MySQL

数值类型

整型

类型	大小	范围（有符号）	范围（无符号）	用途
TINYINT	1 byte	(-128，127)	(0，255)	小整数值
SMALLINT	2 bytes	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 bytes	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT或INTEGER	4 bytes	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 bytes	(-9,223,372,036,854,775,808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 bytes	(-3.402 823 466 E+38，3.402 823 466 351 E+38)	0，(1.175 494 351 E-38，3.402 823 466 E+38)	单精度 浮点数值
DOUBLE	8 bytes	(-1.797 693 134 862 315 7 E+308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度 浮点数值
DECIMAL		依赖于M和D的值	依赖于M和D的值	小数值

字符串

类型	大小	用途
CHAR	0-255 bytes	定长字符串
VARCHAR	0-65535 bytes	变长字符串
TINYBLOB	0-255 bytes	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 bytes	短文本字符串
BLOB	0-65 535 bytes	二进制形式的长文本数据
TEXT	0-65 535 bytes	长文本数据
MEDIUMBLOB	0-16 777 215 bytes	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 bytes	中等长度文本数据
LONGBLOB	0-4 294 967 295 bytes	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 bytes	极大文本数据

日期

类型	大小 (bytes)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	4	1970-01-01 00:00:00/2038 结束时间是第 2147483647 秒，北京时间 2038-1-19 11:14:07，格林尼治时间 2038年1月19日 凌晨 03:14:07	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

对表结构的常用操作-其他操作

功能	SQL
查看当前数据库的所有表名称	show tables;
查看指定某个表的创建语句	show create table 表名;
查看表结构	desc 表名
删除表	drop table 表名

```sql
-- 查看当前数据所有的表
show tables;

-- 查看指定表的创建语句
show create table student;

CREATE TABLE `student` (
  `sid` int NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  `age` int NOT NULL,
  PRIMARY KEY (`sid`),
  UNIQUE KEY `name` (`name`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb3

-- 查看表结构
desc student;

+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  | UNI | NULL    |                |
| age   | int         | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+

-- 删除表
drop table student;
```

对表结构的常用操作-修改表结构格式

```
-- 添加列
alter table student add dept varchar(20);
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  | UNI | NULL    |                |
| age   | int         | NO   |     | NULL    |                |
| dept  | varchar(20) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+


-- 修改列名和类型
alter table student change dept department varchar(30);
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| sid        | int         | NO   | PRI | NULL    | auto_increment |
| name       | varchar(20) | YES  | UNI | NULL    |                |
| age        | int         | NO   |     | NULL    |                |
| department | varchar(30) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+


-- 修改表删除列
alter table student drop department;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| sid   | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  | UNI | NULL    |                |
| age   | int         | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+


-- 修改表名
rename table student to stu;
+---------------------+
| Tables_in_database1 |
+---------------------+
| stu                 |
+---------------------+
```

数据库操作DML

```
-- 数据插入
insert into student (sid,name,age)values(5,'老七',19),(6,'老八',20);

insert into student(sid,age) values(100,20);


insert into student values(7,'老九',21),(8,'老十',22);
```

```
sid name age
1   张三  20
2   李四  20
3   王五  24
4   老六  18
5   老七  19
6   老八  20
7   老九  21
8   老十  22
100     20
-----------------------------------------------------------------

-- 数据修改
-- 将所有学生的年纪修改为20
update student set address ='武汉';

-- 将4的地址改为葛店
update student set address ='葛店' where sid =4;

update student set address ='长职' where sid >4;

-- 将id为3的学生地址修改为光谷 年龄修改为18
update student set address = '光谷',age=18 where sid =3;

-----------------------------------------------------------------
-- 数据删除
-- 删除sid为4的学生数据
delete from student where sid =4;

-- 删除表所有数据
delete from student;

-- 清空表数据
truncate table student;
truncate student;

-----------------------------------------------------------------
-- 案例
-- 创建员工表employee
-- id name gender salary
create table if not exists database1.employee(
id int,
name varchar(20),
gender varchar(10),
salary double
);

-- 插入数据
-- 1 张三 男 2000
-- 2 李四 男 1000
-- 3 王五 女 4000
insert into employee values(1,'张三','男',2000),(2,'李四','男',1000),(3,'王五','女',4000);

-- 修改表数据
```

```sql
-- 将所有员工薪水修改为5000元
update employee  set salary=5000;

-- 将姓名为张三的员工薪水修改为3000元
update employee set salary=3000 where name = '张三';

-- 将姓名为李四的员工薪水修改为4000元 gender改为女
update employee set salary=4000,gender='女' where name = '李四';

-- 将王五的薪水在原有的基础上增加1000元
update employee set salary=salary+1000 where name = '王五';
```

Mysql约束

主键约束

方式1-语法：

```sql
-- 在 create table 语句中，通过 PRIMARY KEY 关键字来指定主键。
--在定义字段的同时指定主键，语法格式如下：
create table 表名(
    ...
    <字段名> <数据类型> primary key
    ...
)
```

方式1-实现：

```sql
create table emp1(
    eid int primay key,
    name VARCHAR(20),
    deptId int,
    salary double
);
```

方式2-语法：

```sql
--在定义字段之后再指定主键，语法格式如下：
create table 表名(
    ...
    [constraint <约束名>] primary key [字段名]
);
```

方式2-实现：

```sql
create table emp2(
    eid INT,
    name VARCHAR(20),
    deptId INT,
    salary double,
    constraint  pk1 primary key(id)
);
```

```sql
-- 联合主键
-- 所谓的联合主键 就是这个主键是由一张表中多个字段组成的
-- primary key (字段1，字段2....，字段n)
create table emp3(
name varchar(20),
deptId int,
salary double,
constraint pk2 primary key(name,deptId)
);
```

```
insert into emp3 values('张三',10,5000);
insert into emp3 values('张三',20,5000);

-- 联合主键的各列 每一列都不能为空
insert into emp3 values(NULL,30,5000);
insert into emp3 values('赵六',NULL,5000);
insert into emp3 values(NULL,NULL,5000);
-----------------------------------------------------------------

-- 添加单列主键
create table emp4(
eid int,
name varchar(20),
deptId int,
salary double
);

alter table emp4 add primary key(eid);


-- 创建多列主键
create table emp5(
eid int,
name varchar(20),
deptId int,
salary double
);

alter table emp5 add primary key(name,deptId);
-----------------------------------------------------------------
-- 删除主键约束
alter table emp5 drop primary key;
```

自增长约束

```
-- 自增长约束
use database1;
create table t_user1(
id int primary key auto_increment,
name varchar(20)
);

INSERT into t_user1 values(null,'张三');
INSERT into t_user1(name) values('李四');

-- 创建表时指定
create table t_user2(
id int primary key auto_increment,
name varchar(20)
)auto_increment=100;

insert into t_user2 values(null,'张三');
insert into t_user2 values(null,'李四');
-----------------------------------------------------------------
```

```
20
21 -- 创建表之后指定
22 create table t_user3(
23 id int primary key auto_increment,
24 name varchar(20)
25 );
26
27 insert into t_user3 values(null,'张三');
28 insert into t_user3 values(null,'李四');
29 insert into t_user3 values(null,'王五');
30
31 alter table t_user3 auto_increment=100;
32 id  name
33 1   张三
34 2   李四
35 100 王五
36 ----------------------------------------------------------------
37 delete和truncate在删除后自增列的变化
38
39 delete
40 use database1;
41 create table t_user1(
42 id int primary key auto_increment,
43 name varchar(20)
44 );
45
46 INSERT into t_user1 values(null,'张三');
47 delete from t_user1; -- delete删除数据之后 自增长还是在最后一个值基础上+1
48 id  name
49 3   张三
50
51
52 truncate
53 create table t_user2(
54 id int primary key auto_increment,
55 name varchar(20)
56 )auto_increment=100;
57
58 insert into t_user2 values(null,'张三');
59 insert into t_user2 values(null,'李四');
60
61 truncate t_user2;  -- truncate删除数据之后 自增长从1开始+1
62 id  name
63 1   张三
```

非空约束(not null)

```
1 -- 非空约束
2 -- 创建表时指定
3 create table t_user6(
4 id int,
5 name varchar(20) not null,
6 address varchar(20) not null
7 );
8 +---------+-------------+------+-----+---------+-------+
```

```
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | NO   |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+

-- 创建表之后指定
alter table t_user3 modify name varchar(20) not null;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | NO   |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+

-- 删除非空约束
alter table t_user6 modify address varchar(20);
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

唯一约束

```
-- 唯一约束
-- 创建表时添加
create table t_user8(
id int,
name varchar(20),
phone_number varchar(20) unique
);
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| id           | int         | YES  |     | NULL    |       |
| name         | varchar(20) | YES  |     | NULL    |       |
| phone_number | varchar(20) | YES  | UNI | NULL    |       |
+--------------+-------------+------+-----+---------+-------+

insert into t_user8 values(1001,'张三',null);
insert into t_user8 values(1002,'张三1',null);
id      name    phone
1001    张三     (NULL)
1002    张三1    (NULL)
在mysql中NULL和任何值都不相同 NULL不等于NULL

-- 创建表之后指定
alter table t_user6 add constraint UNI unique(address);
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
```

```
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | YES  | UNI | NULL    |       |
+---------+-------------+------+-----+---------+-------+

-- 删除唯一约束
alter table t_user6 drop index UNI;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | NO   |     | NULL    |       |
| address | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

默认约束

```
-- 默认约束
-- 创建表时添加
create table t_user10(
id int,
name varchar(20),
address varchar(20) default '鄂州'
);
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| address | varchar(20) | YES  |     | 鄂州    |       |
+---------+-------------+------+-----+---------+-------+

-- 创建表之后指定
alter table t_user10 modify address varchar(20) default '长职';
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| address | varchar(20) | YES  |     | 长职    |       |
+---------+-------------+------+-----+---------+-------+

-- 删除默认约束
alter table t_user10 modify column address varchar(20) default null;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| id      | int         | YES  |     | NULL    |       |
| name    | varchar(20) | YES  |     | NULL    |       |
| address | varchar(20) | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

零填充约束

```
-- 零填充约束
create table t_user12(
id int zerofill,  -- zerofill默认为int(10) 当插入字段的值小于定义的长度时 会在该值的前面补上相应的0
name varchar(20)
);
+-------+--------------------------+------+-----+---------+-------+
| Field | Type                     | Null | Key | Default | Extra |
+-------+--------------------------+------+-----+---------+-------+
| id    | int(10) unsigned zerofill | YES  |     | NULL    |       |
| name  | varchar(20)              | YES  |     | NULL    |       |
+-------+--------------------------+------+-----+---------+-------+


```

MySQL索引

```
-- 创建表的时候直接指定索引
create table student(
sid int primary key,
car_id varchar(20),
name varchar(20),
gender varchar(20),
age int,
birth date,
phone_num varchar(20),
score double,
index index_name(name)
);

select * from student where name = '张三';

-- 直接创建索引
create index index_gender on student(gender);

-- 修改表结构添加索引
alter table student add index index_age(age);
```

MySQL索引-查看索引

```
-- 查看数据库所有索引
select * from mysql.`innodb_index_stats` a where a.`database_name` = 'database1';

-- 查看表中所有索引
select * from mysql.`innodb_index_stats` a where a.`database_name` = 'database1' and a.table_name like '%student%';

-- 查看表中所有索引
show index from student;
```

MySQL索引-删除索引

```
-- 删除索引
drop index index_age on student;

alter table student drop index index_gender;
```

MySQL索引-唯一索引

```
-- 唯一索引
-- 创建表的时候直接指定
create table student2(
sid int primary key,
card_id varchar(20),
name varchar(20),
gender varchar(20),
age int,
birth date,
phone_num varchar(20),
score double,
unique index_card_id(card_id)
);

-- 直接创建
create unique index index_name on student2(name);

-- 修改表结构(添加索引)
alter table student2 add unique index_phone_num(phone_num);

-- 删除索引
drop index index_card_id on student2;

alter table student2 drop index index_phone_num;
```

MySQL索引-主键索引

```
-- 主键索引
show index from student2;
```

MySQL索引-组合索引

```
-- 组合索引
-- 普通索引
create index index_phone_name on student(phone_num,name);
drop index index_phone_name on student;

-- 唯一索引
create unique index index_phone_name on student(phone_num,name);
```

MySQL索引-全文索引

◆ 索引的操作-全文索引

➢ 概述

MySQL 中的全文索引，有两个变量，最小搜索长度和最大搜索长度，对于长度小于最小搜索长度和大于最大搜索长度的词语，都不会被索引。通俗点就是说，想对一个词语使用全文索引搜索，那么这个词语的长度必须在以上两个变量的区间内。这两个的默认值可以使用以下命令查看:

```
show variables like '%ft%';
```

Variable_name	Value
ft_boolean_syntax	+ -><()~*:""&|
ft_max_word_len	84
ft_min_word_len	4
ft_query_expansion_limit	20
ft_stopword_file	(built-in)
innodb_ft_aux_table	
innodb_ft_cache_size	8000000
innodb_ft_enable_diag_print	OFF
innodb_ft_enable_stopword	ON
innodb_ft_max_token_size	84
innodb_ft_min_token_size	3
innodb_ft_num_word_optimize	2000
innodb_ft_result_cache_limit	2000000000
innodb_ft_server_stopword_table	
innodb_ft_sort_pll_degree	2
innodb_ft_total_cache_size	640000000

```sql
-- 全文索引
show variables like '%ft%';

create table t_article (
     id int primary key auto_increment ,
     title varchar(255) ,
     content varchar(1000) ,
     writing_date date -- ,
     -- fulltext (content) -- 创建全文检索
);
-- 数据准备
insert into t_article values(null,"Yesterday Once More","When I was young I listen to the radio",'2021-10-01');
insert into t_article values(null,"Right Here Waiting","Oceans apart, day after day,and I slowly go insane",'2021-10-02');
insert into t_article values(null,"My Heart Will Go On","every night in my dreams,i see you, i feel you",'2021-10-03');
insert into t_article values(null,"Everything I Do","eLook into my eyes,You will see what you mean to me",'2021-10-04');
insert into t_article values(null,"Called To Say I Love You","say love you no new year's day, to celebrate",'2021-10-05');
insert into t_article values(null,"Nothing's Gonna Change My Love For You","if i had to live my life without you near me",'2021-10-06');
insert into t_article values(null,"Everybody","We're gonna bring the flavor show U how.",'2021-10-07');

-- 修改表结构添加全文索引
alter table t_article add fulltext index_content(content);

-- 添加全文索引
create fulltext index index_content on t_article(content);

-- 使用全文索引
select * from  t_article where match(content) against('yo');

select * from t_article where match(content) against('you');
```

MySQL索引-空间索引

◆ 索引的操作-空间索引

➢ 操作

类型	含义	说明
Geometry	空间数据	任何一种空间类型
Point	点	坐标值
LineString	线	有一系列点连接而成
Polygon	多边形	由多条线组成

```
-- 空间索引
create table shop_info(
id int(10) primary key auto_increment comment 'id',
shop_name varchar(64) not null comment '门店名称',
geom_point geometry not null comment '经纬度',
spatial key geom_index(geom_point)
);
```

MySQL的存储引擎

◆ 分类

- **MyISAM**：Mysql 5.5之前的默认数据库引擎，最为常用。拥有较高的插入，查询速度，但不支持事务
- **InnoDB**：事务型速记的首选引擎，支持ACID事务，支持行级锁定，MySQL5.5成为默认数据库引擎
- Memory：所有数据置于内存的存储引擎，拥有极高的插入，更新和查询效率。但是会占用和数据量成正比的内存空间。并且其内容会在MYSQL重新启动是会丢失。
- Archive：非常适合存储大量的独立的，作为历史记录的数据。因为它们不经常被读取。Archive 拥有高效的插入速度，但其对查询的支持相对较差
- Federated：将不同的 MySQL 服务器联合起来，逻辑上组成一个完整的数据库。非常适合分布式应用

功能	MyISAM	MEMORY	InnoDB
存储限制	256TB	RAM	64TB
支持事务	No	No	Yes
支持全文索引	Yes	No	No
支持B树索引	Yes	Yes	Yes
支持哈希索引	No	Yes	No
支持集群索引	No	No	Yes
支持数据索引	No	Yes	Yes
支持数据压缩	Yes	No	No
空间使用率	低	N/A	高
支持外键	No	No	Yes

-- 修改MySQL默认存储引擎方法
1. 关闭mysql服务
2. 找到mysql安装目录下的my.ini文件：
3. 找到default-storage-engine=INNODB 改为目标引擎，
   如：default-storage-engine=MYISAM
4. 启动mysql服务

```
-- 查询当前数据库支持的存储引擎
show engines;

-- 查看当前的默认存储引擎
show variables like '%default_storage_engine%';

-- 查看某个表用了什么引擎
show create table student;

-- 创建新表指定存储引擎
create table stud1(
id int primary key,
name varchar(10)
) engine = MyISAM;

show create table stud1;

-- 修改引擎
alter table student engine = innodb;
alter table student engine = MyISAM;
```

DQL-基本查询

```
create  table product(
pid int primary key auto_increment,
pname varchar(20) not null,
price double,
categroy_id varchar(20)
);

insert into product values(null,'海尔洗衣机',5000,'c001');
insert into product values(null,'美的冰箱',3000,'c001');
insert into product values(null,'格力电饭煲',5000,'c001');
insert into product values(null,'九阳电饭煲',5000,'c001');

-- 查询所有的商品
select * from product;

-- 查询商品名和商品的价格
select pname,price from product;

-- 别名查询.使用的关键字是as
-- 表别名
select * from product as p;
select *from product p;

select p.id,u.id from product p, user u;

-- 列别名
```

```sql
select pname as '商品名',price '商品价格' from product;

-- 去掉重复值
select distinct price from product;

-- 查询结果是表达式(运算查询):将所有商品的价格加价10元进行显示
select pname, price +10 new_price from product;

-- 将所有商品的价格加价10%进行显示
select pname, price * 1.1 as new_price from product;

-- 查询商品名称为海尔洗衣机的商品所有信息
select * from product where pname = '海尔洗衣机';

-- 查询价格为5000商品
select * from product where price = 5000;

-- 查询价格不为5000的所有商品
select * from product where price != 5000;
select * from product where price <> 5000;
select * from product where not(price = 5000);

-- 查询价格大于3000元的所有商品信息
select * from product where price > 3000;

-- 查询价格在3000到5000之间的商品信息
select * from product where price >=3000 && price<=5000;
select * from product where price between 3000 and 5000;

