SQL

基础：选择select；表连接join/left join/union

常用：去重distinct；聚合max/min/sum/count/group by；筛选having/where；排序order by/sort by；条件case when...end；字符串substr/concat/split

进阶：日期函数to_date/dateoff；分组排除row_number()；取百分比percentile

高级：窗口函数（分组排序row_number/rank/dense_rank；分组取最大最小first_value/last_value；累积百分比cume_dist/sum()over；错位lead/lag）；日期函数（时间和日期戳转化unix_timestamp/from_unixtime；日期操作to_date/datesub/date_add/dateoff）；字符串函数（substr,concat,concat_ws,length,upper这类基础函数；regexp_replace正则/get_json_object）；其他常用函数（lateral view explode行列转换；rand等随机抽样）

SQL是一种操作数据库的语言，主要包括CRUD，对于数据分析师来说，一般只要查找数据的，不能对数据库进行增删改

```
SELECT comlumns_name --查找一列或多了之间用逗号隔开
FROM Table           --目标表
WHERE condition      --过滤条件
GROUP BY columns_name --按列值分组，可以1个或多个列
HAVING condition     --分组后的筛选条件，HAVING和WHERE的区别在于前者表达式中可包含函数
ORDER BY columns_name --按列排序
LIMIT start,row_count --对结果进行限定，start表示从哪行开始，row_count表示结果行数
```

一、基础用法

1.简单查询

2.条件查询

有一张学生表student，包括学生id，姓名，年龄，班级，分数字段，下面用SQL来查找数据吧。

stu_id	stu_name	age	class	grade
1	小明	17	1	60
2	小花	18	2	90
3	小王	16	2	70
4	小李	20	1	20
5	小陈	17	2	60
6	小赵	15	1	80

知乎 @呆呆玩数据

student表

```
--查找id.姓名，成绩列
SELECT stu_id,stu_name,grade FROM student;
```

执行结果

信息 结果1 概况 状态

stu_id	stu_name	grade
1	小明	60
2	小花	90
3	小王	70
4	小李	20
5	小陈	60
6	小赵	80

```
--查找成绩大于70分的学生id，姓名，成绩列
SELECT s.stu_id,s.stu_name,s.grade FROM student s WHERE s.grade >=70;
```

执行结果

信息 结果1 概况 状态

stu_id	stu_name	grade
2	小花	90
3	小王	70
6	小赵	80

3.聚合函数

```
--计算班级人数，平均成绩，最高成绩，最低成绩
SELECT s.class,COUNT(s.stu_id) as
count1,AVG(s.grade),MAX(s.grade),MIN(s.grade) FROM student s GROUP BY s.class;
```

执行结果

信息 结果1 概况 状态

class	count1	AVG(s.grade)	MAX(s.grade)	MIN(s.grade)
1	3	53.333333333333336	80	20
2	3	73.33333333333333	90	60

注释：计算班级平均成绩用到了group by语句，同样的class被分为一组。当使用分组语句时，select语句后面只能出现分组字段和聚合函数。

4.分组后筛选

```
--Having语句筛选班级平均分大于60分的班级
SELECT s.class,COUNT(s.stu_id) as count1,AVG(s.grade),MAX(s.grade),MIN(s.grade) FROM student s GROUP BY s.class HAVING AVG(s.grade)>60;
```

执行结果

信息 结果1 概况 状态

class	count1	AVG(s.grade)	MAX(s.grade)	MIN(s.grade)
2	3	73.33333333333333	90	60

5.模糊查询

```
--like进行模糊匹配
where name like '陈%'     --找出姓陈的人
where name like '%铭%'     --找出名字中有铭字的人
where name like '陈_'     --找出姓陈且名字为两个字的人
```

