mysql笔记

author: Python杰哥

公众号：python专栏

1.数据库系统中的单元

数据库系统===(1-->n)===>数据库===(1-->n)===>数据表===(1-->n)===>多个字段

数据库系统和数据库之间是一对多的关系

数据库和数据表之间是一对多的关系

数据表和字段之间是一对多的关系

2.数据库相关的操作

通过指令登录数据库

语法：mysql -u用户名 -p密码

示例:

```
mysql -uroot -p
12345678
注意： -p的后面一般情况下，不会直接跟密码，防止密码泄露。
```

查看所有的数据库

语法: show databases;

创建数据库

语法：create database 数据库名

示例:

```
create database test;
```

示例说明:

```
创建一个数据库名字为test的数据库。
```

删除数据库

语法： drop database 数据库名

示例:

```
drop database demo;
```

示例说明:

```
删除一个数据库名字为demo的数据库。
```

选中库

语法: use 数据库名

示例:

```
use test;
```

示例说明:

选中数据库名为test的数据库进行操作。

3.数据表相关的操作

创建数据表

语法: create table 数据表名（字段1 类型，字段2 类型，字段3 类型.....）

示例:

```
create table user(username varchar(100),password char(10));
```

示例说明:

创建一个数据表名为user的数据表，第一个字段为username,类型为varchar 长度为100，第二个字段为password,类型为char,长度为10

查看数据表结构

语法: desc 数据表名字

示例:

```
desc user;
```

示例说明:

查看user数据表的结构。

查看所有的数据表

语法: show tabels;

修改表名

语法: alter table 旧数据表名 rename 新数据表名

示例:

```
alter table user rename new_user;
```

示例说明:

将名字为user的数据表修改为名字为new_user。

删除表

语法: drop table 数据表名;

示例:

```
drop table new_user;
```

示例说明:

```
将数据表new_user从数据库中删除。
```

4.数据表字段的操作

增加表字段

语法: alter table 数据表名字 add 字段名 类型;

示例:

```
alter table user add age int;
```

示例说明:

```
向user表中增加一个age字段。
```

修改表字段

语法: alter table 数据表名字 modify 字段名 类型;

示例:

```
alter table user modify username varchar(20);
```

示例说明:

```
将user表中的username字段修改为varchar(20);
```

删除表字段

语法: alter table 数据表名字 drop 字段名;

示例:

```
alter table user drop username;
```

示例说明:

```
将user表中的username 字段删除。
```

表字段改名

语法: alter table 数据表名 change 字段原名 字段新名 类型;

示例:

```
alter table user change password pwd varchar(20);
```

示例说明:

```
将user表中的password字段修改为pwd,类型为varchar(20);
```

5.数据库中的字段类型

整型:

数据类型	所占字节	取值范围
tinyint	1字节	-128~127
smallint	2字节	-32768~32767
mediumint	3字节	-8388608~8388607
int	4字节	-2147483648~2147483647
bigint	8字节	+-9.22*10的18次方

小数类型（浮点类型和定点类型）:

数据类型	所占字节	取值范围
float(m, d)	4字节	单精度浮点型，m总个数，d小数位
double(m, d)	8字节	双精度浮点型，m总个数，d小数位
decimal(m, d)		定点类型，一般用于精度要求比较高的数据（比如货币等等）

字符类型:

数据类型	所占字节	取值范围
char	0-255字节	定长字符串（规定多少字长必须存储多少字长，超过长度的字段只能截取出对应的长度进行存储，不够长度的字段使用空格补齐。）
varchar	0-655355字节	变长字符串（只要在规定长度内，有多少存多少，无需补齐。超出部分，和char类型一样，舍去即可。）
tinyblob	0-255字节	不超过255个字符的二进制字符串
tinytext	0-255字节	短文本字符串
blob	0-65535字节	二进制形式的长文本数据
text	0-65535字节	长文本数据
mediumblob	0-16 777 215字节	二进制形式的中等长度文本数据
mediumtext	0-16 777 215字节	中等长度文本数据
longblob	0-4 294 967 295字节	二进制形式的极大文本数据
longtext	0-4 294 967 295字节	极大文本数据
varbinary(M)	允许长度0-M个字节	不定长字节字符串，值的长度+1个字节
binary(M)	M	允许长度0-M个字节的定长字节字符串