-- 查询价格是3000或5000的所有商品
select * from product where price = 3000 || price =5000;

-- 查询含有'格力'的所有商品
select * from product where pname like '%格力%';        -- %为任意字符

-- 查询以'海'开头的所有商品
select * from product where pname like '海%';

-- 查询第二个字为'力'的所有商品
select * from product where pname like '_力%';          -- 下划线匹配单个字符

-- 查询category_id为Null的商品
select * from product where categroy_id is null;

-- 查询categroy_id不为null分类的商品
select * from product where categroy_id is not null;

-- 使用least求最小值
select least(10,20,30) as small_number;
select least(10,null,20);       -- 如果求最小值中 有个值为Null 则不会进行比较 结果直接为null

-- 使用greatest求最大值
select greatest(50,20,40) as big_number;
select greatest(10,null,20);       -- 如果求最大值中 有个值为Null 则不会进行比较结果直接为null
```

排序查询

```sql
-- 排序查询
-- 使用价格查询 降序
select * from product order by price desc;

-- 在价格降序的基础上 以分类降序
select * from product order by price desc,categroy_id desc;

-- 显示商品的价格去重 并降序
select distinct price from product order by price desc;
```

聚合查询

```sql
-- 聚合查询
-- 查询商品的总条目
select count(pid) from product;
select count(*) from product;

-- 查询价格大于200商品的总条目
select count(pid) from product where price > 4500;

-- 查询分类为'c001'的所有商品的总和
select sum(price) from product where categroy_id ='c001';

-- 查询商品的最大价格
select max(price) from product;

-- 查询商品的最小价格
select min(price) from product;

-- 查询分类为'c001'所有商品的平均价格
select avg(price) from product where categroy_id='c001';


-- 聚合查询Null值的处理
create table test_null(
c1 varchar(20),
c2 int
);

insert into test_null values('aaa',3);
insert into test_null values('bbb',3);
insert into test_null values('ccc',null);
insert into test_null values('ddd',6);

select count(*), count(1), count(c2) from test_null;  -- 4 4 3
select sum(c2), max(c2), min(c2), avg(c2) from test_null;  -- 12 6 3 4
```

分组查询

```
-- 分组查询
-- select 字段1,字段2 from group by 分组字段 having 分组条件
-- 统计各个分类商品的个数
-- 分组之后 select的后边只能写分组字段和聚合函数
select categroy_id,count(pid) from product group by categroy_id;

-- where子句用来筛选from子句中指定的操作所产生的行
-- group by子句用来分组where子句的输出
-- having子句用来从分组的结果中筛选行
-- select 字段1,字段2 from 表名 group by 分组字段 having 分组条件;

-- 统计各个分类商品的个数 且只显示个数大于4的信息
select categroy_id,count(*) from product group by categroy_id having count(*) > 1 order by categroy_id;
```

分页查询

```
-- 分页查询
-- 显示前3条
select * from product limit 3;

-- 从第2条开始显示 显示3条
select * from product limit 2,3;

-- 分页显示
select * from product limit 0,60;
select * from product limit 60,60;
select * from product limit 120,60;
select * from product limit (n-1)*60,60;
```

insert into select语句

```
-- insert into select
select * from product;

create table product2(
pname varchar(20),
price double
);

insert into product2(pname,price) select pname,price from product;
select * from product2;

create table product3(
categroy_id varchar(20),
product_count int
);

insert into product3 select categroy_id , count(*) from product group by categroy_id;
select * from product3;
```

练习

```sql
-- 练习
create table student(
id int,
name varchar(20),
gender varchar(20),
chinese int,
english int,
math int
);

insert into student values(1,'张明','男',89,78,90);
insert into student values(2,'李进','男',67,53,95);
insert into student values(3,'王五','女',87,78,77);
insert into student values(4,'李一','女',88,98,92);
insert into student values(5,'李财','男',82,84,67);
insert into student values(6,'张宝','男',55,85,45);
insert into student values(7,'黄蓉','女',75,65,30);
insert into student values(7,'黄蓉','女',75,65,30);

-- 查询学生表中所有的信息
select * from student;

-- 查询表中所有学生的姓名和对应的英语成绩
select name,english from student;

-- 过滤表中重复数据
select distinct * from student;

-- 统计每个学生的总分
select name,chinese + english + math sum from student;

-- 在所有学生总分数上加10分特长分
select name,chinese + english + math +10 from student;

-- 使用别名表示学生的分数
select name,chinese '语文成绩', english '英语成绩', math '数学成绩' from
student;

-- 查询英语成绩大于90的同学
select name from student where english > 90;

-- 查询总分大于200分的同学
select name from student where (chinese + english + math) > 200;

-- 查询英语分数在80-90之间的同学
select name from student where english >= 80 && english <= 90;

-- 查询英语分数不在80-90之间的同学
select name from student where english <80 || english >90;

-- 查询数学分数为89 90 91的同学
select * from student where math in(89,90,91);
select name from student where math=89 || math=90 || math=91;
```

```sql

-- 查询数学分数不为89 90 91的同学
select * from student where math not in(89,90,91);
select * from student where not math in(89,90,91);

-- 查询所有姓李的学生的英语成绩
select name,english from student where name like '李%';

-- 查询数学分80并且语文分80的同学
select * from student where math=80 && chinese=80;

-- 查询英语80或者总分200的同学
select * from student where english=80 || (chinese + english + math)=200;

-- 对数学成绩降序
select * from student order by math desc;

-- 对总分降序排序
select * from student order by (chinese + english + math) desc;

-- 对姓李的学生成绩排序输出
select * from student where name like '李%' order by (chinese + english +
math) desc;

-- 查询男生和女生分别又多少人 并将人数降序输出
select gender,count(*) as total_cnt from student group by gender order by
total_cnt desc;

-- 查询男生和女生分别又多少人 并将人数降序输出 查询出人数大于4的性别人数信息
select gender,count(*) as total_cnt from student group by gender having
total_cnt >4 order by total_cnt desc;

----------------------------------------------------------------
create table emp(
empno int,
ename varchar(50),
job varchar(50),
mgr int,
hiredate date,
sal int,
comn int,
deptno int
);

insert into emp values(7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL,20),

(7499,'ALEN','SALESMAN',7698,'1981-02-20',1600,300,30),

(7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500,30),

(7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL,20),

(7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400,30),

(7698,'BLAKE','MANAGER',7839,'1981-05-01',2850,NULL,30),
```

```
(7782,'CLARK','MANAGER',7839,'1981-06-09',2450,NULL,10),

(7788,'SCOTT','ANALYST',7566,'1987-04-19',3000,NULL,20),
                        (7839,'KING','PRESIDENT',NULL,'1981-11-17',5000,NULL,10),
                        (7844,'TURNER','SALESMAN',7698,'1981-09-08',1500,0,30),
                        (7876,'ADAMS','CLERK',7788,'1987-05-23',1100,NULL,20),
                        (7900,'JAMES','CLERK',7698,'1981-12-03',950,NULL,30),
                        (7902,'FORD','ANALYST',7566,'1981-12-03',3000,NULL,20),
                        (7934,'MILLER','CLERK',7782,'1982-01-23',1300,NULL,10);

-- 按员工编号升序排列不在10号部门工作的员工信息
select * from emp where deptno != 10 order by empno;

-- 查询姓名第二个字母不是A且薪水大于1000元的员工信息 按年薪降序排序
-- ifnull(sal,0) 如果sal的值为null 则当作0 不为Null 则还是原来的值
select * from emp where ename not like '_A%' and sal > 1000 order by (12*sal+ifnull(comn,0)) desc;

-- 求每个部门的平均薪水
select deptno,avg(sal) from emp group by deptno;
select deptno,avg(sal) as avg_sal from emp group by deptno order by avg_sal desc;

-- 求每个部门的最高薪水
select deptno,max(sal) from emp group by deptno;

-- 求每个部门每个岗位的最高薪水
select deptno,job,max(sal) from emp group by deptno,job order by deptno;

-- 求平均薪水大于2000的部门编号
select deptno,avg(sal) avg_sal from emp group by deptno having avg_sal > 2000;

-- 将部门平均薪水大于1500的部门编号列出来 按部门平均薪水降序排列
select deptno,avg(sal) avg_sal from emp group by deptno having avg_sal > 1500 order by avg_sal desc;

-- 选择公司中有奖金的员工姓名 工资
select ename,sal from emp where comn is not null;

-- 查询员工最高工资和最低工资的差距
select max(sal) - min(sal) as cha from emp;
```

运算符

比较运算符	说明
=	等于
< 和 <=	小于和小于等于
> 和 >=	大于和大于等于
<=>	安全的等于，两个操作码均为NULL时，其所得值为1；而当一个操作码为NULL时，其所得值为0
<> 或!=	不等于
IS NULL 或 ISNULL	判断一个值是否为 NULL
IS NOT NULL	判断一个值是否不为 NULL
LEAST	当有两个或多个参数时，返回最小值
GREATEST	当有两个或多个参数时，返回最大值
BETWEEN AND	判断一个值是否落在两个值之间
IN	判断一个值是IN列表中的任意一个值
NOT IN	判断一个值不是IN列表中的任意一个值
LIKE	通配符匹配
REGEXP	正则表达式匹配

正则表达式

模式	描述
^	匹配输入字符串的开始位置。
$	匹配输入字符串的结束位置。
.	匹配除 "\n" 之外的任何单个字符。
[...]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
[^...]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'。
p1|p2|p3	匹配 p1 或 p2 或 p3。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。

模式	描述
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,m}	m 和 n 均为非负整数，其中n <= m。最少匹配 n 次且最多匹配 m 次。

```
-- 正则表达式
-- ^ 在字符串开始处进行匹配
select 'abc' regexp '^a';
select * from student where name regexp '^张';

-- $ 在字符串末尾开始匹配
select 'abc' regexp 'a$';
select 'abc' regexp 'c$';
select * from student where name regexp '宝$';

-- . 匹配任意字符 可以匹配除换行符外的任意字符
select 'abc' regexp '.b';-- 1
select 'abc' regexp '.c';-- 1
```

```
select 'abc' regexp 'a.';-- 1

-- [...] 匹配括号内的任意单个字符  正则表达式的任意字段是否在前边的字符串中出现
select 'abc' regexp '[xyz]';  -- 0
select 'abc' regexp '[xaz]';  -- 1

-- [^...] 注意^符合只有在[]内才是取反的意思 在别的地方都是表示开始处匹配
select 'a' regexp '[^abc]';  -- 0
select 'x' regexp '[^abc]';  -- 1
select 'abc' regexp '[^a]';  -- 1

-- a* 匹配0个或多个a 包括空字符串 可以作为占位符使用  .有没有指定字符都可以匹配到数据
select 'stab' regexp '.ta*b'; -- 1
select 'stb' regexp '.ta*b';  -- 1
select '' regexp 'a*';  -- 1

-- a+ 匹配1个或者多个a 但是不包括空字符
select 'stab' regexp '.ta+b';  -- 1
select 'stb' regexp '.ta+b';  -- 0

-- a? 匹配0个或者1个a
select 'stb' regexp '.ta?b';  -- 1
select 'stab' regexp '.ta?b'; -- 1
select 'staab' regexp '.ta?b'; -- 0

-- a1|a2 匹配a1或者a2
select 'a' regexp 'a|b';  -- 1
select 'b' regexp 'a|b';  -- 1
select 'b' regexp '^(a|b)'; -- 1
select 'a' regexp '^(a|b)'; -- 1
select 'c' regexp '^(a|b)'; -- 0

-- a{m} 匹配m个a
select 'auuuuc' regexp 'au{4}c'; -- 1
select 'auuuuc' regexp 'au{3}c'; -- 0

-- a{m,} 匹配m个或者更多个a
select 'auuuuc' regexp 'au{3,}c'; -- 1
select 'auuuuc' regexp 'au{4,}c'; -- 1
select 'auuuuc' regexp 'au{5,}c'; -- 0

-- a{m,n} 匹配m到n个a 包含m和n
select 'auuuuc' regexp 'au{3,5}c'; -- 1
select 'auuuuc' regexp 'au{4,5}c'; -- 1
select 'auuuuc' regexp 'au{5,10}c'; -- 0

-- (abc) abc作为一个序列匹配 不用括号括起来都是用单个字符去匹配 如果要把多个字符作为一个整体去匹配就需要用到括号 所以括号适合上面的所有情况
select 'xababy' regexp 'x(abab)y';  -- 1
select 'xababy' regexp 'x(ab)*y';   -- 1
select 'xababy' regexp 'x(ab){1,2}y'; -- 1
select 'xababy' regexp 'x(ab){3}y'; -- 0
```

MySQL的多表操作

外键约束-一对多

```sql
-- 创建部门表 主表
create table if not exists dept(
detpno varchar(20) primary key,
name varchar(20)
);

-- 创建员工表 并创建dept_id 外键约束
create table if not exists emp(
eid varchar(20) primary key,
ename varchar(20),
age int,
dept_id varchar(20),
-- 创建表时外键约束
constraint emp_fk foreign key (dept_id) references dept (detpno)
);

-- 创建表之后添加外键约束
create table if not exists dept2(
detpno varchar(20) primary key,
name varchar(20)
);

create table if not exists emp2(
eid varchar(20) primary key,
ename varchar(20),
age int,
dept_id varchar(20)
);

-- 创建外键约束
alter table emp2 add constraint dept_id_fk foreign key(dept_id) references
dept2(detpno);

-- 删除外键约束
alter table emp2 drop foreign key dept_id_fk;
```

外键约束-多对多

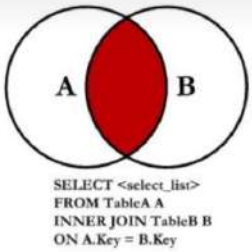
```sql
-- 学生表和课程表（多对多）
-- 创建学生表student(左侧主表)
create table if not exists student(
sid int primary key auto_increment,
name varchar(20),
age int,
gender varchar(20)
);

-- 创建课程表course（右侧主表）
create table course(
cid int primary key auto_increment,
```

```
cidname varchar(20)
);

-- 修改和删除时 中间从表可以随便删除和修改 但是两边的主表受从表依赖的数据不能删除或者修改
-- 创建中间表student_course/score(从表)
create table score(
sid int,
cid int,
score double
);

-- 建立外键约束
alter table score add foreign key(sid) references student(sid);
alter table score add foreign key(cid) references course(cid);

-- 给学生表添加数据
insert into student values(1,'小龙女',18,'女'),(2,'阿紫',19,'女'),(3,'周芷若',20,'男');

-- 给课程表添加数据
insert into course values(1,'语文'),(2,'数学'),(3,'英语');

-- 给中间表添加数据
insert into score values(1,1,78),(1,2,79),(2,1,80),(2,3,81),(3,2,82),(3,3,83);
```

多表联合查询-内连接查询

◆ 内连接查询

内连接查询求多张表的交集



➢ 格式

```
隐式内连接（SQL92标准）：select * from A,B where 条件;
显示内连接（SQL99标准）：select * from A inner join B on 条件;
```

➢ 操作

```
-- 查询每个部门的所属员工
select * from dept3,emp3 where dept3.deptno = emp3.dept_id;
select * from dept3 inner join emp3 on dept3.deptno = emp3.dept_id;
```

```
-- 创建部门表
create table if not exists dept3(
deptno varchar(20) primary key,
name varchar(20)
);

-- 创建员工表
create table if not exists emp3(
eid varchar(20) primary key,
```

```
ename varchar(20),
age int,
dept_id varchar(20)
);

-- 给dept3表添加数据
insert into dept3 values('1001','研发部');
insert into dept3 values('1002','销售部');
insert into dept3 values('1003','财务部');
insert into dept3 values('1004','人事部');

-- 给emp表添加数据
insert into emp3 values('1','乔峰',20,'1001');
insert into emp3 values('2','段誉',21,'1001');
insert into emp3 values('3','虚竹',23,'1001');
insert into emp3 values('4','阿紫',18,'1001');
insert into emp3 values('5','扫地僧',85,'1002');
insert into emp3 values('6','李秋水',33,'1002');
insert into emp3 values('7','鸠摩智',50,'1002');
insert into emp3 values('8','天山童姥',60,'1003');
insert into emp3 values('9','慕容博',58,'1003');
insert into emp3 values('10','丁春秋',71,'1005');

-- 交叉连接查询
select * from dept3,emp3;

-- 内连接查询
-- 隐式内连接 select * from A,B where 条件;
-- 显示内连接 select * from A inner join B on 条件;

-- 查询每个部门的所属员工
-- 隐式内连接
select * from dept3,emp3 where dept3.deptno = emp3.dept_id;
select * from dept3 a,emp3 b where a.deptno = b.dept_id;

-- 显式连接
select * from dept3 inner join emp3 on  dept3.deptno = emp3.dept_id;
select * from dept3 a join emp3 b on a.deptno = b.dept_id;


-- 查询研发部门的所属员工
select * from dept3 a,emp3 b where a.deptno = b.dept_id;
-- 隐式内连接
select * from dept3 a,emp3 b where a.deptno = b.dept_id and name ='研发部';

-- 显示内连接
select * from dept3 a join emp3 b on a.deptno = b.dept_id and name ='研发部';


-- 查询研发部和销售部的所属员工
-- 隐式内连接
select * from dept3 a,emp3 b where a.deptno = b.dept_id and name in ('研发
部','销售部');

-- 显示内连接
```

```sql
select * from dept3 a join emp3 b on a.deptno = b.dept_id and (name = '研发部' or name = '销售部');
select * from dept3 a join emp3 b on a.deptno = b.dept_id and name in ('研发部','销售部');


-- 查询每个部门的员工数 并升序排列
-- 隐式内连接
select a.name,a.deptno,count(1) from dept3 a,emp3 b where a.deptno = b.dept_id group by a.deptno;

-- 显示内连接
select  a.name,a.deptno,count(1) from dept3 a join emp3 b on a.deptno =b.dept_id group by a.deptno;

-- 查询人数大于等于3的部门 并按照人数降序排序
-- 隐式内连接
select a.deptno,a.name,count(1) as total_cnt from dept3 a,emp3 b where a.deptno = b.dept_id group by a.deptno,a.name having total_cnt >=3 order by total_cnt desc;

