SQL中嵌套一个完整的SELECT查询语句，这个嵌套的查询通常被称为子查询或内部查询
就是把看似非常复杂的条件查询拆分成若干简单的查询然后进行合并

- 标量子查询：子查询返回单行单列，这个值通常用于条件判断、过滤或者作为SELECT查询的一部分
- 行子子查询：子查询返回一行多列，通常用于更新或插入数据，或者作为一个整体来比较
- 列子子查询：子查询返回一列多行，通常用于IN、ANY、ALL等条件中，或者作为其他查询的条件
- 表子子查询：子查询返回多行多列，通常不能用于查询条件，用于连接或联合查询中的虚拟表

标量子查询

查询的结果是一个值

```
# 查询分数表中大于平均分的学生id，学生姓名和其分数
SELECT student_id, student_name, score FROM t_score WHERE score > (SELECT
AVG(score) FROM t_score);
```

子查询也可以在INSERT、UPDATE或DELETE语句中使用，用于提供额外的条件、过滤结果或进行比较

```
# 将分数表中学生id为5的学生的分数设置为平均分
UPDATE t_score SET score = (SELECT AVG(score) FROM t_score) WHERE student_id =
5;
```

行子子查询

行子子查询一般都是整体比较，查询和给定条件完全相同的行，条件之间是AND关系

```
SELECT * FROM t WHERE (列1, 列2,...) = 或 IN (SELECT 列1, 列2, ... FROM x WHERE 条
件);
```

列子子查询

列子子查询相当于查询满足某些条件的所有行，条件之间是OR关系，只要满足至少一个即可

```
SELECT * FROM t WHERE id IN或NOT IN或= ANY(SELECT did FROM xx WHERE 查询条件);
```

表子子查询

表子查询相当于查出一个虚拟表，然后利用连接查询将结果合并，不能忘了给虚拟表起一个别名

```
SELECT * FROM t LEFT JOIN (SELECT 列名1, 列名2,... FROM t GROUP BY 列名) AS x ON t.
列名 = x.列名;
```

UPDATE嵌套子查询

```
# 将分数表中学生id为5的学生的分数设置为平均分
UPDATE t_score SET score = (SELECT AVG(score) FROM t_score) WHERE student_id =
5;
# 上述代码是错误的，会报错1903，因为外面的修改和里面的查询是同一张表，因为MySQL的保护机制，不让双方同时占用一个表的引用，其中任何一方都不能修改，但可以查询的时候创建一个虚拟表，也就是表子子查询，这样就可以避免被UPDATE操作锁住，这样就变成了两张表互相操作
UPDATE t_score SET score = (SELECT score FROM (SELECT AVG(score) FROM t_score) AS
t) WHERE student_id = 5;
```

DELETE前套子查询

与UPDATE前套子查询类似，外面删除的和里面查询的不能是同一张表，需要额外创建一个虚拟表

INSERT前套子查询

- 创建表y，复制表x的结构，不复制数据

  ```
  CREATE TABLE y LIKE x;
  ```

- 表y复制表x的数据，此时INSERT不用写VALUES，只复制数据

  ```
  INSERT INTO y (SELECT * FROM x);
  # 也可以选择复制哪些结构和数据
  INSERT INTO y (SELECT 列1, 列2,... FROM x);
  ```

- 同时创建表y和复制表x的数据，也就是前两个操作的合并

  ```
  CREATE TABLE y AS (SELECT * FROM x);
  ```