时间类型：

数据类型	所占字节	取值范围
date	4字节	日期，格式：2014-09-18
time	3字节	时间，格式：08:42:30
datetime	8字节	日期时间，格式：2014-09-18 08:42:30
timestamp	4字节	自动存储记录修改的时间（current_timestamp）
year	1字节	年份1901~2155

复合类型：

数据类型	所占字节	取值范围
set	集合类型	set(“m1”, “m2", … ,“m63")
enum	枚举类型	enum(“m1", “m2", … ,“m65535")

注意：

ENUM 类型因为只允许在集合中取得一个值，有点类似于单选项。在处理相互排拆的数据时容易让人理解，比如人类的性别。ENUM 类型字段可以从集合中取得一个值或使用null值，除此之外的输入将会使 MySQL 在这个字段中插入一个空字符串。另外如果插入值的大小写与集合中值的大小写不匹配，MySQL会自动使用插入值的大小写转换成与集合中大小写一致的值。

SET 类型与 ENUM 类型相似但不相同。SET类型可以从预定义的集合中取得任意数量的值。并且与 ENUM 类型相同的是任何试图在 SET 类型字段中插入非预定义的值都会使MySQL插入一个空字符串。如果插入一个即有合法的元素又有非法的元素的记录，MySQL 将会保留合法的元素，除去非法的元素。比如爱好等。

一个 SET 类型最多可以包含 64 项元素。在 SET 元素中值被存储为一个分离的“位”序列， 这些“位”表示与它相对应的元素。“位”是创建有序元素集合的一种简单而有效的方式。

其他：

auto_increment：自动增加，只用于整型，可以设置起始值，默认为1

常与后面primary key一起使用

创建表时在整型字段后加上：auto_increment=起始值 primary key

修改起始值：alter table user auto_increment=起始值

6.字符集

6-1:字符集是什么?

为了更好的识别中文、日文、英文、希腊语。对于常用的符号进行了编码，这个编码就是字符集。

字符集确定了文字的存储方式。

字符集相当于是计算机中人类的语言。

如果我说的是中文，用英文字符来存储的话。那么人们就看不懂也看不明白，就是我们所说的乱码。

因为字符集太多了，足够有几十种上百种之多。所以我们不需要了解太多的字符集的知识，也不需要了解字符集到底是如何变成人类可见字符的。

6-2:常见的字符集

字符集	说明	字节长度
ASCII	美国标准信息交换代码	单字节
GBK	汉字内码扩展规范	双字节
Unicode	万国码	4字节
UTF8	Unicode的可变长度字符编码	1到6个字节

6-3:实际工作中使用的编码

字符集	说明
gbk_chinese_ci	简体中文, 不区分大小写
utf8_general_ci	Unicode (多语言), 不区分大小写

7.mysql中常见的索引

7-1:索引

索引看着挺高大上的一个名字，说白了就是我们一本书最前面的目录。

假如你用新华字典来查找“张”这个汉字，不使用目录的话，你可能要从新华字典的第一页找到最后一页，可能要花二个小时。字典越厚呢，你花的时间就越多。现在你使用目录来查找“张”这个汉字，张的首字母是z，z开头的汉字从900多页开始，有了这条线索，你查找一个汉字可能只要一分钟，由此可见索引的重要性。

索引用于快速找出在某个列中有一特定值的行。

不使用索引，MySQL必须从第1条记录开始然后读完整个表直到找出相关的行。表越大，花费的时间越多。如果表中查询的列有一个索引，MySQL能快速到达一个位置去搜寻到数据文件的中间，没有必要看所有数据。

当然索引也不易过多，索引越多写入，修改的速度越慢。因为，写入修改数据时，也要修改索引。

创建数据表：

```
create table user (
    id int,
    name varchar(20),
    tel varchar(11),
    email varchar(100)
)
```

查看数据表索引的语句:

```
show index from 数据表名
```

7-2:索引分类

普通索引	**最基本的索引，它没有任何限制**
唯一索引	某一行启用了唯一索引则不准许这一列的行数据中有重复的值。针对这一列的每一行数据都要求是唯一的
主键索引	它是一种特殊的唯一索引，不允许有空值。一般是在建表的时候同时创建主键索引，常用于ID。类似于书中的页码
全文索引	对于需要全局搜索的数据，进行全文索引