-- 显示内连接
select a.deptno,a.name,count(1) as total_cnt from dept3 a join emp3 b on a.deptno = b.dept_id group by a.deptno,a.name having total_cnt >=3 order by total_cnt desc;
```

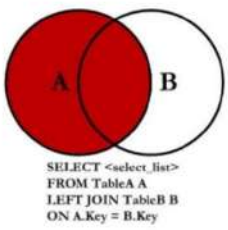
多表联合查询-外连接查询

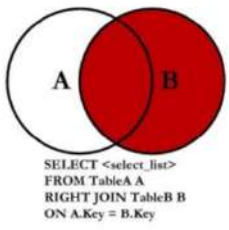
◆ 外连接查询

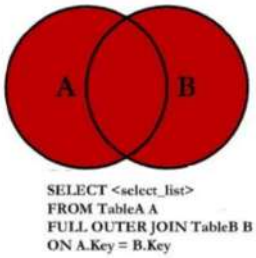
外连接分为左外连接（left outer join）、右外连接(right outer join)，满外连接(full outer join)。
注意：oracle里面有full join,可是在mysql对full join支持的不好。我们可以使用union来达到目的。

➢ 格式

左外连接：left outer join
　　select * from A left outer join B on 条件;
右外连接：right outer join
　　select * from A right outer join B on 条件;
满外连接: full outer join
　　select * from A full outer join B on 条件;








```sql
-- 外连接查询
-- 查询哪些部门有员工 哪些部门没有员工
select * from dept3 left outer join emp3 on dept3.deptno = emp3.dept_id;

select * from A
left join B on 条件1
left join C on 条件2
left join D on 条件3;
```

```sql

-- 查询哪些员工有对应的部门 哪些没有
select * from dept3 right outer join emp3 on dept3.deptno = emp3.dept_id;

select * from A
right join B on 条件1
right join C on 条件2
right join D on 条件3;

-- 使用union关键字实现左外连接和右外连接的并集
select * from dept3 left outer join emp3 on dept.deptno = emp3.dept_id
union
select * from dept3 right outer join emp3 on dept3.deptno = emp3.dept_id;
```

多表联合查询-子查询

◆ 子查询

➢ 介绍

子查询就是指的在一个完整的查询语句之中，嵌套若干个不同功能的小查询，从而一起完成复杂查询的一种编写形式，通俗一点就是包含select嵌套的查询。

➢ 特点

子查询可以返回的数据类型一共分为四种：

1.单行单列：返回的是一个具体列的内容，可以理解为一个单值数据；
2.单行多列：返回一行数据中多个列的内容；
3.多行单列：返回多行记录之中同一列的内容，相当于给出了一个操作范围；
4.多行多列：查询返回的结果是一张临时表

```sql
-- 查询年龄最大的员工信息 显示信息包含员工号 员工名字 员工年龄
-- 1.查询最大年龄：
select max(age) from emp3;

-- 2.让每一个员工的年龄和最大年龄进行比较 相等则满足条件
select * from emp3 where age = (select max(age) from emp3); -- 单行单列 可以作为一个值来用

-- 查询研发部和销售部得到员工信息 包含员工号 员工名字
-- 方式1-关联查询
select * from dept3 a join emp3 b on a.deptno = b.dept_id and (name ='研发部' or name = '销售部');

-- 方式2-子查询
-- 先查询研发部和销售部的部门号 deptno
select deptno from dept3 where name = '研发部' or name ='销售部';

-- 查询哪个员工的部门号是100 或者 1002
select * from emp3 where dept_id in (select deptno from dept3 where name = '研发部' or name ='销售部'); -- 多行单列多个值

-- 查询研发部20岁以下的员工信息 包括员工号 员工名字 部门名字
-- 方式1 关联查询
select * from dept3 a join emp3 b on a.deptno = b.dept_id and (name = '研发部' and age <20);

-- 方式2 子查询
```

```sql
-- 在部门表中查询研发部信息
select * from dept3 where name ='研发部';  -- 一行多列

-- 在员工表中查询年龄小于20岁的员工信息
select * from emp3 where age < 30;

-- 将以上两个查询的结果进行关联查询
select * from (select * from dept3 where name = '研发部') t1 join (select *
from emp3 where age <30) t2 on t1.deptno = t2.dept_id;  -- 多行多列
```

多表联合查询-子查询关键字

子查询关键字-ALL

◆ 子查询关键字-ALL

➢ 格式

```sql
select ...from ...where c > all(查询语句)
--等价于:
select ...from ... where c > result1 and c > result2 and c > result3
```

➢ 特点

- ALL: 与子查询返回的所有值比较为true 则返回true
- ALL可以与=、>、>=、<、<=、<>结合是来使用，分别表示等于、大于、大于等于、小于、小于等于、不等于其中的其中的所有数据。
- ALL表示指定列中的值必须要大于子查询集的每一个值，即必须要大于子查询集的最大值；如果是小于号即小于子查询集的最小值。同理可以推出其它的比较运算符的情况。

➢ 操作

```sql
-- 查询年龄大于'1003'部门所有年龄的员工信息
select * from emp3 where age > all(select age from emp3 where dept_id = '1003');
-- 查询不属于任何一个部门的员工信息
select * from emp3 where dept_id != all(select deptno from dept3);
```

```sql
-- 子查询关键字All
-- 查询年龄大于'1003'部门所有年龄的员工信息
select * from emp3 where age >all(select age from emp3 where dept_id =
'1003');

-- 查询不属于任何一个部门的员工信息
select * from emp3 where dept_id !=all(select deptno from dept3);
```

子查询关键字-any和some

◆ 子查询关键字-ANY和SOME

➢ 格式

```
select …from …where c > any(查询语句)
--等价于:
select ...from ... where c > result1 or c > result2 or c > result3
```

➢ 特点

- ANY:与子查询返回的任何值比较为true 则返回true
- ANY可以与=、>、>=、<、<=、<>结合是来使用，分别表示等于、大于、大于等于、小于、小于等于、不等于其中的其中的任何一个数据。
- 表示制定列中的值要大于子查询中的任意一个值，即必须要大于子查询集中的最小值。同理可以推出其它的比较运算符的情况。
- SOME和ANY的作用一样，SOME可以理解为ANY的别名

➢ 操作

```
-- 查询年龄大于'1003'部门任意一个员工年龄的员工信息
select * from emp3 where age > all(select age from emp3 where dept_id = '1003');
```

```
1 -- 子查询关键字-any和some
2 -- 查询年龄大于'1003'部门任意一个员工年龄的员工信息
3 select * from emp3 where age > any(select age from emp3 where dept_id =
  '1003') and dept_id != '1003';
```

子查询关键字-in

◆ 子查询关键字-IN

➢ 格式

```
select …from …where c in(查询语句)
--等价于:
select ...from ... where c = result1 or c = result2 or c = result3
```

➢ 特点

- IN关键字，用于判断某个记录的值，是否在指定的集合中
- 在IN关键字前边加上not可以将条件反过来

➢ 操作

```
-- 查询研发部和销售部的员工信息，包含员工号、员工名字
select eid,ename,t.name from emp3 where dept_id in (select deptno from dept3
where name = '研发部' or name = '销售部') ;
```

```
1 -- 子查询关键字-in
2 -- 查询研发部和销售部的员工信息 包含员工号 员工名字
3 select eid,ename from emp3 where dept_id in (select deptno from dept3 where
  name ='研发部' or name ='销售部');
```

子查询关键字-EXISTS

子查询关键字-EXISTS

➢ 格式

```
select ...from ...where exists(查询语句)
```

➢ 特点

- 该子查询如果“有数据结果”(至少返回一行数据)， 则该EXISTS() 的结果为“true”，外层查询执行
- 该子查询如果“没有数据结果”（没有任何数据返回），则该EXISTS()的结果为“false”，外层查询不执行
- EXISTS后面的子查询不返回任何实际数据，只返回真或假，当返回真时 where条件成立
- 注意，EXISTS关键字，比IN关键字的运算效率高，因此，在实际开发中，特别是大数据量时，推荐使用EXISTS关键字

➢ 操作

```
-- 查询公司是否有大于60岁的员工，有则输出
select * from emp3 a where exists(select * from emp3 b where a.age > 60);

-- 查询有所属部门的员工信息
select * from dept3 a where exists(select * from emp3 b where a.deptno =
b.dept_id);
```

```
1  -- 子查询关键字-EXISTS
2  -- select ... from ...where exists(查询语句)
3  select * from emp3 where exists(select 1); -- 全表输出
4  select * from emp3 where exists(select * from emp3); -- 全表输出
5
6  -- 查询公司是否有大于60岁的员工 有则输出
7  select * from emp3 a where exists ( select * from emp3  where  a.age > 60);
8  select * from emp3 a where eid in ( select eid from emp3  where  a.age > 60);
9
10 -- 查询有所属部门的员工信息
11 select * from emp3 a where exists ( select * from dept3 b where a.dept_id =
   b.deptno);
12 select * from emp3 a where dept_id in ( select deptno from dept3 b where
   a.dept_id = b.deptno);
```

多表联合查询-自关联查询

◆ 自关联查询

➢ 概念

MySQL有时在信息查询时需要进行对表自身进行关联查询，即一张表自己和自己关联，一张表当成多张表来用。注意自关联时表必须给表起别名。

➢ 格式

```
select 字段列表 from 表1 a , 表1 b where 条件;
或者
select 字段列表 from 表1 a [left] join 表1 b on 条件;
```

➢ 操作

```
-- 创建表,并建立自关联约束
create table t_sanguo(
    eid int primary key ,
    ename varchar(20),
    manager_id int,
 foreign key (manager_id) references t_sanguo (eid)  -- 添加自关联约束
);
```

```
1  -- 多表联合查询-自关联查询
```

```sql
-- 创建表 并建立自关联查询
create table t_sanguo(
eid int primary key,   -- 主键列
ename varchar(20),
manager_id int,  -- 外键列
foreign key(manager_id) references t_sanguo(eid)  -- 添加自关联约束
);

-- 添加数据
insert into t_sanguo values(1,'刘协',NUll);
insert into t_sanguo values(2,'刘备',1);
insert into t_sanguo values(3,'关羽',2);
insert into t_sanguo values(4,'张飞',2);
insert into t_sanguo values(5,'曹操',1);
insert into t_sanguo values(6,'许褚',5);
insert into t_sanguo values(7,'典韦',5);
insert into t_sanguo values(8,'孙权',1);
insert into t_sanguo values(9,'周瑜',8);
insert into t_sanguo values(10,'鲁肃',8);

-- 进行关联查询
-- 查询每个三国人物及他的上级信息 如 关羽 刘备
select * from t_sanguo a,t_sanguo b where a.manager_id=b.eid;
select a.ename,b.ename from t_sanguo a,t_sanguo b where a.manager_id=b.eid;
select a.ename,b.ename from t_sanguo a join t_sanguo b on a.manager_id=b.eid;

-- 查询所有人物及上级
select a.ename,b.ename from t_sanguo a left join t_sanguo b on
a.manager_id=b.eid;

-- 查询所有任务 上级 上上级
select
a.ename,b.ename,c.ename
from t_sanguo a
left join t_sanguo b on a.manager_id=b.eid
left join t_sanguo c on b.manager_id=c.eid;
```

练习

```sql
-- 创建部门表
create table dept(
deptno int primary key,
dname varchar(14),
loca varchar(13)
);

insert into dept values(10,'accounting','new york');
insert into dept values(20,'research','dallas');
insert into dept values(30,'sales','chicago');
insert into dept values(40,'operations','boston');

-- 创建员工表
create table emp(
empno int primary key,
ename varchar(10),
```

```
jon varchar(9),
mgr int,
hiredate date,
sal double,
comm double,
deptno int
);

-- 添加 部门和员工 之间的主外键关系
alter table emp add constraint foreign key emp(deptno) references
dept(deptno);

insert into emp values(7369, 'smith' , 'clerk',7902,'1980-12-
17',800,null,20);
insert into emp values(7499, 'allen' , 'salesman' ,7698,'1981-02-
20',1600,300,30);
insert into emp values(7521, 'ward' , 'salesman',7698,'1981-02-
22',1250,500,30);
insert into emp values(7566,'jones' , 'manager',7839,'1981-04-
02',2975,null,20);
insert into emp values(7654, 'martin' , 'salesman',7698,'1981-09-
28',1250,1400,30);
insert into emp values(7698, 'blake' , 'manager' ,7839,'1981-05-01
',2850,null,30);
insert into emp values(7782,'clark' , 'manager',7839,'1981-06-
09',2450,null,10);
insert into emp values(7788, 'scott' , 'analyst',7566,'1987-07-03' ,
3000,null,20);
insert into emp values(7839, 'king' , 'president' , null,'1981-11-
17',5000,null,10);
insert into emp values(7844, 'turner' , 'salesman' ,7698,'1981-09-
08',1500,0,30);
insert into emp values(7876, 'adams' , 'clerk',7788,'1987-07-
13',1100,null,20);
insert into emp values(7900, 'james' , 'clerk',7698,'1981-12-03'
,950,null,30);
insert into emp values(7902, 'ford' , 'analyst' ,7566,'1981-12-
03',3000,null,20);
insert into emp values(7934, 'miller' , 'clerk',7782,'1981-01-
23',1300,null,10);

-- 创建工资等级表
create table salgrade(
grade int,
losal double,
hisal double
);

insert into salgrade values(1,700,1200);
insert into salgrade values(2,1201,1400);
insert into salgrade values(3,1401,2000);
insert into salgrade values(4,2001,3000);
insert into salgrade values(5,3001,9999);

-- 返回拥有员工的部门名 部门号
```

```sql
select distinct d.dname,d.deptno from dept d join emp e on d.deptno =
e.deptno;

-- 工资水平多于smith的员工信息
select * from emp where sal > (select sal from emp where ename = 'smith');

-- 返回员工和所属经理的姓名
select a.ename,b.ename from emp a,emp b where a.mgr = b.empno;

-- 返回雇员的雇佣日期早于其经理雇佣日期的员工及其经理姓名
select a.ename,a.hiredate,b.ename,b.hiredate from emp a,emp b where a.mgr =
b.empno and a.hiredate < b.hiredate;

-- 返回员工姓名及其所在的部门名称
select e.ename,d.dname from emp e,dept d where e.deptno = d.deptno;

-- 返回从事clerk工作的员工姓名和所在部门名称
select e.ename,d.dname,e.jon from emp e,dept d where e.deptno =d.deptno and
e.jon = 'clerk';

-- 返回部门号及其本部门的最低工资
select deptno,min(sal) from emp group by deptno;

-- 返回销售部(sales)所有员工的姓名
select e.ename from emp e,dept d where e.deptno = d.deptno and d.dname =
'sales';

-- 返回工资水平多于平均工资的员工
select * from emp where sal > (select avg(sal) from emp);

-- 返回与scott从事相同工作的员工
select * from emp where jon = (select jon from emp where ename = 'scott')
and ename != 'scott';

-- 返回工资高于30部门所有员工工资水平的员工信息
select * from emp where sal > all(select sal from emp where deptno=30);

-- 返回员工工作及其从事此工作的最低工资
select jon,min(sal) from emp group by jon;

-- 计算出员工的年薪 并且以年薪降序排列
select ename,(sal *12 +ifnull(comm,0)) as year_sal  from emp order by (sal
*12 +ifnull(comm,0)) desc;   -- 非空ifnull

-- 返回工资处于第四级别的员工的姓名
select ename from emp where sal between (select losal from salgrade where
grade = 4) and (select hisal from salgrade where grade =4);

-- 返回工资为二级别的职员名字 部门所在地
select * from dept d join emp e on e.deptno = d.deptno join salgrade s on
grade =2 and e.sal >= s.losal and e.sal <=s.hisal;

select * from dept d,emp e,salgrade s where e.deptno = d.deptno and grade =
2 and e.sal >=s.losal and e.sal <= s.hisal;
```

MySQL的函数

mysql的函数-聚合函数

MySQL的函数-聚合函数

◆ 概述

- 在MySQL中，聚合函数主要由：count,sum,min,max,avg,这些聚合函数我们之前都学过，不再重复。这里我们学习另外一个函数:group_concat()，该函数用户实现行的合并
- group_concat()函数首先根据group by指定的列进行分组，并且用分隔符分隔，将同一个分组中的值连接起来，返回一个字符串结果。

◆ 格式

```
group_concat([distinct] 字段名 [order by 排序字段 asc/desc] [separator '分隔符'])
```

说明：

(1) 使用distinct可以排除重复值；
(2) 如果需要对结果中的值进行排序，可以使用order by子句；
(3) separator是一个字符串值，默认为逗号。

```
create table emp(
emp_id int primary key auto_increment comment '编号',
emp_name char(20) not null default '' comment '姓名',
salary decimal(10,2) not null default 0 comment '工资',
department char(20) not null default '' comment '部门'
);

insert into emp(emp_name,salary,department) values('张晶晶',5000,'财务部'),('王飞飞',5800,'财务部'),('赵刚',6200,'财务部'),
('刘小贝',5700,'人事部'),('王大鹏',6700,'人事部'),('张小斐',5200,'人事部'),('刘云云',7500,'销售部'),('刘云鹏',7200,'财务部'),('刘云鹏',7800,'财务部');

-- 将所有员工的名字合并成一行
select GROUP_CONCAT(emp_name) from emp;

-- 指定分隔符合并
select GROUP_CONCAT(emp_name separator ';') from emp;

-- 指定排序方式和分隔符
select department,group_concat(emp_name separator ';') from emp group by department;
select department,group_concat(emp_name order by salary desc separator ';') from emp group by department;
```

mysql函数-数学函数

MySQL的函数-数学函数

函数名	描述	实例
ABS(x)	返回 x 的绝对值	返回 -1 的绝对值： SELECT ABS(-1) -- 返回1
CEIL(x)	返回大于或等于 x 的最小整数	SELECT CEIL(1.5) -- 返回2
FLOOR(x)	返回小于或等于 x 的最大整数	小于或等于 1.5 的整数： SELECT FLOOR(1.5) -- 返回1
GREATEST(expr1, expr2, expr3, ...)	返回列表中的最大值	返回以下数字列表中的最大值： SELECT GREATEST(3, 12, 34, 8, 25); -- 34 返回以下字符串列表中的最大值： SELECT GREATEST("Google", "Runoob", "Apple"); -- Runoob
LEAST(expr1, expr2, expr3, ...)	返回列表中的最小值	返回以下数字列表中的最小值： SELECT LEAST(3, 12, 34, 8, 25); -- 3 返回以下字符串列表中的最小值： SELECT LEAST("Google", "Runoob", "Apple"); -- Apple

```sql
-- 数学函数
-- 绝对值
select abs(-10);
select abs(表达式或者字段) from 表;

-- 向上取整
select ceil(1.1);  -- 2
select ceil(1.0);  -- 1

-- 向下取整
select floor(1.1);  -- 1
select floor(1.9);  -- 1

-- 取列表最大值
select greatest(1,2,3);  -- 3

-- 取列表最小值
select least(1,2,3); -- 1

-- 求余数
select mod(5,2); -- 1

-- 取x的y次方
select power(2,3); -- 8

-- 取随机数
-- 0~100之间
select floor(rand()*100);

-- 取小数的四舍五入
select round(3.5415);
-- 保留三位小数
select round(3.5415,3)
```

mysql函数-字符串函数

MySQL的函数-字符串函数

函数	描述	实例
CHAR_LENGTH(s)	返回字符串 s 的字符数	返回字符串 RUNOOB 的字符数 SELECT CHAR_LENGTH("RUNOOB") AS LengthOfString;
CHARACTER_LENGTH(s)	返回字符串 s 的字符数	返回字符串 RUNOOB 的字符数 SELECT CHARACTER_LENGTH("RUNOOB") AS LengthOfString;
CONCAT(s1,s2...sn)	字符串 s1,s2 等多个字符串合并为一个字符串	合并多个字符串 SELECT CONCAT("SQL ", "Runoob ", "Gooogle ", "Facebook") AS ConcatenatedString;
CONCAT_WS(x, s1,s2...sn)	同 CONCAT(s1,s2,...) 函数，但是每个字符串之间要加上 x，x 可以是分隔符	合并多个字符串，并添加分隔符： SELECT CONCAT_WS("-", "SQL", "Tutorial", "is", "fun!")AS ConcatenatedString;
FIELD(s,s1,s2...)	返回第一个字符串 s 在字符串列表(s1,s2...)中的位置	返回字符串 c 在列表值中的位置： SELECT FIELD("c", "a", "b", "c", "d", "e");

函数	描述	实例
RIGHT(s,n)	返回字符串 s 的后 n 个字符	返回字符串 runoob 的后两个字符： SELECT RIGHT('runoob',2) -- ob
RTRIM(s)	去掉字符串 s 结尾处的空格	去掉字符串 RUNOOB 的末尾空格： SELECT RTRIM("RUNOOB ") AS RightTrimmedString; -- RUNOOB
STRCMP(s1,s2)	比较字符串 s1 和 s2，如果 s1 与 s2 相等返回 0，如果 s1>s2 返回 1，如果 s1<s2 返回 -1	比较字符串： SELECT STRCMP("runoob", "runoob"); -- 0
SUBSTR(s, start, length)	从字符串 s 的 start 位置截取长度为 length 的子字符串	从字符串 RUNOOB 中的第 2 个位置截取 3 个 字符： SELECT SUBSTR("RUNOOB", 2, 3) AS ExtractString; -- UNO
SUBSTRING(s, start, length)	从字符串 s 的 start 位置截取长度为 length 的子字符串	从字符串 RUNOOB 中的第 2 个位置截取 3 个 字符： SELECT SUBSTRING("RUNOOB", 2, 3) AS ExtractString; -- UNO

```sql
-- 字符串函数
-- 获取字符串字符个数
select char_length('hello');  -- 5
select char_length('长职');  -- 2

-- length 取长度 返回的单位是字节
select length('hello'); -- 5
select length('长职'); -- 6

-- 字符串合并
select concat('hello','world');
select concat(c1,c2) from table_name;

-- 指定分隔符进行字符串合并
select concat_ws('-','hello','world');

-- 返回字符串在列表中第一次出现的位置
select field('aa','aa','bb','cc');

-- 去除字符串左边空格
select ltrim(' aaaa');
```

```sql
-- 去除字符串右边空格
select rtrim(' aaaa   ');
-- 去除两端空格
select trim(' aaaaa ');

-- 字符串截取
select mid("helloworld",2,3);  -- 从第二个字符开始截取 截取长度为3

-- 获取字符串A在字符串中出现的位置
select position('abc' in 'hellabcoworld');

-- 字符串替换
select replace('helloaaaworld','aaa','bbb');

-- 字符串反转
select reverse('abc');

-- 返回字符串后几个字符
select right('hello',3);  -- 返回后3个字符

-- 字符串比较
select strcmp('hello','world');  -- -1 world比hello大

-- 字符串截取
select substr('hello',2,3);  -- 从第二个字符开始截取 截取三个字符
select substring('hello',2,3);  -- 从第二个字符开始截取 截取三个字符

-- 将小写转大写
select ucase('helloworld');
select upper('helloworld');

-- 将大写转为小写
select lcase('HELLOWORLD');
select lower('HELLOWORLD');
```

mysql函数-日期函数

MySQL的函数-日期函数

函数名	描述	实例
UNIX_TIMESTAMP()	返回从1970-01-01 00:00:00到当前毫秒值	select UNIX_TIMESTAMP() -> 1632729059
UNIX_TIMESTAMP(DATE_STRING)	将制定日期转为毫秒值时间戳	SELECT UNIX_TIMESTAMP('2011-12-07 13:01:03');
FROM_UNIXTIME(BIGINT UNIXTIME[, STRING FORMAT])	将毫秒值时间戳转为指定格式日期	SELECT FROM_UNIXTIME(1598079966,'%Y-%m-%d %H:%i:%s'); (1598079966,'%Y-%m-%d %H:%i:%s'); -> 2020-08-22 15-06-06
CURDATE()	返回当前日期	SELECT CURDATE(); -> 2018-09-19
CURRENT_DATE()	返回当前日期	SELECT CURRENT_DATE(); -> 2018-09-19

函数名	描述	实例
TIMEDIFF(time1, time2)	计算时间差值	SELECT TIMEDIFF("13:10:11", "13:10:10"); -> 00:00:01
DATE_FORMAT(d,f)	按表达式 f的要求显示日期 d	SELECT DATE_FORMAT('2011-11-11 11:11:11','%Y-%m-%d %r') -> 2011-11-11 11:11:11 AM
STR_TO_DATE(string, format_mask)	将字符串转变为日期	SELECT STR_TO_DATE("August 10 2017", "%M %d %Y"); -> 2017-08-10
DATE_SUB(date,INTERVAL expr type)	函数从日期减去指定的时间间隔。	Orders 表中 OrderDate 字段减去 2 天： SELECT OrderId,DATE_SUB(OrderDate,INTERVAL 2 DAY) AS OrderPayDate FROM Orders

```sql
-- 日期函数
-- 获取时间戳(毫秒值)
select unix_timestamp();

-- 将一个日期字符串转为毫秒值
select unix_timestamp('2023-10-1 08:08:08');

-- 将时间戳毫秒值转为指定格式的日期
select from_unixtime(1696118888,'%Y-%m-%d %H:%i:%s');

-- 获取当前的年月日
select curdate();
select current_date();

-- 获取当前的时分秒
select current_time();
select curtime();

-- 获取年月日 和 时分秒
select current_timestamp();

-- 从日期字符串中获取年月日
select date('2023-9-30 12:59:59');

-- 获取日期之间的差值
select datediff(current_date(),'2008-08-08');

-- 获取时分秒之间的差值
select timediff('23:08:09','13:01:22');

-- 日期格式化
select date_format('2021-1-1 01:12:01','%Y-%m-%d %h:%i:%s');

-- 将字符串转为日期
select str_to_date('2021-12-13 11:11:11','%Y-%m-%d %H:%i:%s');

-- 将日期进行减法
select date_sub('2021-10-01',interval 2 day);
select date_sub('2021-10-01',interval 2 month);

-- 将日期进行加法
select date_add('2021-10-01',interval 2 day);
select date_add('2021-10-01',interval 2 month);

-- 从日期中获取小时
```

```sql
select extract(hour from '2021-12-13 11:12:13');
select extract(year from '2021-12-13 11:12:13');
select extract(month from '2021-12-13 11:12:13');

-- 获取给定日期所在月的最后一天
select last_day('2021-8-13');

-- 获取指定年份和天数的日期
select makedate('2021',53);

-- 根据日期获取年月日 时分秒
select year('2021-12-13 11:12:13');
select month('2021-12-13 11:12:13');
select minute('2021-12-13 11:12:13');
select quarter('2021-12-13 11:12:13');  -- 获取季度

-- 根据日期获取信息
select monthname('2021-12-13 12:12:13');
select dayname('2021-12-13 12:12:13');  -- 获取周几 Monday
select dayofmonth('2021-12-13 12:12:13');  -- 当月的第几天
select dayofweek('2021-12-13 12:12:13');  -- 1周日 2周一
select dayofyear('2021-12-13 12:12:13');  -- 获取一年的第几天

select week('2021-12-13 12:12:13');
select week('2021-01-01 12:12:13');
select week('2021-12-31 12:12:13');

select yearweek('2021-3-01');

select now();
```

MySQL函数-控制流函数

MySQL的函数-控制流函数

◆ if逻辑判断语句

格式	解释	案例
IF(expr,v1,v2)	如果表达式 expr 成立，返回结果 v1；否则，返回结果 v2。	SELECT IF(1 > 0,'正确','错误') ->正确
IFNULL(v1,v2)	如果 v1 的值不为 NULL，则返回 v1，否则返回 v2。	SELECT IFNULL(null,'Hello Word') ->Hello Word
ISNULL(expression)	判断表达式是否为 NULL	SELECT ISNULL(NULL); ->1
NULLIF(expr1, expr2)	比较两个字符串，如果字符串 expr1 与 expr2 相等 返回 NULL，否则返回 expr1	SELECT NULLIF(25, 25); ->

格式	解释	操作
CASE expression WHEN condition1 THEN result1 WHEN condition2 THEN result2 ... WHEN conditionN THEN resultN ELSE result END	CASE 表示函数开始，END 表示函数结束。如果 condition1 成立，则返回 result1, 如果 condition2 成立，则返回 result2，当全部不成立则返回 result，而当有一个成立之后，后面的就不执行了。	select case 100 when 50 then 'tom' when 100 then 'mary'else 'tim' end ; select case when 1=2 then 'tom' when 2=2 then 'mary' else'tim' end ;

```sql
-- 控制流函数
-- if
select if(5>3,'大于','小于');
select name,if(chinese >= 80 ,'优秀','良好') flag from student;

-- ifnull
select ifnull(5,0);
select ifnull(NULL,0);
select *,ifnull(comm,0) conmm_flag from emp;

-- ifnull
select isnull(5);
select isnull(NULL);

-- nullif
select nullif(12,12);  -- null
select nullif(12,13);  -- 12

-- case when
select
   case 5
      when 1 then 'hello'
      when 2 then 'world'
      when 5 then '!!!'
        else
          'other'
    end as info;

    select
   case
      when 2>1 then 'hello'
      when 2<1 then 'world'
      when 5>3 then '!!!'
        else
          'other'
    end as info;

  -- 创建表
  create table orders(
  oid int primary key,
  price double,
  payType int  -- 1微信 2支付宝 3银行卡 4其他
  );

  insert into orders values(1,1200,1);
  insert into orders values(2,1000,2);
  insert into orders values(3,200,3);
```

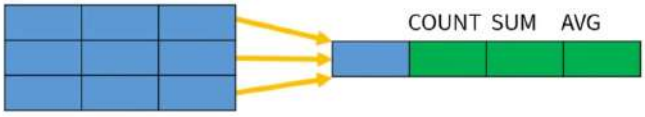
```sql
  insert into orders values(4,3000,1);
  insert into orders values(5,1500,2);

--  方式1
select
   *,
   case payType
         when 1 then '微信'
         when 2 then '支付宝'
         when 3 then '银行卡'
         else
           '其他支付方式'
      end as payTypeStr
    from orders;

--  方式2
select
  *,
    case
      when payType>0 then '微信'
      when payType>1 then '支付宝'
      when payType>2 then '银行卡'
        else
          '其他支付方式'
      end as payTypeStr
    from orders;
```

MySQL函数-窗口函数

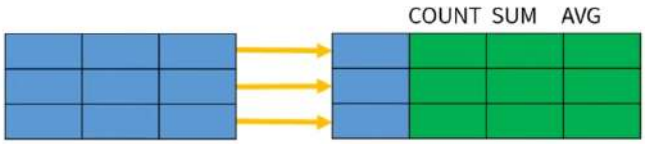
MySQL的函数-窗口函数

◆ 介绍

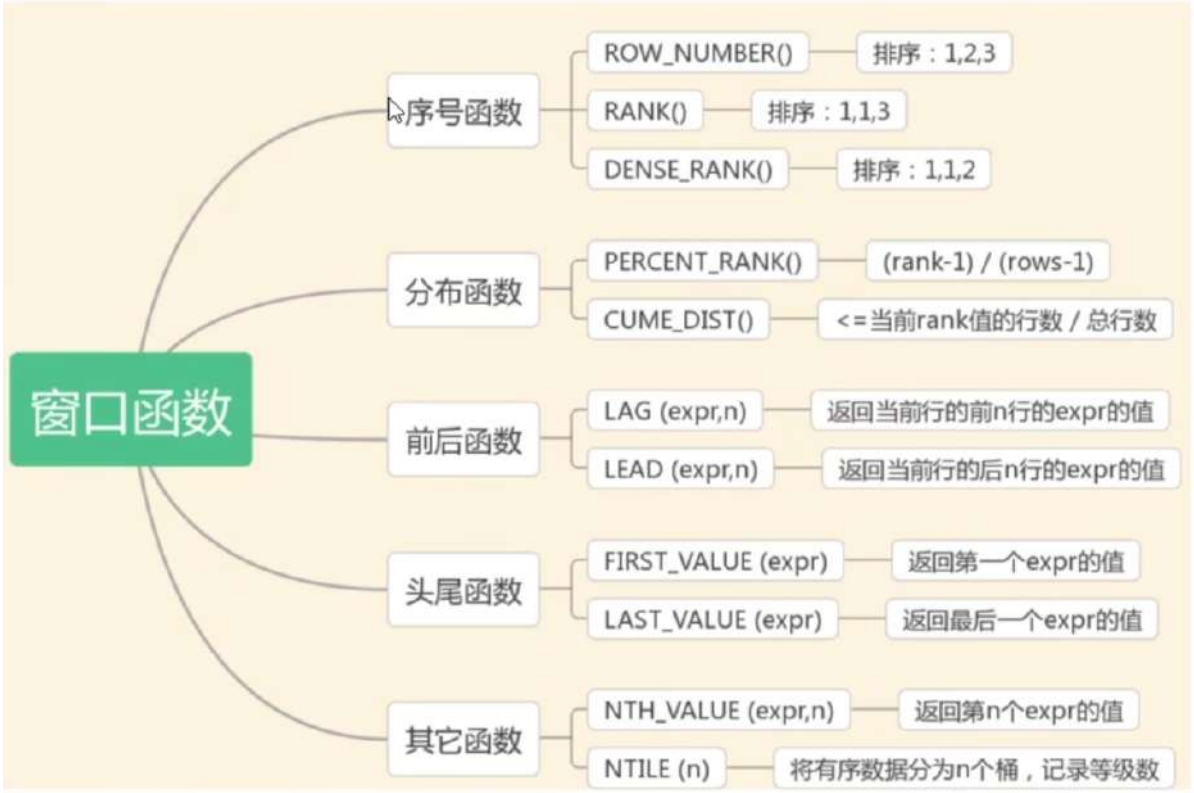
- MySQL 8.0 新增窗口函数,窗口函数又被称为开窗函数，与Oracle 窗口函数类似，属于MySQL的一大特点.
- 非聚合窗口函数是相对于聚函数来说的。聚合函数是对一组数据计算后返回单个值（即分组），非聚合函数一次只会处理一行数据。窗口聚合函数在行记录上计算某个字段的结果时，可将窗口范围内的数据输入到聚合函数中，并不改变行数。



聚合函数



窗口函数



MySQL的函数-窗口函数

◆ 语法结构

```
window_function ( expr ) OVER (
  PARTITION BY ...
  ORDER BY ...
  frame_clause
)
```

其中，window_function 是窗口函数的名称；expr 是参数，有些函数不需要参数；OVER子句包含三个选项：

➢ 分区（PARTITION BY）

PARTITION BY选项用于将数据行拆分成多个分区（组），它的作用类似于GROUP BY分组。如果省略了 PARTITION BY，所有的数据作为一个组进行计算

➢ 排序（ORDER BY）

OVER 子句中的ORDER BY选项用于指定**分区内**的排序方式，与 ORDER BY 子句的作用类似

➢ 以及窗口大小（frame_clause）。

frame_clause选项用于在**当前分区**内指定一个计算窗口，也就是一个与当前行相关的数据子集。

mysql函数-序号函数

◆ 序号函数

序号函数有三个：ROW_NUMBER()、RANK()、DENSE_RANK()，可以用来实现分组排序，并添加序号

➢ 格式

```
row_number()|rank()|dense_rank() over (
  partition by ...
  order by ...
)
```

➢ 操作

```
use mydb4;
create table employee(
    dname varchar(20), -- 部门名
    eid varchar(20),
    ename varchar(20),
    hiredate date, -- 入职日期
    salary double -- 薪资
);
```

```
create table employee(
dname varchar(20),
eid varchar(20),
ename varchar(20),
hiredate date,
salary double
);

insert into employee values('研发部','1001','刘备','2021-11-01',3000);
insert into employee values('研发部','1002','关羽','2021-11-02' ,5000);
insert into employee values( '研发部','1003','张飞','2021-11-03',7000);
insert into employee values ('研发部','1004','赵云','2021-11-04',7000);
insert into employee values( '研发部','1005','马超','2021-11-05',4000);
insert into employee values( '研发部','1006','黄忠','2021-11-06',4900);
insert into employee values( '销售部','1007','曹操','2021-11-01',2000);
insert into employee values( '销售部','1008','许褚','2021-11-02',3000);
insert into employee values( '销售部','10091','典韦','2021-11-03',5000);
insert into employee values('销售部','1010','张辽','2021-11-04',6000);
insert into employee values( '销售部','1011','徐晃','2021-11-05',9000);
insert into employee values('销售部','1012','曹洪','2021-11-06',6000);

-- 对每个部门的员工按照薪资排序 并给出排名 row_number()
select dname,ename,salary,row_number() over(partition by dname order by
salary desc) as rn from employee;

-- 对每个部门的员工按照薪资排序 并给出排名 rank
select dname,ename,salary,rank() over(partition by dname order by salary
desc) as rn from employee;

-- 对每个部门的员工按照薪资排序 并给出排名 dense-rank
select dname,ename,salary,dense_rank() over(partition by dname order by
salary desc) as rn from employee;

```

```sql
select
dname,ename,salary,
            row_number() over(partition by dname order by salary desc) as rn1,
            rank() over(partition by dname order by salary desc) as rn2,
            dense_rank() over(partition by dname order by salary desc) as rn3
from employee;