7-3:普通索引

索引类型	功能说明
基本语法	alter table 表名 add index(字段)
示例	alter table user add index(name)
示例解释	为user表中的name字段设置为普通索引。

7-4:唯一索引

索引类型	功能说明
基本语法	alter table 表名 add unique(字段)
示例	alter table user add unique(tel)
示例解释	为user表中的tel字段设置为唯一索引。

7-5:全文索引

索引类型	功能说明
基本语法	alter table 表名 add fulltext(字段)
示例	alter table user add fulltext(email)
示例解释	为user表中的email字段设置为全文索引。

7-6:主键索引

索引类型	功能说明
基本语法	alter table 表名 add primary key(字段)
示例	alter table user add primary key(id)
示例解释	为user表中的id字段设置为主键索引。

7-7:创建表时可在创建表语句后加上对应的类型即可声明索引:

primary key(字段)

index (字段)

fulltext (字段)

unique (字段)

示例:

```
create table user1(
    id int auto_increment,
    name varchar(20),
    primary key(id),
    unique(name)
)engine = innodb default charset=utf8;
```

8.mysql数据库sql语句的增删改查

操作前的准备:

```
create table star(
    id int auto_increment,
    name varchar(50) not null,
    money float not null,
    province varchar(30) default null,
    age int unsigned not null,
    sex int not null,
    primary key(id)
) engine = innodb default charset=utf8;
```

查询数据的基本语句:

```
select * from 表名;
```

8.1.1-插入数据:

类别	详细解释
基本语法	insert into 表名 values(值1，值2，值3，值4.....) 注意：值的数量和数据表的字段的数量保持一致。
示例	insert into star values(1,'王宝强',9876,'河北省',41,0);
示例说明	向star表中插入一条数据，（id为1,姓名为'王宝强',money为9876,省份为'河北省',年龄为41,性别为0（0表示男 1表示女））

8.1.2-插入数据:(常用)

类别	详细解释
基本语法	insert into 表名(字段1，字段2，字段3，字段4.....) values(值1，值2，值3，值4.....)
示例	insert into star(name,money,province,age,sex) values('郭德纲',898989,'天津市',51,0);
示例说明	向star表中插入一条数据，（姓名为'郭德纲'，money为898989，省份为天津市，年龄为51，性别为0）

8.1.3-插入数据:

类别	详细解释
基本语法	insert into 表名(字段1，字段2，字段3，字段4.....) values(值1，值2，值3，值4.....),(值1，值2，值3，值4.....),(值1，值2，值3，值4.....),(值1，值2，值3，值4.....);
示例	insert into star(name,money,province,age,sex) values('于谦',87654,'北京市',53,0),('迪丽热巴',765432,'新疆',30,1),('佟丽娅',1765432,'新疆',41,1);
示例说明	向star表中插入3条数据，('于谦',87654,'北京市',53,0),('迪丽热巴',765432,'新疆',30,1),('佟丽娅',1765432,'新疆',41,1)

9.sql语句的查询

9-1:基础查询

类别	详细解释
基本语法	select * from 表名。 * 表示所有的字段
示例	select * from star;
示例说明	查询出star表中的所有的字段。

9-2:指定字段查询

类别	详细解释
基本语法	select 字段名1，字段名2，字段名..... from 表名;
示例	select name,province from star;
示例说明	查询star表中的name和province字段信息;

9-3:指定字段组合不重复记录

类别	详细解释
基本语法	select distinct 字段名 from 表名;
示例	select distinct name from star;
示例说明	查询star表中name字段不重复的记录

9-4:条件查询

类别	详细解释
基本语法	select 字段名1，字段名2..... from 表名 where 条件;

where后可接:

符号	解释
<	小于
>	大于
>=	大于等于
<=	小于等于
!= 或 <>	不等于
=	等于
or	或者
and	并且
between and	在某个闭区间
In / not in	在/不在指定的集合中
like	模糊查询

示例:

类型	示例
= 示例	select * from star where age = 41;
< 示例	select * from star where age < 41;
> 示例	select * from star where age > 41;
>= 或者 <= 示例	select * from star where age >= 41;
!=或者 <> 示例	select * from star where age != 41;
or示例	select * from star where age < 41 or province = '天津市';
and示例	select * from star where age < 41 and province = '河南省';
between and示例	select * from star where id between 2 and 5;
in 示例	select * from star where id in (1,3,5,7);
not in 示例	select * from star where id not in (1,3,5,7);
like示例 (模糊查询 % 表示通配符)	select * from star where name like "王%"； 匹配 姓王的数据 select * from star where name like "%丽%"; 匹配名字中含有丽 的数据

9-5:结果集排序

类型	示例
基本语法	select 字段名 from 表名 order by 字段 排序关键词;

关键词	说明
asc	升序排序（默认值）
desc	降序排序

示例	
select * from star order by money asc;	查询出star表中的所有的数据，按照money字段升序排序
select * from star order by money desc;	查询出star表中的所有的数据，按照money字段降序排序

9-6:多字段排序

类型	说明
基本语法	select 字段名 from 表名 order by 字段1 desc|asc，.....字段2 desc|asc;
示例	select * from star order by money desc,age asc;
示例说明	查询star表中的所有的数据，按照money字段降序排序，若money字段的值一样，则按照age字段升序排序。

9-7:限制查询的结果集

类型	说明
基本语法	select 字段 from 表名 limit 数量;
示例	select * from star limit 3;
示例说明	查询star表中的前3条记录;

对于查询或者排序后的结果集，如果希望只显示一部分，

使用limit关键字对结果集进行数量限制。

9-8:限制排序后的结果集

类型	说明
基本语法	select 字段名 from 表名 order by 字段 排序规则 limit 数量;
示例	select * from star order by money desc limit 3;
示例说明	查询出star表中money数字最大的前3个数据;

9-9:结果集区间选择

类型	说明
基本语法	select 字段名 from 表名 order by 字段名 排序规则 limit 偏移量,数量;
示例	select * from star order by money desc limit 2,3;
示例说明	查询出star表中按照money降序排序，从第二条数据开始往后取3条数据;

9-10:常见统计函数

类型	说明	示例
sum	求和	select sum(money) as 总财富 from star; 统计star表中money字段的总和
count	统计数量	select count(id) as 总数量 from star; 统计star表中总共有多少条数据
max	最大值	select max(money) as money最大值 from star; 查询star表中的money字段的最大值
min	最小值	select min(money) as money最小值 from star; 查询star表中的money字段的最小值
avg	平均值	select avg(money) as money平均数 from star; 查询star表中的money字段的平均值

9-11:分组

类型	说明
基本语法	select * from 表名 group by 字段名;
示例	select province from star group by province;
示例说明	将star表中的数据按照province进行分组;

9-12:分组统计

类型	说明
基本语法	select count(*),字段名 from 表名 group by 字段名;
示例	select count(*) as 数量,province from star group by province;
示例说明	统计star表中每个省份的明星的数量;

9-13:结果集过滤

类型	说明
基本语法	select count(*) as result,字段 from 表名 group by 字段 having 条件;
示例	select count(*) as result,province from star group by province having result >= 2;
示例说明	统计star表中的明星的数量大于等于2 的省份;

10.修改数据:

类别	详细解释
基本语法	update 表名 set 字段1 = 值1，字段2 = 值2，字段3 = 值3 where 条件;
示例	update star set name = '谦爷',money = 9000,province = '湖南省' where id = 6;
示例说明	将star表中的id为6的数据name = 谦爷，money = 9000,province = '湖南省';

注意： 在mysql的修改语句中，where条件不能省略，条件字段一般使用id字段居多。

11.删除数据:

类别	详细解释
基本语法	delete from 表名 where 条件;
示例	delete from star where id = 6;
示例说明	将star表中的id为6的数据进行删除。

注意：在mysql中的删除语句中，where条件不能省略，条件字段一般使用id字段居多。

12.联合查询

多表联合查询

很多时候在实际的业务中我们不只是查询一张表。如：
1.在电子商务系统中，查询用户购买过哪些商品。
2.银行中可以查询违规记录，同时查询出用户的基本信息
3.查询中奖信息，同时查询中奖人员的基本信息。
而上述业务中需要多表联合在一起查询，其本质就是表连接。