-- 求出每个部门薪资排在前三名的员工 分组求TOPN
select * from
(select dname,ename,salary,dense_rank() over(partition by dname order by salary desc) as rn from employee)t
where t.rn <=3;

-- 对所有员工进行全局排序(不分组)
select * from (select dname,ename,salary,dense_rank() over(order by salary desc) as rn from employee)t
where t.rn=1;
```

MySQL函数-开窗聚合函数

◆ 开窗聚合函数- SUM,AVG,MIN,MAX

➢ 概念

在窗口中每条记录动态地应用聚合函数（SUM()、AVG()、MAX()、MIN()、COUNT()），可以动态计算在指定的窗口内的各种聚合函数值。

➢ 操作

```sql
select
 dname,
 ename,
 salary,
 sum(salary) over(partition by dname order by hiredate) as pv1
from employee;

select cookieid,createtime,pv,
sum(pv) over(partition by cookieid) as pv3
from itcast_t1;  -- 如果没有order by排序语句 默认把分组内的所有数据进行sum操作
```

```sql
-- 开窗聚合函数 sum
select dname,ename,salary,sum(salary) over(partition by dname order by hiredate) as pv1 from employee;

-- 如果没有order by 排序语句 默认把分组内的所有数据进行sum操作
select dname,ename,hiredate,salary,sum(salary) over(partition by dname) as pv1 from employee;

-- 从第一行开始加到当前行
select dname,ename,hiredate,salary,sum(salary) over(partition by dname order by hiredate rows between unbounded preceding and current row) as pv1 from employee;

-- 从向上三行加到当前行
```

```sql
select dname,ename,hiredate,salary,sum(salary) over(partition by dname order by hiredate rows between 3 preceding and current row) as pv1 from employee;

-- 从向上三行 向后一行 相加
select dname,ename,hiredate,salary,sum(salary) over(partition by dname order by hiredate rows between 3 preceding and 1 following) as pv1 from employee;

-- 从当前行加到最后
select dname,ename,salary,sum(salary) over(partition by dname order by hiredate rows between current row and unbounded following) as pv1 from employee;

-- 求平均值
select dname,ename,salary,avg(salary) over(partition by dname order by hiredate) as pv1 from employee;

-- 求最大值
select dname,ename,salary,max(salary) over(partition by dname order by hiredate) as pv1 from employee;
```

mysql窗口函数-分布函数

◆ 分布函数- CUME_DIST和PERCENT_RANK

➢ 介绍-CUME_DIST

- 用途：分组内小于、等于当前rank值的行数 / 分组内总行数
- 应用场景：查询小于等于当前薪资（salary）的比例

➢ 操作

```sql
select
 dname,
 ename,
 salary,
 cume_dist() over(order by salary) as rn1, -- 没有partition语句 所有的数据位于一组
 cume_dist() over(partition by dept order by salary) as rn2
from employee;
```

◆ 分布函数- CUME_DIST和PERCENT_RANK

➢ 介绍-PERCENT_RANK

- 用途：每行按照公式(rank-1) / (rows-1)进行计算。其中，rank为RANK()函数产生的序号，rows为当前窗口的记录总行数
- 应用场景：不常用

➢ 操作

```sql
select
 dname,
 ename,
 salary,
 rank() over(partition by dname order by salary desc ) as rn,
 percent_rank() over(partition by dname order by salary desc ) as rn2
from employee;
```

```sql
-- cume_dist
select dname,ename,salary,cume_dist() over(order by salary) as rn1,
                          cume_dist() over(partition by dname order by salary)
as rn2
from employee;

-- percent_rank
select dname,ename,salary,rank() over(partition by dname order by salary desc)
as rn,
                          percent_rank() over(partition by dname order by
salary desc) as rn2
from employee;
```

mysql窗口函数-前后函数

◆ 前后函数-LAG和LEAD

➢ 介绍

•用途：返回位于当前行的前n行（LAG(expr,n)）或后n行（LEAD(expr,n)）的expr的值
•应用场景：查询前1名同学的成绩和当前同学成绩的差值

➢ 操作

```sql
-- lag的用法
select
 dname,
 ename,
 hiredate,
 salary,
 lag(hiredate,1,'2000-01-01') over(partition by dname order by hiredate) as
last_1_time,
 lag(hiredate,2) over(partition by dname order by hiredate) as last_2_time
from employee;
```

```sql
-- 前后函数
-- lag的用法
select dname,ename,hiredate,salary,lag(hiredate,1,'2000-01-01')
over(partition by dname order by hiredate) as last_1_time,
       lag(hiredate,2) over(partition by dname order by hiredate) as
last_2_time
             from employee;

-- lead的用法
select dname,ename,hiredate,salary,lead(hiredate,1,'2000-01-01')
over(partition by dname order by hiredate) as time1,
             lead(hiredate,2) over(partition by dname order by hiredate) as
time2
             from employee;
```

mysql窗口函数-头尾函数

◆ 头尾函数-FIRST_VALUE和LAST_VALUE

➢ 介绍

•用途：返回第一个（FIRST_VALUE(expr)）或最后一个（LAST_VALUE(expr)）expr的值
•应用场景：截止到当前，按照日期排序查询第1个入职和最后1个入职员工的薪资

➢ 操作

```
-- 注意，  如果不指定ORDER BY，则进行排序混乱，会出现错误的结果
select
  dname,
  ename,
  hiredate,
  salary,
  first_value(salary) over(partition by dname order by hiredate) as first,
  last_value(salary) over(partition by dname order by  hiredate) as last
from  employee;
```

```
-- 头尾函数
-- 如果不指定order by 则进行排序混乱 会出现错误
select dname,ename,hiredate,salary,first_value(salary) over(partition by dname order by hiredate) as first,
            last_value(salary) over(partition by dname order by hiredate) as last
            from employee;
```

MySQL窗口函数-其他函数

◆ 其他函数-NTH_VALUE(expr, n)、NTILE(n)

➢ 介绍-NTH_VALUE(expr,n)

•用途：返回窗口中第n个expr的值。expr可以是表达式，也可以是列名
•应用场景：截止到当前薪资，显示每个员工的薪资中排名第2或者第3的薪资

➢ 操作

```
-- 查询每个部门截止目前薪资排在第二和第三的员工信息
select
  dname,
  ename,
  hiredate,
  salary,
  nth_value(salary,2) over(partition by dname order by hiredate) as second_score,
  nth_value(salary,3) over(partition by dname order by hiredate) as third_score
from employee
```

```sql
-- 其他函数
-- NTH_VALUE(expr,n)
-- 查询每个部门截至目前薪资排在第二和第三的员工信息
select dname,ename,hiredate,salary,nth_value(salary,2) over(partition by dname order by hiredate) as second_score,
       nth_value(salary,3) over(partition by dname order by hiredate) as third_score
             from employee;

-- ntile
select dname,ename,salary,hiredate,ntile(3) over(partition by dname order by hiredate) as nt from employee;

-- 取出每一个部门的第一组员工
select * from(select dname,ename,salary,hiredate,ntile(3) over(partition by dname order by hiredate) as nt from employee)t
         where t.nt=1;
```

Mysql的视图

视图创建

◆ 视图的创建

创建视图的语法为：

```sql
create [or replace] [algorithm = {undefined | merge | temptable}]

view view_name [(column_list)]

as select_statement

[with [cascaded | local] check option]
```

参数说明：
(1) algorithm：可选项，表示视图选择的算法。
(2) view_name：表示要创建的视图名称。
(3) column_list：可选项，指定视图中各个属性的名词，默认情况下与SELECT语句中的查询的属性相同。
(4) select_statement
：表示一个完整的查询语句，将查询记录导入视图中。
(5) **[with [cascaded | local] check option]**：可选项，表示更新视图时要保证在该视图的权限范围之内。

➢ 数据准备

创建 数据库mydb6_view,然后在该数据库下执行sql脚本view_data.sql 导入数据

```sql
create database mydb6_view;
```

➢ 操作

```sql
create or replace view view1_emp
as
select ename,job from emp;

-- 查看表和视图
show full tables;
```

```sql
-- mysql视图
-- 视图创建
create table dept(
```

```
    deptno int primary key,
  dname varchar(20),
    loc varchar(20)
);

insert into dept values(10, '教研部','北京'),
(20, '学工部','上海'),
(30, '销售部','广州'),
(40, '财务部','武汉');

create table emp(
    empno int primary key,
    ename varchar(20),
    job varchar(20),
    mgr int,
    hiredate date,
    sal numeric(8,2),
    comm numeric(8, 2),
    deptno int,
    FOREIGN KEY (deptno) REFERENCES dept(deptno) ON DELETE SET NULL ON UPDATE
CASCADE
);

insert into emp values
(1001, '甘宁', '文员', 1013, '2000-12-17', 8000.00, null, 20),
(1002, '黛绮丝', '销售员', 1006, '2001-02-20', 16000.00, 3000.00, 30),
(1003, '殷天正', '销售员', 1006, '2001-02-22', 12500.00, 5000.00, 30),
(1004, '刘备', '经理', 1009, '2001-4-02', 29750.00, null, 20),
(1005, '谢逊', '销售员', 1006, '2001-9-28', 12500.00, 14000.00, 30),
(1006, '关羽', '经理', 1009, '2001-05-01', 28500.00, null, 30),
(1007, '张飞', '经理', 1009, '2001-09-01', 24500.00, null, 10),
(1008, '诸葛亮', '分析师', 1004, '2007-04-19', 30000.00, null, 20),
(1009, '曾阿牛', '董事长', null, '2001-11-17', 50000.00, null, 10),
(1010, '韦一笑', '销售员', 1006, '2001-09-08', 15000.00, 0.00, 30),
(1011, '周泰', '文员', 1008, '2007-05-23', 11000.00, null, 20),
(1012, '程普', '文员', 1006, '2001-12-03', 9500.00, null, 30),
(1013, '庞统', '分析师', 1004, '2001-12-03', 30000.00, null, 20),
(1014, '黄盖', '文员', 1007, '2002-01-23', 13000.00, null, 10);

create table salgrade(
    grade int primary key,
    losal int,
    hisal int
);

insert into salgrade values
(1, 7000, 12000),
(2, 12010, 14000),
(3, 14010, 20000),
(4, 20010, 30000),
(5, 30010, 99990);

create or replace view view1_emp
as
select ename,job from emp;

```

```sql
-- 查看表和视图
show full tables;

select * from view1_emp;
```

修改视图

◆ 修改视图

修改视图是指修改数据库中已存在的表的定义。当基本表的某些字段发生改变时，可以通过修改视图来保持视图和基本表之间一致。MySQL中通过CREATE OR REPLACE VIEW语句和ALTER VIEW语句来修改视图。

➢ 格式

```sql
alter view 视图名 as select语句
```

➢ 操作

```sql
alter view view1_emp
as
select a.deptno,a.dname,a.loc,b.ename,b.sal from dept a, emp b where
a.deptno = b.deptno;
```

```sql
-- 修改视图
alter view view1_emp
as
select a.deptno,a.dname,a.loc,b.ename,b.sal from dept a,emp b where a.deptno =
b.deptno;

select * from view1_emp;
```

更新视图

◆ 更新视图

某些视图是可更新的。也就是说，可以在UPDATE、DELETE或INSERT等语句中使用它们，以更新基表的内容。对于可更新的视图，在视图中的行和基表中的行之间必须具有一对一的关系。如果视图包含下述结构中的任何一种，那么它就是不可更新的：

- 聚合函数（SUM(), MIN(), MAX(), COUNT()等）
- DISTINCT
- GROUP BY
- HAVING
- UNION或UNION ALL
- 位于选择列表中的子查询
- JOIN
- FROM子句中的不可更新视图
- WHERE子句中的子查询，引用FROM子句中的表。
- 仅引用文字值（在该情况下，没有要更新的基本表）

视图中虽然可以更新数据，但是有很多的限制。一般情况下，最好将视图作为查询数据的虚拟表，而不要通过视图更新数据。因为，使用视图更新数据时，如果没有全面考虑在视图中更新数据的限制，就可能会造成数据更新失败。

```sql
-- 更新视图
create or replace view view1_emp
as
select ename,job from emp;
select * from view1_emp;

update view1_emp set ename = '周瑜' where ename ='鲁肃';
insert into view1_emp values('周瑜','文员');

-- 包含聚合函数不可更新
```

```sql
create or replace view view2_emp
as
select count(*)cnt from emp;

select * from view2_emp;

insert into view2_emp values(100);
update view2_emp set cnt=100;

-- 视图包含distinct不可更新
create or replace view view3_emp
as
select distinct job from emp;

insert into view3_emp values('财务');

-- 视图包含group by不可更新
create or replace view view4_emp
as
select deptno,count(*) from emp group by deptno;

insert into view4_emp values(30,100);

-- 视图包含having不可更新
create or replace view view5_emp
as
select deptno,count(*) cnt from emp group by deptno having cnt >2;

insert into view5_emp values(30,100);

-- 视图包含union不可更新
create or replace view view6_emp
as
select empno,ename from emp where empno <=5
union
select empno,ename from emp where empno >5;

insert into view6_emp values(1015,'韦小宝');

-- 视图包含子查询不可更新
create or replace view view7_emp
as
select empno,ename,sal from emp where sal = (select max(sal) from emp);

insert into view7_emp values(1015,'韦小宝',30000);

-- 视图包含join不可更新
create or replace view view8_emp
as
select dname,ename,sal from emp a join dept b on a.deptno=b.deptno;

insert into view8_emp(dname,ename,sal) values('行政部','韦小宝',30000);

-- 视图引用文字值不可更新
create or replace view view8_emp
as
```

```
select '行政部' dname,'杨过' ename;

insert into view8_emp values('行政部','韦小宝');
```

其他操作

◆ 其他操作

➢ 重命名视图

```
-- rename table 视图名 to 新视图名;
rename table view1_emp to my_view1
```

➢ 删除视图

```
-- drop view 视图名[,视图名…];
drop view if exists view_student;
```

删除视图时，只能删除视图的定义，不会删除数据。

```
-- 其他操作
-- 重命名视图
rename table view1_emp to myview1;

-- 删除视图
drop view if exists myview1;
```

MySQL的视图练习

```
-- 查询部门平均薪水最高的年薪
SELECT
    a.deptno,
    a.dname,
    a.loc,
    ttt.avg_sal
FROM
    dept a,
    (
SELECT
    *
FROM
    (
SELECT
    *,
    rank ( ) over ( ORDER BY avg_sal DESC ) rn
FROM
    ( SELECT deptno, avg( deptno ) avg_sal FROM emp GROUP BY deptno ) t
    ) tt
WHERE
    rn = 1
    ) ttt
WHERE
    a.deptno = ttt.deptno;
```

```sql
-----------------------------------------------------------------
-- 创建视图优化
create view test_view1
as
select deptno,avg(deptno) avg_sal from emp group by deptno

create view test_view2
as
select *,rank() over (order by avg_sal desc) rn from test_view1

create view test_view3
as
select * from test_view2 tt where rn=1

select a.deptno,a.dname,a.loc,avg_sal from dept a,test_view3 ttt where
a.deptno = ttt.deptno;
-----------------------------------------------------------------
create view test_view11
as
SELECT
    a.deptno,
    a.dname,
    a.loc,
    ttt.avg_sal
FROM
    dept a,
    (
SELECT
    *
FROM
    (
SELECT
    *,
    rank ( ) over ( ORDER BY avg_sal DESC ) rn
FROM
    ( SELECT deptno, avg( deptno ) avg_sal FROM emp GROUP BY deptno ) t
    ) tt
WHERE
    rn = 1
    ) ttt
WHERE
    a.deptno = ttt.deptno;

select * from test_view11;
-----------------------------------------------------------------
-- 查询员工比所属领导薪资高的部门名 员工名 员工领导编号
-- 1查询员工比领导工资高的部门号
create view test_view4
as
SELECT
    a.ename ename,
    a.sal esal,
    b.ename mgrname,
    b.sal msal,
    a.deptno
FROM
```

```
    emp a,
    emp b
WHERE
    a.mgr = b.empno
    AND a.sal > b.sal;
-- 2查询出来的部门号和部门表进行链表查询
select * from dept a join test_view4 b on a.deptno = b.deptno;
-----------------------------------------------------------------
-- 查询工资等级为4级 2000年以后入职的工作地点为上海的员工编号 姓名和工资 并查询出薪资在前三名的员工信息
-- 1查询工资等级为4级 2000年以后入职的工作地点为上海的员工编号 姓名和工资
create view test_view5
as
SELECT
    a.deptno,
    a.dname,
    a.loc,
    b.empno,
    b.ename,
    b.sal,
    c.grade
FROM
    dept a,
    emp b,
    salgrade c
WHERE
    a.deptno = b.deptno
    AND grade = 4
    AND ( b.sal BETWEEN c.losal AND c.hisal )
    AND YEAR ( hiredate ) > '2000'
    AND a.loc = '上海';

    select * from(select *,rank() over(order by sal desc)rn from test_view5)t where rn<=3;
```

MySQL的存储过程

入门案例

◆ 入门案例

➢ 格式

```
delimiter 自定义结束符号
create procedure 储存名([ in ,out ,inout ] 参数名 数据类形...)
begin
  sql语句
end 自定义的结束符合
delimiter ;
```

➢ 操作-数据准备

```
-- 1: 创建数据库
create database mydb7_procedure;

-- 2: 在该数据库下导入sql脚本:procedure_data.sql
```

```sql
-- 入门案例
delimiter $$
create procedure proc01()
begin
 select empno,ename from emp;
end $$
delimiter ;

-- 调用存储过程
call proc01();
```

MySQL操作-局部变量

◆ MySQL操作-变量定义

➢ 格式

- 局部变量

  用户自定义，在begin/end块中有效

```
语法： 声明变量 declare var_name type [default var_value];
举例：declare nickname varchar(32);
```

➢ 操作

```sql
delimiter $$
create procedure proc02()
begin
    declare var_name01 varchar(20) default 'aaa';
    set var_name01 = 'zhangsan';
    select var_name01;
end $$
-- 调用存储过程
call proc02()
```

```sql
-- MySQL操作 变量定义 局部变量
delimiter $$
create procedure proc02()
begin
  declare var_name01 varchar(20) default 'aaa';
  set var_name01 = 'zhangsan';
    select var_name01;
end $$

delimiter ;

call proc02();
```

mysql变量-selet into

◆ MySQL操作-变量定义

➢ 操作

MySQL 中还可以使用 **SELECT..INTO** 语句为变量赋值。其基本语法如下：

```
delimiter $$
create procedure proc03()
begin
  declare my_ename varchar(20) ;
  select ename into my_ename from emp where empno=1001;
    select my_ename;
end $$
-- 调用存储过程
call proc03()
```

```
delimiter $$
create procedure proc03()
begin
 declare my_ename varchar(20);
 select ename into my_ename from emp where empno = 1001;
 select my_ename;
end $$
delimiter ;

call proc03();
```

MySQL操作-用户变量

◆ MySQL操作-变量定义

- 用户变量

➢ 格式

用户自定义，当前会话（连接）有效。类比java的成员变量

```
语法：
@var_name
不需要提前声明，使用即声明
```

➢ 操作

```
-- 赋值
delimiter $$
create procedure proc04()
begin
    set @var_name01  = 'ZS';
end $$
call proc04() $$
select @var_name01  $$  --可以看到结果
```

```
-- 用户变量
delimiter $$
create procedure proc04()
begin
  set @var_name01='ZS';
end $$
delimiter ;

call proc04();
select @var_name01;
```

MySQ操作-全局变量


```
-- 查看全局变量
show global variables;

-- 查看某全局变量
select @@global.auto_increment_increment;

-- 修改全局变量的值
set global sort_buffer_size = 40000;
set @@global.sort_buffer_size = 40000;

select @@global.sort_buffer_size;
```

MySQL操作-会话变量

◆ MySQL操作-变量定义

- 系统变量-会话变量

  由系统提供，当前会话（连接）有效

➢ 格式

```
语法：
@@session.var_name
```

➢ 操作

```
-- 查看会话变量
show session variables;
-- 查看某会话变量
select @@session.auto_increment_increment;
-- 修改会话变量的值
set session sort_buffer_size = 50000;
set @@session.sort_buffer_size = 50000 ;
```

```
-- 查看会话变量
show session variables;

-- 查看某会话变量
select @@session.auto_increment_increment;

-- 修改会话变量
set session sort_buffer_size = 50000;
set @session.sort_buffer_size = 50000;

select @@session.sort_buffer_size;
```

MySQL存储过程传参-in

◆ 存储过程传参-in

in 表示传入的参数， 可以传入数值或者变量，即使传入变量，并不会更改变量的值，可以内部更改，仅仅作用在函数范围内。

```
-- 封装有参数的存储过程，传入员工编号，查找员工信息
delimiter $$
create procedure dec_param01(in param_empno varchar(20))
begin
        select * from emp where empno = param_empno;
end $$

delimiter ;
call dec_param01('1001');
```

```
-- 存储传参-in
-- 封装有参数的存储过程 传入员工编号 查找员工信息
delimiter $$
create procedure dec_param01(in param_empno varchar(20))
begin
 select * from emp where empno = param_empno;
end $$
delimiter ;
```

```
call dec_param01('1001');

-- 封装有参数的存储过程 可以通过传入部门名和薪资 查询指定部门 并且薪资大于指定值的员工信息
delimiter $$
create procedure dec_param02(in para_dname varchar(50),in para_sal decimal(7,2))
begin
  select * from dept a, emp b where a.deptno = b.deptno and a.dname = param_dname and b.sal > param_sal;
end $$
delimiter ;
call dec_param02('学工部',20000);
call dec_param02('销售部',10000);
```

MySQL存储过程传参-out

◆ 存储过程传参-out

out 表示从存储过程内部传值给调用者

```
-- ---------传出参数：out----------------------------------
use mysql7_procedure;
-- 封装有参数的存储过程，传入员工编号，返回员工名字
delimiter $$
create procedure proc08(in empno int ,out out_ename varchar(50) )
begin
  select ename into out_ename from emp where emp.empno = empno;
end $$

delimiter ;

call proc08(1001, @o_ename);
select @o_ename;
```

```
-- 存储传参-out
-- 封装有参数的存储过程 传入员工编号 返回员工名字
delimiter $$
create procedure proc08(in empno int, out out_ename varchar(50))
begin
 select ename into out_ename from emp where emp.empno = empno;
end $$
delimiter ;

call proc08(1001,@o_ename);
select @o_ename;
```

MySQL存储过程传参-inout

◆ 存储过程传参-inout

inout 表示从外部传入的参数经过修改后可以返回的变量，既可以使用传入变量的值也可以修改变量的值（即使函数执行完）

```
-- 传入员工名，拼接部门号，传入薪资，求出年薪
delimiter $$
create procedure proc10(inout inout_ename varchar(50),inout inout_sal int)
begin
  select  concat(deptno,"_",inout_ename) into inout_ename from emp where ename = inout_ename;
  set inout_sal = inout_sal * 12;
end $$
delimiter ;
set @inout_ename = '关羽';
set @inout_sal = 3000;
call proc10(@inout_ename, @inout_sal) ;
select @inout_ename ;
select @inout_sal ;
```

```
-- 存储过程传参-inout
-- 传入员工名 拼接部门号 传入薪资 求出年薪
delimiter $$
create procedure proc10(inout inout_ename varchar(50),inout inout_sal int)
begin
  select concat(deptno,"_",inout_ename) into inout_ename from emp where ename = inout_ename;
where ename = inout_ename;
  set inout_sal = inout_sal * 12;
end $$
delimiter ;

set @inout_ename='关羽';
set @inout_sal = 3000;

call proc10(@inout_ename,@inout_sal);

select @inout_ename;
select @inout_sal;

-- 传入一个数字 传出这个数字的10倍值
delimiter $$
create procedure proc11(inout num int)
begin
  set num =num *10;
end $$
delimiter ;

set @inout_num = 3;

call proc11(@inout_num);

select @inout_num;
```

MySQL存储过程传参-流程控制 if

```
-- 流程判断
-- 案例1
-- 输入学生的成绩 判断成绩的级别
/*score<60 不及格
score>=60,score<80及格
score>=80,score<90良好
score>=90,score<=100优秀
score>100:成绩错误
*/
delimiter $$
create procedure proc_12_if(in score int)
begin
  if score < 60
       then select '不及格';
  elseif score >=60 and score < 80
       then select '及格';
  elseif score >=80 and score <90
       then select '良好';
  elseif score >=90 and score <=100
       then select '优秀';
  else
       select '成绩错误';
    end if;
end $$
delimiter ;

call proc_12_if(102)
-----------------------------------------------------------------
-- 案例2
-- 输入员工的名字 判断工资的情况
/*
sal < 10000:试用薪资
sal >=10000 and sal <20000 :转正薪资
sal >=20000:元老薪资
*/
delimiter  $$
create procedure proc_13_if(in IN_ename varchar(20))
begin
-- 定义变量
 declare var_sal decimal(7,2);
 declare result varchar(20);
 select sal into var_sal from emp where ename = in_ename;

 if var_sal <10000
    then set result = '试用薪资';
 elseif var_sal <20000
      then set result = '转正薪资';
 else
    set result = '元老薪资';
 end if;
      select result;
end $$
delimiter ;
```

```
54
55 -- 调用
56 call proc_13_if('关羽');
```

MySQL存储过程传参-流程控制 case

```
1  -- 流程控制语句 case
2  /*
3  支付方式
4  1 微信支付
5  2 支付宝支付
6  3 银行卡支付
7  */
8  -- 格式1
9  delimiter $$
10 create procedure proc14_casae(in pay_type int)
11 begin
12   case pay_type
13       when 1 then select '微信支付';
14         when 2 then select '支付宝支付';
15         when 3 then select '银行卡支付';
16         else select '其他方式支付';
17     end case;
18 end $$
19 delimiter ;
20
21 call proc14_casae(2);
22 call proc14_casae(4);
23
24 -- 格式2
25 delimiter $$
26 create procedure proc15_case(in score int)
27 begin
28   case
29     when score <60
30        then select '不及格';
31   when score<80
32      then select '及格';
33     when score >=80 and score <90
34      then select '良好';
35     when score >=90 and score <=100
36      then select '优秀';
37     else
38      select '成绩错误';
39     end case;
40 end $$
41 delimiter ;
42 call proc15_case(100);
```

MySQL存储过程传参-流程控制 while

```
-- 流程控制while
-- 创建测试表
create table user(
uid int primary key,
username varchar(50),
password varchar(50)
);

-- 需求 向表中添加指定条数语句
delimiter $$
create procedure proc16_while(in insertCount int)
begin
    declare i int default 1;
    label:while i<=insertCount do
        insert into user(uid,username,password) values(i,concat('user-',i),'123456');
        set i = i + 1;
    end while label;
end $$
delimiter ;

call proc16_while(10);

-- 循环控制 while + leave
-- 跳出本层循环contiue
truncate table user;

delimiter $$
create procedure proc17_while(in insertCount int)
begin
  declare i int default 1;
    label:while i <=insertCount do
      insert into user(uid,username,password) values(i,concat('user-',i),'123456');
        if i = 5 then
          leave label;
        end if;
        set i = i + 1;
    end while label;
    select '循环结束';
end $$
delimiter ;

call proc17_while(10);

-- 循环控制 while + iterate
-- 跳出本次循环 break
use database1;
truncate table user1;

create table user1(
uid int,
username varchar(50),
```

```
password varchar(50)
);

-- 死循环
delimiter $$
create procedure proc18_while(in insertCount int)
begin
  declare i int default 1;
    label:while i <=insertCount do
      insert into user1(uid,username,password) values(i,concat('user-',i),'456789');
        if i = 5 then
          iterate label;
        end if;
        set i = i+1;
    end while label;
    select '循环结束';
end $$
delimiter ;

call proc18_while(10);

-- 1 2 3 4 6 7 8 9 10
delimiter $$
create procedure proc19_while(in insertCount int)
begin
  declare i int default 0;
    label:while i <insertCount do
        set i = i+1;
        if i = 5 then
            iterate label;
        end if;
        insert into user1(uid,username,password) values(i,concat('user-',i),'789123');
    end while label;
    select '循环结束';
end $$
delimiter ;

call proc19_while(10);
```

MySQL存储过程传参-流程控制 repeat

```
-- 循环流程 repeat
truncate table user;

delimiter $$
create procedure proc20_repeat(in insertCount int)
begin
     declare i int default 1;
     label:repeat
         insert into user(uid,username,password) values(i,concat('user-',i),'9456312');
         set i=i+1;
         until i > insertCount
```

```
      end repeat label;
      select '循环结束';
end $$
delimiter ;

call proc20_repeat(10);
```

MySQL存储过程传参-流程控制 loop

```
--  循环流程 loop
truncate table user;

delimiter $$
create procedure proc21_loop(in insertCount int)
begin
  declare i int default 1;
    label:loop
      insert into user(uid,username,password) values(i,concat('user-',i),'74651231');
        set i=i+1;
        if i>insertCount
          then leave label;
        end if ;
    end loop label;
    select '循环结束';
end $$
delimiter ;

call proc21_loop(10);
```

MySQL存储过程传参-游标 cursor

◆ 游标

游标(cursor)是用来存储查询结果集的数据类型，在存储过程和函数中可以使用光标对结果集进行循环的处理。光标的使用包括光标的声明、OPEN、FETCH 和 CLOSE.

➢ 格式

```
-- 声明语法
declare cursor_name cursor for select_statement
-- 打开语法
open cursor_name
-- 取值语法
fetch cursor_name into var_name [, var_name] ...
-- 关闭语法
close cursor_name
```

```
-- 游标cursor
-- 输入一个部门名,查询该部门员工的编号 名字 薪资,将查询的结果集添加游标
delimiter $$
create  procedure proc22_cursor(in in_dname varchar(50))
begin
-- 定义局部变量
declare var_empno int;
declare var_ename varchar(50);
declare var_sal decimal(7,2);
```

```
10
11  -- 声明游标
12  declare my_cursor cursor for
13    select empno,ename,sal
14      from dept a,emp b
15      where a.deptno = b.deptno and a.dname = in_dname;
16
17  -- 打开游标
18      open my_cursor;
19  -- 通过游标获取值
20      fetch my_cursor into var_empno,var_ename,var_sal;
21      select var_empno,var_ename,var_sal;
22
23  -- 关闭游标
24      close my_cursor;
25  end $$
26
27  delmiter ;
28  call proc22_cursor('销售部');
29  ----------------------------------------------------------------
30  -- 输入一个部门名,查询该部门员工的编号 名字 薪资,将查询的结果集添加游标
31  delimiter $$
32  create  procedure proc22_cursor(in in_dname varchar(50))
33  begin
34  -- 定义局部变量
35  declare var_empno int;
36  declare var_ename varchar(50);
37  declare var_sal decimal(7,2);
38
39  -- 声明游标
40  declare my_cursor cursor for
41    select empno,ename,sal
42      from dept a,emp b
43      where a.deptno = b.deptno and a.dname = in_dname;
44
45  -- 打开游标
46      open my_cursor;
47  -- 通过循环  游标获取值
48      label:loop
49          fetch my_cursor into var_empno,var_ename,var_sal;
50          select var_empno,var_ename,var_sal;
51      end loop label;
52
53  -- 关闭游标
54      close my_cursor;
55  end $$
56
57  delmiter ;
58  call proc22_cursor('销售部');
```

MySQL存储过程传参-异常处理 handler句柄

◆ 异常处理-HANDLER句柄

MySql存储过程也提供了对异常处理的功能：通过定义HANDLER来完成异常声明的实现.
官方文档：https://dev.mysql.com/doc/refman/5.7/en/declare-handler.html

➢ 格式

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_action: {
    CONTINUE
  | EXIT
  | UNDO
}

condition_value: {
    mysql_error_code
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
```

```
1  -- handler句柄
2  delimiter $$
3  create procedure proc23_handle(in in_dname varchar(50))
4  begin
5
6      declare var_empno int;
7      declare var_ename varchar(50);
8      declare var_sal decimal(7,2);
9
10 -- 定义标记值
11 declare flag int default 1;
12 declare my_cursor cursor for
13    select empno,ename,sal
14        from dept a,emp b
15          where a.deptno =b.deptno and a.dname = in_dname;
16
17 -- 定义句柄 定义异常的处理方式
18 /*
19 1异常处理完之后该怎么执行
20  continue 继续执行剩余代码
21  exit 直接终止程序
22  undo 不支持
23
24 2触发条件
25  条件码
26  1329
27
28 3已成触发后执行什么代码
29  设置flag的值
30 */
31      declare continue handler for 1329 set flag= 0;
32      open my_cursor;
33
34      label:loop
35        fetch my_cursor into var_empno,var_ename,var_sal;
```

```
      -- 判断flag 如果flag 的值为1 则执行 否则不执行
          if flag =1 then
           select var_empno, var_ename,var_sal;
          else
           leave label;
          end if;
      end loop label;

   close my_cursor;
end $$
delimiter ;

call proc23_handle('销售部');
```

MySQL的存储过程-练习

```
PREPARE stmt_name FROM preparable_stmt
EXECUTE stmt_name [USING @var_name [, @var_name] ...]
{DEALLOCATE | DROP} PREPARE stmt_name
-- 知识点 时间的处理
-- EXTRACT(unit FROM date)截取时间的指定位置值
-- DATE_ADD(date,INTERVAL expr unit)  日期运算
-- LAST_DAY(date)  获取日期的最后一天
-- YEAR(date) 返回日期中的年
-- MONTH(date) 返回日期的月
-- DAYOFMONTH(date) 返回日
```

```
-- 练习
/*
创建下个月的每天对应的表user_2022_02_01、user_2022_02_02、...

需求描述：
我们需要用某个表记录很多数据，比如记录某某用户的搜索、购买行为(注意，此处是假设用数据库保存)，当每天记录较多时，如果把所有数据都记录到一张表中太庞大，需要分表，我们的要求是，每天一张表，存当天的统计数据，就要求提前生产这些表--每月月底创建下一个月每天的表！
*/
-- 思路：循环构建表名 user_2021_11_01 到 user_2020_11_30；并执行create语句。

-- 预备知识：
-- 查询当前时间日期
SELECT now();
-- 获取下个月的日期
SELECT date_add(now(), INTERVAL 1 MONTH);
-- 获取下个月的年份
SELECT YEAR(date_add(now(), INTERVAL 1 MONTH));
-- 获取下个月的月份
SELECT MONTH (date_add( now(), INTERVAL 1 MONTH));
-- 获取下个月的最后一天
SELECT DAYOFMONTH(LAST_DAY(date_add( now(), INTERVAL 1 MONTH)));


drop database if exists mydb_proc_demo;
create database mydb_proc_demo;
use mydb_proc_demo;
drop procedure if exists proc21_auto_create_tables_next_month;
delimiter $$
```

```
create procedure proc21_auto_create_tables_next_month()
begin
    declare next_year int; -- 下一个月的年
    declare next_month int; -- 下一个月
    declare last_day int; -- 下一个月的最后一天
    declare next_date DATE; -- 下个月的日期
    declare next_month_str char(2); -- 月份小于10在前面填充一位0
    declare next_day_str char(2); -- 天数小于10在前面填充一位0
    declare table_name varchar(20);
    declare i int default 1;

    SET next_date = DATE_ADD(NOW(),INTERVAL 1 MONTH);
    set next_year = year(next_date);
    set next_month = month(next_date);
    set last_day = DAYOFMONTH(LAST_DAY(next_date));
    if next_month < 10 then
        set next_month_str = concat('0', next_month);
    end if;
    -- select next_year, next_month, next_day;
    -- 循环拼接表名，并且建表
    while i <= last_day do
        if i < 10 then
            set next_day_str = concat('0', i);
        else
          set next_day_str = concat('', i);
        end if;
        set table_name = concat_ws('_', next_year, next_month_str,
next_day_str);
        -- select table_name;
        -- 拼接建表sql语句
        set @create_table_sql = concat('create table user_', table_name,
'(`uid` int primary key, `username` varchar(20), `password` varchar(20))');
        -- FROM后面不能使用局部变量
        prepare create_table_stmt from @create_table_sql;-- 预编译sql
        execute create_table_stmt;  -- 执行sql
        deallocate prepare create_table_stmt;
        set i = i + 1;
    end while;

end $$
delimiter ;

call proc21_auto_create_tables_next_month();
```

MySQL存储函数

格式

在MySQL中，创建存储函数使用create function关键字，其基本形式如下：

```
create function func_name ([param_name type[,...]])
returns type
[characteristic ...]
begin
    routine_body
end;
```