多表联合查询分类

内连接:选出两个表中存在连接关系的字段符合连接关系的那些记录。
外连接:会选出其他不匹配的记录，分为外左连接和外右连接。

准备数据:

```
# 创建用户表:
create table user(
    id int auto_increment,
    username char(100),
    password char(100),
    gid int,
    primary key(id)
);
# 插入用户数据
insert into user(username,password,gid) values('张三',123456,2),('李四',987654,4),
('王五',676767,1),('赵六',898989,3),('宋琪',654433,2),('柳岩',123356,0),('郭德
纲',987654,7),('黄渤',673767,2),('王宝强',835989,0),('宋慧桥',652433,3);
# 创建商品表:
create table goods(
    id int auto_increment,
```

```
    name char(100),
    price int(20),
    primary key(id)
);
insert into goods(name,price) values('奥迪A6',987654),('比亚迪-秦',378783),('奔驰GL8',1243224),('大众辉腾',1346245),('特斯拉',367889),('宝马X5',887542),('丰田霸道',875432);
```

隐式内连接

类型	说明
基本语法	select 表1.字段名 [as 别名], 表n.字段名 from 表1[as 别名], 表n where 条件;
示例	select user.username，goods.name from user,goods where user.gid = goods.id;
示例说明	查询用户表中的哪些用户购买过商品，并将购买的商品信息查询出来。

注意： 以上的连接方式是隐式内连接，因为在上述的查询语句中未出现join关键字。

显示内连接

类型	说明
基本语法	select 表1.字段名[as 别名]，表n.字段名 from 表1 inner join 表n on 条件;
示例	select user.username,goods.name from user inner join goods on user.gid = goods.id;
示例说明	查询用户表中的哪些用户购买过商品，并将购买的商品信息查询出来。

注意：在显示内连接中，一般把inner join 中的inner省略。

外连接之左连接

类型	说明
基本语法	select 表1.字段名 [as 别名]，表n.字段名 from 表1 left join 表n on条件;
示例	select * from user left join goods on user.gid = goods.id;
示例说明	以左表（user）为主，查询出用户购买过哪些商品，并且把商品信息查询出来。

注意：包含左表中所有的数据和右表中满足条件的数据。

外连接之右连接

类型	说明
基本语法	select 表1.字段名 [as 别名]，表n.字段名 from 表1 right join 表n on条件;
示例	select * from user right join goods on user.gid = goods.id;
示例说明	以右表（goods）为主，查询出哪些商品被用户购买过，并且把用户信息查询出来。

注意：包含右表中所有的数据和左表中满足条件的数据。

子(嵌套)查询

类型	说明
基本语法	select 字段名 from 表名 where 字段 in (子查询语句);
示例	select * from user where gid in (select id from goods);
示例说明	将购买过商品的用户信息查询出来。

13.窗口函数

一.窗口函数有什么用?

在日常工作中，经常会遇到需要在每组内排名，比如下面的业务需求：

```
排名问题：每个部门按业绩来排名
topN问题：找出每个部门排名前N的员工进行奖励

面对这类需求，就需要使用sql的高级功能窗口函数了。
```

二.什么是窗口函数?

窗口函数，也叫OLAP函数（Online Anallytical Processing，联机分析处理），可以对数据库数据进行实时分析处理。

窗口函数的基本语法如下：

```
<窗口函数> over (partition by <用于分组的列名>
                 order by <用于排序的列名>)
```