参数说明：
（1）func_name：存储函数的名称。
（2）param_name type：可选项，指定存储函数的参数。type参数用于指定存储函数的参数类型，该类型可以是MySQL数据库中所有支持的类型。
（3）RETURNS type：指定返回值的类型。
（4）characteristic：可选项，指定存储函数的特性。
（5）routine_body：SQL代码内容。

```
use database1;

-- 创建存储函数
-- 允许创建函数权限信任
set global log_bin_trust_function_creators =TRUE;
delimiter $$
create function myfunc1_emp() returns int
begin
-- 定义局部变量
  declare cnt int default 0;
  select count(*) into cnt from emp;
  return cnt;
end $$
delimiter ;

-- 调用存储函数
select myfunc1_emp();

-- 创建存储过程 有输入参数
-- 传入一个员工的编号返回员工的名字
delimiter $$
create function myfunc2_emp(in_empno int) returns varchar(50)
begin
    declare out_ename varchar(50);
    select ename into out_ename from emp where empno = in_empno;
    return out_ename;
end $$
delimiter ;

-- 调用存储函数
select myfunc2_emp(1000);
```

MySQL的触发器

◆ 操作-创建触发器

➢ 格式

1、创建只有一个执行语句的触发器

```
create trigger 触发器名 before|after 触发事件
on 表名 for each row
执行语句;
```

2、创建有多个执行语句的触发器

```
create trigger 触发器名before|after  触发事件
on 表名for each row
begin
     执行语句列表
end;
```

```
1  -- 用户表
2  create table user(
3  uid int primary key,
4  username varchar(50) not null,
5  password varchar(50) not null
6  );
7
8  -- 用户信息操作日志表
9  create table user_logs(
10 id int primary key auto_increment,
11 time timestamp,
12 log_text varchar(255)
13 );
14
15 -- 需求1 当user表添加一行数据 则会自动在user_log添加日志记录
16 -- 定义触发器trigger_test1
17 create trigger trigger_test1 after insert
18 on user for each row
19 insert into user_logs values(NULL,now(),'有新用户添加');
20
21 -- 在user表添加数据 让触发器自动执行
22 insert into user values(1,'张三','123456');
23
24
25 -- 当user表数据被修改时 则会自动在user_log添加日志记录
26 delimiter $$
27 create trigger trigger_test2 before update
28 on user for each row
29 begin
30     insert into user_logs values(NULL,now(),'有用户修改数据');
31 end $$
32
33 delimiter ;
34
35 -- 在user表中修改数据 让触发器自动执行
36 update user set password = '888888' where uid = 1;
```

new和old

```
-- new和old
-- insert类型的触发器
-- new
create trigger trigger_test3 after insert
on user for each row
insert into user_logs values(NULL,now(),concat('有新用户添加,信息为:',new.uid,new.username,new.password));

insert into user values(4,'赵六','123456');


-- update类型的触发器
-- new
create trigger trigger_test6 after update
on user for each row
insert into user_logs values(NULL,now(),concat_ws(',','有用户修改信息,信息为:',new.uid,new.username,new.password));

update user set password = '123456' where uid = 1;

-- old
create trigger trigger_test4 after update
on user for each row
insert into user_logs values(NULL,now(),concat('有用户信息修改,修改之前为:',old.uid,old.username,old.password));

update user set password = '888888' where uid = 4;

-- delete
-- old
create trigger trigger_test7 after delete
on user for each row
insert into user_logs values(NULL,now(),concat(',','有用户被删除,被删除用户信息为:',old.uid,old.username,old.password));

delete from user where uid = 1;
```

其他操作

```
-- 查看触发器
show triggers;

-- 删除触发器
drop trigger if exists trigger_test1;
```

MySQL事务

MySQL的事务操作主要有以下三种：

1、开启事务：Start Transaction

- 任何一条DML语句(insert、update、delete)执行，标志事务的开启
- 命令：BEGIN 或 START TRANSACTION

2、提交事务：Commit Transaction

- 成功的结束，将所有的DML语句操作历史记录和底层硬盘数据来一次同步
- 命令：COMMIT

3、回滚事务：Rollback Transaction

- 失败的结束，将所有的DML语句操作历史记录全部清空
- 命令：ROLLBACK

```
create table accout(
id int primary key,
name varchar(20),
money double
);

insert into accout values(1,'zhangsan',1000);
insert into accout values(2,'lisi',1000);

-- 设置Mysql的事务为手动提交 关闭自动提交
select @@autocommit;
set autocommit=0;

-- 模拟账户转账
-- 开启事务
begin;

-- id为1的账户转账给id为2的账户
update accout set money = money - 200 where id =1;
update accout set money = money + 200 where id =2;

-- 提交事务
commit;

-- 回滚事务
rollback;
```

事务特性

事务的隔离级别

◆ 事务的隔离级别

- **读未提交(Read uncommitted)**
  一个事务可以读取另一个未提交事务的数据，最低级别，任何情况都无法保证,会造成脏读。

- **读已提交(Read committed)**
  一个事务要等另一个事务提交后才能读取数据，可避免脏读的发生，会造成不可重复读。

- **可重复读(Repeatable read)**
  就是在开始读取数据（事务开启）时，不再允许修改操作，可避免脏读、不可重复读的发生，但是会造成幻读。

- **串行(Serializable)**
  是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。

Mysql的默认隔离级别是Repeatable read。

事务隔离级别	脏读	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
不可重复读（read-committed）	否	是	是
可重复读（repeatable-read）	否	否	是
串行化（serializable）	否	否	否

Read Uncommitted：在该隔离级别下，一个事务可以读取其他未提交事务的脏数据。脏数据是指其他事务未提交的修改，这些修改可能尚未被数据库系统的日志系统记录下来。因此，这个隔离级别可能会导致脏读，即在一个事务中读取到了其他事务的未提交数据。

Read Committed：在该隔离级别下，一个事务只能读取其他事务已经提交的数据。因此，这个隔离级别可以防止脏读，但可能会导致不可重复读和幻读。不可重复读是指在一个事务中读取到了其他事务已经提交的数据，但在另一个事务中修改了这些数据，导致在第一个事务中再次读取这些数据时，得到的结果与第一次不同。幻读是指在一个事务中读取到了其他事务已经提交的行数，但在另一个事务中插入了新的行，导致在第一个事务中再次读取行数时，得到的结果比第一次多。

Repeatable Read：在该隔离级别下，一个事务在多次读取相同的数据时，得到的结果应该是一样的。因此，这个隔离级别可以防止不可重复读，但可能会导致幻读。例如，如果一个事务在一个子事务中插入了一行数据，但在主事务中插入了另一行数据，那么在主事务中再次读取行数时，得到的结果可能比第一次多。

Serializable：在该隔离级别下，一个事务的执行顺序与其他所有事务的执行顺序完全相同。因此，这个隔离级别可以防止幻读和不可重复读，但可能会导致性能问题。这是因为每个事务都需要在执行之前检查其他所有事务的状态，以确保它们不会对当前事务产生影响。

```
-- 事务的隔离级别-操作
-- 查看隔离级别
show variables like '%isolation%';

-- 设置read uncommitted
set session transaction isolation level read uncommitted;
-- 这种隔离级别会引起脏读,A事务读取到B事务没有提交的数据

-- 设置read committed
set session transaction isolation level read committed;
-- 这种隔离级别会引起不可重复读,A事务在没有提交事务之前,可看到数据不一致

-- 设置repeatable read(MySQL默认)
set session transaction isolation level repeatable read;
-- 这种隔离界别会引起幻读,A事务在提交之前和提交之后看到的数据不一致

-- 设置serializable
set session transaction isolation level serializable;
-- 这种隔离级别比较安全,但是效率低,A事务操作表时,表会被锁起,B事务不能操作

mysql> show variables like '%isolation%';
+-----------------------+-----------------+
| Variable_name         | Value           |
+-----------------------+-----------------+
| transaction_isolation | REPEATABLE-READ |
+-----------------------+-----------------+
1 row in set, 1 warning (0.00 sec)
```

MySQL的锁机制

➢ 从对数据操作的粒度分：

1） 表锁：操作时，会锁定整个表。

2） 行锁：操作时，会锁定当前操作行。

➢ 从对数据操作的类型分：

1） 读锁（共享锁）：针对同一份数据，多个读操作可以同时进行而不会互相影响。

2） 写锁（排它锁）：当前操作没有完成之前，它会阻断其他写锁和读锁。

存储引擎	表级锁	行级锁
MyISAM	支持	不支持
InnoDB	支持	支持
MEMORY	支持	不支持
BDB	支持	不支持

锁类型	特点
表级锁	偏向MyISAM 存储引擎，开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高,并发度最低。
行级锁	偏向InnoDB 存储引擎，开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低,并发度也最高。

```
create table `tb_book`(
`id` int(11) auto_increment,
`name` varchar(50) default null,
`publish_time` date default null,
`status` char(1) default null,
primary key (`id`)
)engine =myisam default charset=utf8;

insert into tb_book (id,name,publish_time,status) values(null,'java编程思想','2088-08-01','1');
insert into tb_book (id,name,publish_time,status) values(null,'solr编程思想','2088-08-08','0');

create table `tb_user`(
`id` int(11) auto_increment,
`name` varchar(50) default null,
primary key(`id`)
)engine = myisam default charset = utf8;

insert into tb_user (id,name) values (null,'令狐冲');
insert into tb_user (id,name) values (null,'田伯光');

-- 读锁
mysql> lock table tb_book read;
mysql> update tb_book set status='2' where id = '1';
ERROR 1099 (HY000): Table 'tb_book' was locked with a READ lock and can't be updated

mysql> select * from tb_user;
ERROR 1100 (HY000): Table 'tb_user' was not locked with LOCK TABLES
```

```
-- 解锁
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from tb_user;
+----+-----------+
| id | name      |
+----+-----------+
|  1 | 令狐冲    |
|  2 | 田伯光    |
+----+-----------+
2 rows in set (0.00 sec)
-----------------------------------------------------------------
-- 写锁
mysql> lock table tb_book write;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from tb_book;
+----+-----------------+--------------+--------+
| id | name             | publish_time | status |
+----+-----------------+--------------+--------+
|  1 | java编程思想     | 2088-08-01   | 1      |
|  2 | solr编程思想     | 2088-08-08   | 0      |
+----+-----------------+--------------+--------+
2 rows in set (0.00 sec)

mysql> update tb_book set status = '1' where id = 1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)
-----------------------------------------------------------------

create table test_innodb_lock(
id int(11),
name varchar(16),
sex varchar(1)
)engine = innodb;

insert into test_innodb_lock values(1,'100','1');
insert into test_innodb_lock values(3,'3','1');
insert into test_innodb_lock values(4,'400','0');
insert into test_innodb_lock values(5,'500','1');
insert into test_innodb_lock values(6,'600','0');
insert into test_innodb_lock values(7,'700','0');
insert into test_innodb_lock values(8,'800','1');
insert into test_innodb_lock values(9,'900','1');
insert into test_innodb_lock values(1,'200','0');

-- 行锁

select * from test_innodb_lock;

update test_innodb_lock set sex = '2' where id = 1;
```

```
84
85 select * from test_innodb_lock where id = 1;
```

MySQL日志

错误日志

```
1  -- 默认的日志文件名为 hostname.err 默认存放目录为mysql的数据目录
2  -- Yingyong\mysql-8.1.0-winx64\data\.err
3  mysql> show variables like 'log_error%';
4  +----------------------------+-------------------------------------------------
   ----+
5  | Variable_name              | Value
       |
6  +----------------------------+-------------------------------------------------
   ----+
7  | log_error                  | D:\Yingyong\mysql-8.1.0-
   winx64\data\Walson171.err |
8  | log_error_services         | log_filter_internal; log_sink_internal
       |
9  | log_error_suppression_list |
       |
10 | log_error_verbosity        | 2
       |
11 +----------------------------+-------------------------------------------------
   ----+
```

二进制日志

```
1  -- 默认格式
2  Windows系统：my.ini  Linux系统：my.cnf
3
4  #配置开启binlog日志,日志的文件前缀为mysqlbin ---->生成的文件名
   如:mysqlbin.000001,mysqlbin.000002
5  -- Yingyong\mysql-8.1.0-winx64\my.ini
6  log_bin = mysqlbin
7
8  #配置二进制日志的格式
9  binlog_format=STATMENT
```

查询日志

```
1  -- 查看Mysql是否开启了binlog日志
2  mysql> show variables like 'log_bin';
3  +---------------+-------+
4  | Variable_name | Value |
5  +---------------+-------+
6  | log_bin       | ON    |
7  +---------------+-------+
8
9  -- 查看binlog日志的格式
10 -- \Yingyong\mysql-8.1.0-winx64\data\binlog.000036
11 show variables like 'binlog_format';
```

```
mysql> show variables like 'binlog_format';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+

-- 查看所有日志
show binlog events;
mysql> show binlog events;
+---------------+-----+---------------+-----------+-------------+-----------------------+
| Log_name      | Pos | Event_type    | Server_id | End_log_pos | Info                  |
+---------------+-----+---------------+-----------+-------------+-----------------------+
| binlog.000036 |   4 | Format_desc   |         1 |         126 | Server ver: 8.1.0, Binlog ver: 4 |
| binlog.000036 | 126 | Previous_gtids |         1 |         157 |                       |
| binlog.000036 | 157 | Stop          |         1 |         180 |                       |
+---------------+-----+---------------+-----------+-------------+-----------------------+

-- 查看最新的日志
show master status;
mysql> show master status;
+---------------+----------+--------------+------------------+-------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+---------------+----------+--------------+------------------+-------------------+
| binlog.000092 |    12668 |              |                  |                   |
+---------------+----------+--------------+------------------+-------------------+

-- 查询指定的binlog日志  只有增删改才会进行日志操作
show binlog events in 'binlog.000036';
mysql> show binlog events in 'binlog.000036';
+---------------+-----+---------------+-----------+-------------+-----------------------+
| Log_name      | Pos | Event_type    | Server_id | End_log_pos | Info                  |
+---------------+-----+---------------+-----------+-------------+-----------------------+
| binlog.000036 |   4 | Format_desc   |         1 |         126 | Server ver: 8.1.0, Binlog ver: 4 |
| binlog.000036 | 126 | Previous_gtids |         1 |         157 |                       |
| binlog.000036 | 157 | Stop          |         1 |         180 |                       |
+---------------+-----+---------------+-----------+-------------+-----------------------+
```

```
-- 从指定的位置开始 查看指定的binlog
mysql> show binlog events in 'binlog.000036' from 46;
+---------------+-----+----------------+-----------+-------------+------+
| Log_name      | Pos | Event_type     | Server_id | End_log_pos | Info |
+---------------+-----+----------------+-----------+-------------+------+
| binlog.000036 | 126 | Previous_gtids |         1 |         157 |      |
| binlog.000036 | 157 | Stop           |         1 |         180 |      |
+---------------+-----+----------------+-----------+-------------+------+

-- 从指定的位置开始 查看指定的binlog 限制查询的条数
mysql> show binlog events in 'binlog.000036' from 10 limit 2;
+---------------+-----+----------------+-----------+-------------+------+
| Log_name      | Pos | Event_type     | Server_id | End_log_pos | Info |
+---------------+-----+----------------+-----------+-------------+------+
| binlog.000036 | 126 | Previous_gtids |         1 |         157 |      |
| binlog.000036 | 157 | Stop           |         1 |         180 |      |
+---------------+-----+----------------+-----------+-------------+------+

-- 从指定的位置开始 带有偏移 查看指定的binlog 限制查询的条数
mysql> show binlog events in 'binlog.000036' from 10 limit 1,2;
+---------------+-----+------------+-----------+-------------+------+
| Log_name      | Pos | Event_type | Server_id | End_log_pos | Info |
+---------------+-----+------------+-----------+-------------+------+
| binlog.000036 | 157 | Stop       |         1 |         180 |      |
+---------------+-----+------------+-----------+-------------+------+

-- 清空所有的binlog日志文件
reset master;

-- 查看MySQL是否开启了查询日志
mysql> show variables like 'general_log';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| general_log   | OFF   |
+---------------+-------+

-- 开启查询日志 临时的
-- \Yingyong\mysql-8.1.0-winx64\data\.log
mysql> set global general_log=1;
Query OK, 0 rows affected (0.01 sec)
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| general_log   | ON    |
+---------------+-------+
1 row in set, 1 warning (0.00 sec)

TCP Port: 3306, Named Pipe: MySQL
Time                 Id Command    Argument
2023-10-24T15:39:52.467496Z    21 Query show variables like 'general_log'
2023-10-24T15:41:00.863899Z     8 Query SET PROFILING = 1
2023-10-24T15:41:00.864296Z     8 Query SHOW STATUS
2023-10-24T15:41:00.867358Z     8 Query SHOW STATUS

```

```
-- 慢查询日志
-- 查看慢查询日志是否开启
mysql> show variables like 'slow_query_log%';
+---------------------+-------------------------------------------------------
---+
| Variable_name       | Value
   |
+---------------------+-------------------------------------------------------
---+
| slow_query_log      | OFF
   |
| slow_query_log_file | \Yingyong\mysql-8.1.0-winx64\data\-slow.log |
+---------------------+-------------------------------------------------------
---+

-- 开启慢查询日志
mysql> set global slow_query_log = 1;
Query OK, 0 rows affected (0.01 sec)
+---------------------+-------------------------------------------------------
---+
| Variable_name       | Value
   |
+---------------------+-------------------------------------------------------
---+
| slow_query_log      | ON
  |
| slow_query_log_file | \Yingyong\mysql-8.1.0-winx64\data\-slow.log |
+---------------------+-------------------------------------------------------
---+

-- 查看慢查询的超时时间
mysql> show variables like 'long_query_time%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
TCP Port: 3306, Named Pipe: MySQL
Time                 Id Command    Argument
# Query_time: 11.021823  Lock_time: 0.000000 Rows_sent: 1  Rows_examined: 1
SET timestamp=1698162671;
select sleep(11);
```