<窗口函数>的位置，可以放以下两种函数：

1） 专用窗口函数，包括后面要讲到的rank, dense_rank, row_number等专用窗口函数。

2） 聚合函数，如sum,avg, count, max, min等

因为窗口函数是对where或者group by子句处理后的结果进行操作，所以窗口函数原则上只能写在select子句中。

准备数据：

```
学生表:
CREATE TABLE `student`  (
  `id` int(0) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
  `age` int(0) NULL DEFAULT NULL,
  `sex` enum('男','女') CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
  `score` int(0) NULL DEFAULT NULL,
  `class` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
  PRIMARY KEY (`id`) USING BTREE,
  INDEX `sex`(`sex`) USING BTREE
```

```
) ENGINE = InnoDB AUTO_INCREMENT = 10 CHARACTER SET = utf8 COLLATE =
utf8_general_ci ROW_FORMAT = Dynamic;

INSERT INTO `student` VALUES (1, '天偿好', 28, '男', 45, '1班');
INSERT INTO `student` VALUES (2, '奇奇坏', 20, '女', 85, '2班');
INSERT INTO `student` VALUES (3, '小菲美', 18, '女', 70, '1班');
INSERT INTO `student` VALUES (4, '渣渣辉', NULL, '男', 82, '3班');
INSERT INTO `student` VALUES (5, '涛涛壮', 21, '男', 70, '1班');
INSERT INTO `student` VALUES (6, 'aaaaa', 23, '男', 85, '2班');
INSERT INTO `student` VALUES (7, 'bbb', 34, '女', 74, '3班');
INSERT INTO `student` VALUES (8, 'aacc', 32, '男', 79, '1班');
INSERT INTO `student` VALUES (9, 'cc', 31, '男', 70, '2班');
```

订单表：

```
CREATE TABLE `order1`  (
  `id` int(0) UNSIGNED NOT NULL AUTO_INCREMENT,
  `user_num` int(0) UNSIGNED NOT NULL,
  `amount` float UNSIGNED NULL DEFAULT NULL,
  `dt` date NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 10 CHARACTER SET = utf8 COLLATE =
utf8_general_ci ROW_FORMAT = Dynamic;


INSERT INTO `order1` VALUES (1, 1, 100, '2021-05-31');
INSERT INTO `order1` VALUES (2, 1, 210, '2021-05-13');
INSERT INTO `order1` VALUES (3, 2, 231, '2021-06-11');
INSERT INTO `order1` VALUES (4, 2, 543, '2021-07-11');
INSERT INTO `order1` VALUES (5, 2, 325, '2021-07-30');
INSERT INTO `order1` VALUES (6, 3, 212, '2021-07-21');
INSERT INTO `order1` VALUES (7, 3, 323, '2021-06-11');
INSERT INTO `order1` VALUES (8, 3, 333, '2021-08-02');
INSERT INTO `order1` VALUES (9, 3, 110, '2021-07-28');
INSERT INTO `order1` VALUES (10, 3, 90, '2021-08-03');
```

交易表：

```
CREATE TABLE `transaction`  (
  `id` int(0) UNSIGNED NOT NULL AUTO_INCREMENT,
  `userid` int(0) NULL DEFAULT NULL,
  `amount` int(0) NULL DEFAULT NULL,
  `paydate` date NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Dynamic;

INSERT INTO `transaction` VALUES (1, 1, 212, '2021-11-02');
INSERT INTO `transaction` VALUES (2, 2, 324, '2021-11-04');
INSERT INTO `transaction` VALUES (3, 1, 456, '2021-11-05');
INSERT INTO `transaction` VALUES (4, 3, 3476, '2021-11-06');
INSERT INTO `transaction` VALUES (5, 4, 675, '2021-11-06');
INSERT INTO `transaction` VALUES (6, 5, 12, '2021-11-07');
INSERT INTO `transaction` VALUES (7, 2, 543, '2021-11-08');
INSERT INTO `transaction` VALUES (8, 1, 34, '2021-11-08');
INSERT INTO `transaction` VALUES (9, 2, 4562, '2021-11-02');
INSERT INTO `transaction` VALUES (10, 2, 31, '2021-11-02');
INSERT INTO `transaction` VALUES (11, 3, 23, '2021-11-02');
INSERT INTO `transaction` VALUES (12, 3, 321, '2021-11-04');
INSERT INTO `transaction` VALUES (13, 4, 12, '2021-11-04');
```

```
INSERT INTO `transaction` VALUES (14, 4, 2345, '2021-11-05');
INSERT INTO `transaction` VALUES (15, 2, 32, '2021-11-05');
INSERT INTO `transaction` VALUES (16, 2, 3334, '2021-11-06');
INSERT INTO `transaction` VALUES (17, 1, 76, '2021-11-06');
INSERT INTO `transaction` VALUES (18, 3, 435, '2021-11-07');
INSERT INTO `transaction` VALUES (19, 4, 7765, '2021-11-07');
INSERT INTO `transaction` VALUES (20, 3, 433, '2021-11-08');
INSERT INTO `transaction` VALUES (21, 4, 325, '2021-11-08');
INSERT INTO `transaction` VALUES (22, 2, 433, '2021-11-08');
```

用户表：

```
CREATE TABLE `user`  (
  `id` int(0) UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
  `address` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
  `createtime` datetime(0) NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci ROW_FORMAT =
Dynamic;

INSERT INTO `user` VALUES (1, '张三', '深圳', '2021-08-01 12:41:12');
INSERT INTO `user` VALUES (2, '李四', '广州', '2021-06-01 12:41:26');
INSERT INTO `user` VALUES (3, '王文英', '北京', '2021-10-12 12:41:52');
INSERT INTO `user` VALUES (4, '王五', '纽约', '2021-06-17 12:42:04');
INSERT INTO `user` VALUES (5, '赵敏', '台湾', '2021-11-16 12:42:27');
```

三、主要分类：

用法：窗口函数 over(partition by 分组字段 order by 排序字段)

（1）序号函数：

rank(): 返回数据集中每个值的排名。排名值是根据当前行之前的行数加1，不包含当前行。因此，排序的关联值可能产生顺序上的空隙。 这个排名会对每个窗口分区进行计算；

dense_rank() : 返回一组数值中每个数值的排名。这个函数与 rank() ，相似，除了关联值不会产生顺序上的空隙；

row_number() : 为每行数据返回一个唯一的顺序的行号，从1开始，根据行在窗口分区内的顺序；

需求:按照班级将学生分组,然后将每个班级的学生按照成绩排名.

现在我们说回来，为什么叫“窗口”函数呢？这是因为partition by分组后的结果称为“窗口”，这里的窗口不是我们家里的门窗，而是表示“范围”的意思。

简单来说，窗口函数有以下功能：

同时具有分组和排序的功能.

不减少原表的行数.

专用窗口函数rank, dense_rank, row_number有什么区别呢？

它们的区别我举个例子，你们一下就能看懂：

```
select *,
RANK() over(partition by class order by score desc) as ranking,
DENSE_RANK() over(partition by class order by score desc) as dense,
ROW_NUMBER() over(partition by class order by score desc) as rownumber
from student;
```

从上面的结果可以看出：

rank函数：这个例子中如果有并列名次的行，会占用下一名次的位置。比如正常排名是1，2，3，4，但是现在前3名是并列的名次，结果是：1，1，1，4。

dense_rank函数：这个例子中如果有并列名次的行，不占用下一名次的位置。比如正常排名是1，2，3，4，但是现在前3名是并列的名次，结果是：1，1，1，2。

row_number函数：这个例子中不考虑并列名次的情况。比如前3名是并列的名次，排名是正常的1，2，3，4。

最后，需要强调的一点是：在上述的这三个专用窗口函数中，函数后面的括号不需要任何参数，保持()空着就可以。

（2）分布函数：

cume_dist()：表示当前行及小于当前行在窗口分区总行数中的占比。

需求:按照订单数统计在总订单数中的占比.

```
select *,cume_dist() over(partition by user_num order by amount desc) as cume
from order1;
```

这个数值是怎么计算到的呢？答案是:(rank-1) / (rows-1)

rank就是我们前面使用rank()函数计算出来的排名，rows就是行数

（3）前后函数：

lag() 向上偏移 lead() 向下偏移

应用场景:

```
1.取时间间隔为N天的记录
2.计算本次记录与上次记录的差值
3.取某一字段的前N行数据或后N行数据
```

三个参数: 第一个参数是表达式或者字段，第二个参数是偏移量 第三个参数 就是控制赋值.

需求:查询上一个订单距离当前订单的时间间隔。

```
select *,datediff(dt,last_date) as diff from(
    select user_num,dt,lag(dt,1,dt) over w as last_date from order1
    window w as (order by dt)  # 给窗口函数起别名
)t; # t表示给sql语句起别名
```

(4) 头尾函数：first_value()/ last_value()

按时间排序记录每个用户的最早订单和最晚订单，并不是最小金额和最大金额订单。

需求:查询截止到当前订单，按照日期排序第一个订单和最后一个订单的订单金额。

```
select * from (select *,
     first_value(amount) over w as first_amount, # first_value表示第一个订单， w表示后面的窗口函数
     last_value(amount) over w as last_amount #last_value表示最后一个订单
     from order1
     WINDOW w as (partition by user_num order by dt)
)t;
```