MySQL的优化

参数	含义
Com_select	执行 select 操作的次数，一次查询只累加 1。
Com_insert	执行 INSERT 操作的次数，对于批量插入的 INSERT 操作，只累加一次。
Com_update	执行 UPDATE 操作的次数。
Com_delete	执行 DELETE 操作的次数。
Innodb_rows_read	select 查询返回的行数。
Innodb_rows_inserted	执行 INSERT 操作插入的行数。
Innodb_rows_updated	执行 UPDATE 操作更新的行数。
Innodb_rows_deleted	执行 DELETE 操作删除的行数。
Connections	试图连接 MySQL 服务器的次数。
Uptime	服务器工作时间。
Slow_queries	慢查询的次数。

```
insert into accout values(3,'wangwu',1000);

-- 查看当前会话SQL执行类型的统计信息
show session status like 'Com_____';

-- 查看全局(自从上次MySQL服务器启动至今)执行类型的统计信息
show global status like 'Com_____';

-- 查看针对InnoDB引擎的统计信息
mysql> show status like 'Innodb_row_%';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Innodb_row_lock_current_waits | 0     |
| Innodb_row_lock_time          | 0     |
| Innodb_row_lock_time_avg      | 0     |
| Innodb_row_lock_time_max      | 0     |
| Innodb_row_lock_waits         | 0     |
| Innodb_rows_deleted           | 0     |
| Innodb_rows_inserted          | 1     |
| Innodb_rows_read              | 0     |
| Innodb_rows_updated           | 0     |
+-------------------------------+-------+

-- 查看慢日志记录SQL的最低阈值时间 默认如果SQL的执行时间>=10秒 则算慢查询 则会将该操作记录到慢日志中去
mysql> show variables like '%long_query_time%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set, 1 warning (0.00 sec)

-- 修改慢日志记录SQL的最低阈值时间
set global long_query_time = 5;

```

```
-- 通过show processlist查看当前客户端连接服务器的线程执行状态信息
mysql> show processlist;
+----+-----------------+-----------------+----------+---------+-------+------------------------+------------------+
| Id | User            | Host            | db       | Command | Time  | State                  | Info             |
+----+-----------------+-----------------+----------+---------+-------+------------------------+------------------+
|  5 | event_scheduler | localhost       | NULL     | Daemon  | 31268 | Waiting on empty queue | NULL             |
|  8 | root            | localhost:52288 | database1 | Sleep   |    19 |                        | NULL             |
|  9 | root            | localhost:52381 | database1 | Sleep   | 10855 |                        | NULL             |
| 10 | root            | localhost:52382 | NULL     | Sleep   | 11416 |                        | NULL             |
| 11 | root            | localhost:52545 | database1 | Query   |     0 | init                   | show processlist |
+----+-----------------+-----------------+----------+---------+-------+------------------------+------------------+
```

Explain分析执行计划

字段	含义
id	select查询的序列号，是一组数字，表示的是查询中执行select子句或者是操作表的顺序。
select_type	表示 SELECT 的类型，常见的取值有 SIMPLE（简单表，即不使用表连接或者子查询）、PRIMARY（主查询，即外层的查询）、UNION（UNION 中的第二个或者后面的查询语句）、SUBQUERY（子查询中的第一个 SELECT）等
table	输出结果集的表
type	表示表的连接类型，性能由好到差的连接类型为(system ---> const -----> eq_ref ------> ref -------> ref_or_null---> index_merge ---> index_subquery -----> range -----> index ------> all)
possible_keys	表示查询时，可能使用的索引
key	表示实际使用的索引
key_len	索引字段的长度
rows	扫描行的数量
extra	执行情况的说明和描述

```
-- 查询执行计划
explain select * from emp where empno = "1001";
mysql> explain select * from emp where empno = "1001";
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | emp   | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
```

Explain分析执行计划之select_type

select_type	含义
SIMPLE	简单的select查询，查询中不包含子查询或者UNION
PRIMARY	查询中若包含任何复杂的子查询，最外层查询标记为该标识
SUBQUERY	在SELECT 或 WHERE 列表中包含了子查询
DERIVED	在FROM 列表中包含的子查询，被标记为 DERIVED（衍生） MYSQL会递归执行这些子查询，把结果放在临时表中
UNION	若第二个SELECT出现在UNION之后，则标记为UNION； 若UNION包含在FROM子句的子查询中，外层SELECT将被标记为： DERIVED
UNION RESULT	从UNION表获取结果的SELECT

```
-- Explain的select_type
mysql> explain select * from emp;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | emp   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   14 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
```

Explain分析执行计划-Explain之type

type	含义
NULL	MySQL不访问任何表，索引，直接返回结果
system	系统表，少量数据，往往不需要进行磁盘IO；如果是5.7及以上版本的话就不是system了，而是all，即使只有一条记录
const	命中主键(primary key)或者唯一(unique)索引；被连接的部分是一个常量(const)值；
eq_ref	对于前表的每一行，后表只有一行被扫描。（1）join查询；（2）命中主键(primary key)或者非空唯一(unique not null)索引；（3）等值连接；
ref	非唯一性索引扫描，返回匹配某个单独值的所有行。对于前表的每一行(row)，后表可能有多于一行的数据被扫描。
range	只检索给定返回的行，使用一个索引来选择行。 where 之后出现 between， <, >, in 等操作。
index	需要扫描索引上的全部数据。
all	全表扫描，此时id上无索引

```
结果值从最好到最坏以此是 system>const>eq_ref>ref>range>index>ALL
mysql> explain select now();
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+
|  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL    | NULL | NULL |     NULL | No tables used |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+----------------+

```

```
-- system 查询系统表 表示直接从内存读取数据 不会从磁盘读取 但是5.7及以上版本不再显示system直接显示ALL
mysql> explain select * from mysql.tables_priv;
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table       | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | tables_priv | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    2 |   100.00 | NULL  |
+----+-------------+-------------+------------+------+---------------+------+---------+------+------+----------+-------+

mysql> explain select * from emp where ename = '甘宁';
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | emp   | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   14 |    10.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+

create unique index index_uname on emp(ename);
mysql> explain select * from emp where ename = '甘宁';
+----+-------------+-------+------------+-------+---------------+-------------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type  | possible_keys | key         | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+-------+---------------+-------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | emp   | NULL       | const | index_uname   | index_uname | 63      | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+-------+---------------+-------------+---------+-------+------+----------+-------+


create unique index index_uname on emp(ename);

drop index index_ename on emp;

create index index_ename on emp(ename);  -- 添加普通索引

-- eq_ref
create table user2(
id int,
name varchar(20)
);

insert into user2 values(1,'张三'),(2,'李四'),(3,'王五');

create table user2_ex(
id int,
```

```
age int
);
insert into user2_ex values(1,20),(2,21),(3,22);

--
explain select * from user2 a,user_ex b where a.id = b.id;

-- 给user2表添加主键索引
alter table user2 add primary key(id);

explain select * from user2 a,user2_ex b where a.id = b.id;

-- 在user_ex 表添加一个重复的行数据  ALL
mysql> explain select * from user2 a,user2_ex b where a.id = b.id;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | a     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    3 |   100.00 | NULL        |
|  1 | SIMPLE      | b     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    4 |    25.00 | Using where; Using join buffer (hash join) |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------------+

-- ref 左表有普通索引 和右表配置时可能会匹配多行
-- 删除user2表的主键索引
alter table user2 drop primary key;
-- 给user2表添加普通索引
create index index_id on user2(id);  -- 添加普通索引

-- range范围查询
mysql> explain select * from user2 where id >2;
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+
| id | select_type | table | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered | Extra                 |
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+
|  1 | SIMPLE      | user2 | NULL       | range | index_id      | index_id | 5       | NULL |    1 |   100.00 | Using index condition |
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-----------------------+


mysql> explain select id from user2;
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+----------+---------+------+------+----------+-------------+
```

```
|  1 | SIMPLE      | user2 | NULL       | index | NULL          | index_id |
5       | NULL |    3 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+----------+-
--------+------+------+----------+-------------+
```

Explain分析执行计划-其他字段指标

◆ Explain分析执行计划-其他指标字段

➢ Explain 之 table

显示这一步所访问数据库中表名称有时不是真实的表名字，可能是简称，

➢ explain 之 rows

扫描行的数量。

➢ Explain 之 key

possible_keys : 显示可能应用在这张表的索引， 一个或多个。

key： 实际使用的索引， 如果为NULL， 则没有使用索引。

key_len : 表示索引中使用的字节数， 该值为索引字段最大可能长度，并非实际使用长度，在不损失精确性的前提下， 长度越短越好 。

extra	含义
using filesort	说明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取， 称为 “文件排序”, 效率低。
using temporary	需要建立临时表(temporary table)来暂存中间结果，常见于 order by 和 group by； 效率低
using index	SQL所需要返回的所有列数据均在一棵索引树上， 避免访问表的数据行， 效率不错。

```
-- Explain其他字段
mysql> explain select * from user2 where id = 1;
+----+-------------+-------+------------+------+---------------+----------+---
------+-------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key      |
key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+----------+---
------+-------+------+----------+-------+
|  1 | SIMPLE      | user2 | NULL       | ref  | index_id      | index_id | 5
      | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+----------+---
------+-------+------+----------+-------+
```

MySQL 的优化-show file分析

字段	含义
Status	sql 语句执行的状态
Duration	sql 执行过程中每一个步骤的耗时
CPU_user	当前用户占有的cpu
CPU_system	系统占有的cpu

```sql
-- 执行SQL
show databases;
use database1;
show tables;
select count(*) from user2;
select * from user2;

show profiles;

show profile for query 271;

show profile cpu for query 271;
```

MySQL 的优化-trace分析优化器执行计划

```sql
-- trace分析优化器执行
use information_schema;

set optimizer_trace="enable=on",end_markers_in_json=on;
set optimizer_trace_max_mem_size=100000;

select * from user2 where id > 1;

select * from information_schema.optimizer_trace \G;
```

MySQL 的优化-索引优化

```sql
create table tb_seller(
sellerid varchar(100),
name varchar(100),
nickname varchar(50),
password varchar(30),
status varchar(1),
adress varchar(100),
createtime datetime,
primary key(sellerid)
);

insert into tb_seller values
('alibaba','阿里巴巴','阿里小店','e10abc3949ba59zbbe56e057f20f883e','1','北京市','2088-01-01-12:00:00'),
('baidu','百度科技有限公司','百度小店','e10abc3949ba59zbbe56e057f20f883e','1','北京市','2088-01-01-12:00:00'),
('huawei','华为科技有限公司','华为小店','e10abc3949ba59zbbe56e057f20f883e','0','北京市','2088-01-01-12:00:00'),
('itcast','传智播客教育科技有限公司','传智博客','e10abc3949ba59zbbe56e057f20f883e','1','北京市','2088-01-01-12:00:00'),
('itheima','黑马程序员','阿里小电','e10abc3949ba59zbbe56e057f20f883e','0','北京市','2008-01-01-12:00:00'),
('luoji','罗技科技有限公司','罗技小店','e10abc3949ba59zbbe56e057f20f883e','1','北京市','2088-01-01-12:00:00');

-- 创建组合索引
create index idx_seller_name_sta_addr on tb_seller(name,status,adress);
```

```
22
23  -- 全值匹配
24  explain select * from tb_seller where name = '小米科技' and status='1' and
    address = '北京市';
25
26  mysql> explain select * from tb_seller where name = '北京市'  and name = '小
    米科技' and status = '1';
27  +----+-------------+-------+------------+------+---------------+------+-----
    ----+------+------+----------+-----------------+
28  | id | select_type | table | partitions | type | possible_keys | key  |
    key_len | ref  | rows | filtered | Extra           |
29  +----+-------------+-------+------------+------+---------------+------+-----
    ----+------+------+----------+-----------------+
30  |  1 | SIMPLE      | NULL  | NULL       | NULL | NULL          | NULL | NULL
       | NULL | NULL |     NULL | Impossible WHERE |
31  +----+-------------+-------+------------+------+---------------+------+-----
    ----+------+------+----------+-----------------+
32
33
34  -- 最左前缀法则
35  -- 如果索引了多列,要遵守最左前缀法则。指的是查询从索引的最左前列开始,并且不跳过索引中的
    列
36  explain select * from tb_seller where name ='小米科技';  -- 303
37
38  explain select * from tb_seller where name='小米科技' and status='1';  -- 309
39
40  -- 违反最左前缀法则 索引失效
41  explain select * from tb_seller where status ='1';  -- NULL
42
43  -- 如果符合最左法则 但是出现跳跃某一列 只有最左列索引生效
44  explain select * from tb_seller where name = '小米科技' and adress ='北京市';
     -- 303
45
46
47  -- 范围查询右边的列 不能使用索引
48  -- 根据前面的两个字段name status 查询是走索引的 但是最后一个条件address没有用到索引
49  explain select * from tb_seller where name='小米科技' and status > '1' and
    adress = '北京市';  -- 309
50
51  -- 不要再索引列上进行运算操作 索引将失效
52  explain select * from tb_seller where substring(name,3,2)='科技'; -- NULL
53
54  -- 字符串不加单引号,造成索引失效
55  explain select * from tb_seller where name ='小米科技' and status = 1;  --
    303
56
57  -- 尽量使用覆盖索引 避免select *
58  -- 需要从原表及磁盘上读取数据
59  explain select * from tb_seller where name ='小米科技' and adress ='北京市';
60
61  -- 从索引树中就可以查询到所有数据
62  explain select name from tb_seller where name='小米科技' and adress='北京市';
     -- 效率高  Using where; Using index
63
64  -- 尽量使用覆盖索引（只访问索引的查询 索引列完全包含查询列）减少select *
```

```
explain select * from tb_seller where name='小米科技' and adress='北京市';  --
303

/*
using index 使用覆盖索引的时候就会出现
using where 在查找使用索引的情况下 需要回表去查询所需的数据
using index condition 查找使用了索引 但是需要回表查询数据
using index;using where:查找使用了索引 但是需要的数据都在索引列中能找到所以不需要回
表查询
*/

-- 用or分割开的条件 如果or钱的条件中的列有索引 而后面的列中没有索引 那么涉及的索引都不
会被引用
-- using where
explain select * from tb_seller where name ='黑马程序员' or createtime='2088-
01-01 12:00:00';
explain select * from tb_seller where name ='黑马程序员' or adress = '西安市';
explain select * from tb_seller where name ='黑马程序员' or status = '1';

-- 以%开头的Like模糊查询 索引失败
explain select * from tb_seller where name like '科技%'; -- 用索引
explain select * from tb_seller where name like '%科技'; -- 不用索引

-- 优化 不用*使用索引列
explain select name from tb_seller where name like '%科技%';

-- 如果MySQL评估使用索引比全表更慢 则不使用索引
  -- 这种情况是由数据的本身情况决定的
create index index_adress on tb_seller(adress);

explain select * from tb_seller where adress ='北京市';
explain select * from tb_seller where adress ='西安市';

-- is NULL is not NULL 有时有效 有时索引失效
create index index_address on tb_seller(nickname);
explain select * from tb_seller where nickname is null;    -- 索引有效
explain select * from tb_seller where nickname is not null;  -- 无效

-- in 走索引 not in 索引失效
-- 普通索引
explain select * from tb_seller where nickname in('阿里小店','百度小店'); --
使用索引
explain select * from tb_seller where nickname not in('阿里小店','百度小店');
-- 使用索引

-- 主键索引
explain select * from tb_seller where sellerid in('alibaba','baidu'); -- 使
用索引
explain select * from tb_seller where sellerid not in('alibaba','baidu'); --
使用索引

-- 单列索引和复合索引 尽量使用复合索引
create index idx_seller_name_sta_adr on tb_seller(name,status,adress);
/*
name
name + status
```

```
name + status + adress
*/

-- 如果一张表有多个单列索引 即使where中都使用了这些索引列 则只有一个最优索引生效
drop index idx_seller_name_sta_addr on tb_seller;

show index from tb_seller;

create index index_adess on tb_seller(name);
create index index_status on tb_seller(status);
create index index_statu on tb_seller(adress);

explain select * from tb_seller where name ='小米科技' and status = '1' and adress = '西安市';
explain select * from tb_seller where status = '1' and adress = '西安市';
```

MySQL 的优化-大批量数据

```
-- 大批量数据导入
create table tb_user(
id int(11) not null auto_increment,
username varchar(45) not null,
password varchar(96) not null,
name varchar(45) not null,
birthday datetime default null,
sex char(1) default null,
email varchar(45) default null,
phone varchar(45) default null,
qq varchar(32) default null,
status varchar(32) not null comment '用户状态',
create_time datetime not null,
update_time datetime default null,
primary key (id),
unique key unique_user_username (username)
);

-- 首先 检查一个全局系统变量'local_infile'的状态 如果得到如下显示Value=OFF 则说明这是不可用的
show global variables like 'local_infile';

-- 修改local_infile值为on 开启local_infile
set global local_infile=1;

-- 加载数据
-- 当通过load向表加载数据时 尽量保证文件中的主键有序 这样可以提高执行效率
load data local infile 'D:\\sql_data\\sql1.log' into table tb_user fields terminated by ',' lines terminated by '\n';
load data local infile 'D:\\sql_data\\sql2.log' into table tb_user fields terminated by ',' lines terminated by '\n';

-- 关闭唯一性校验
set unique_check=0;
```

MySQL 的优化-insert语句

```
-- insert语句
insert into tb_test(1,'Tom');
insert into tb_test(3,'Cat');
insert into tb_test(3,'Jerry');

-- 优化后的方案
insert into tb_test values(1,'Tom'),(2,'Cat'),(3,'Jerry');

-- 在事务中进行数据插入
begin;
insert into tb_test(1,'Tom');
insert into tb_test(2,'Cat');
insert into tb_test(3,'Jerry');
commit;

-- 有序插入
insert into tb_test(1,'Tom');
insert into tb_test(2,'Cat');
insert into tb_test(3,'Jerry');
```

MySQL 的优化-order by语句

```
-- 创建索引
create index idx_emp_age_salary on emp(deptno,sal);

explain select * from emp order by deptno;  -- Using filesort
explain select * from emp order by deptno,sal;  -- Using filesort

explain select empno from emp order by deptno;  -- Using index

-- order by 后面的多个排序字段要求尽量排序方式相同
-- order by 后面的多个排序字段顺序尽量和组合索引字段顺序一致
explain select empno,deptno from emp order by empno,sal;  -- Using index;
Using filesort

show variables like 'max_length_for_sort_data';  -- 4096
show variables like 'sort_buffer_size';  -- 262144
```

MySQL 的优化-子查询语句

```
-- 多表查询高于子查询
-- system>const>sq_ref>red>range>index>ALL
-- 索引
explain select * from user where uid in (select uid from user_role);

-- 多表
explain select * from user u join user_role ur on u.uid =ur.uid;
```

MySQL 的优化-limit查询

```
select count(*) from emp;  -- 0.020s
select * from emp limit 0,10;  -- 0.021s
select * from emp limit 3,14;  -- 0.013s

select empno from emp order by empno limit 3,14;  -- 0.013s

select * from emp a,(select empno from emp order by empno limit 3,14) b where a.empno =b.empno;  -- 0.012s

select * from emp where empno > 1002 limit 5;  -- 0.013s
```