注释：like进行模糊匹配时，%可以匹配任意多个任意字符，_匹配任意一个字符。

二、表连接

表连接是指两张表可以通过想用的关键字段进行连接，包括内连接和外连接。外连接包括左外连接、右外连接和全外连接。

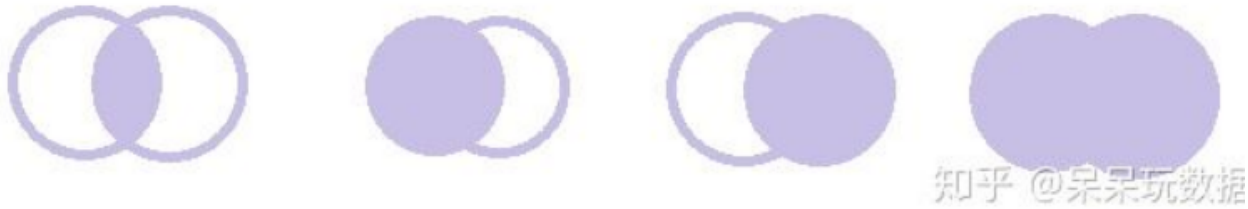
1.内连接指结果表只包含同时存在两张表中的连接字段

2.左外连接是指既包含内连接的连接字段还包含左表未连接的字段

3.右外连接是指既包含内连接的连接字段还包含右表未连接的字段

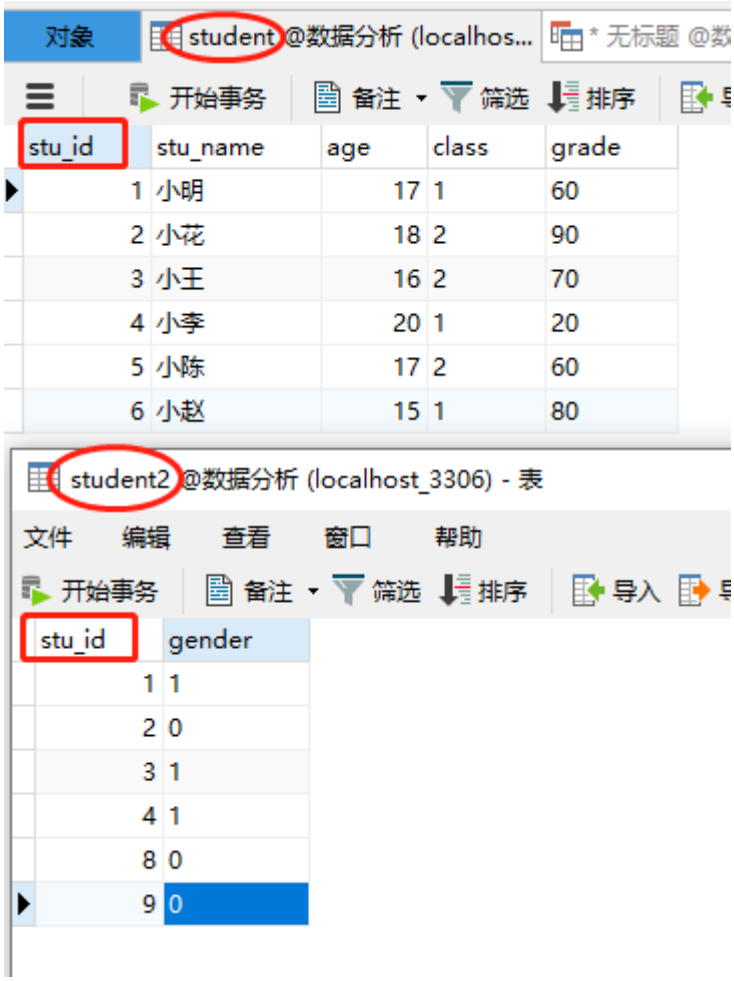
4.全外连接是指包含两张表的所有连接字段.

韦恩图



从左到右，依次是内连接，左外连接，右外连接，全外连接

表连接，两张表student，student2，stu_id为连接字段



1.内连接

```
-- 内连接，*是通配符，取出表中所有变量
SELECT s1.*,s2.* FROM student s1 INNER JOIN student2 s2 ON s1.stu_id=s2.stu_id;
```

执行结果

信息 结果1 概况 状态

stu_id	stu_name	age	class	grade	stu_id1	gender
1	小明	17	1	60	1	1
2	小花	18	2	90	2	0
3	小王	16	2	70	3	1
4	小李	20	1	20	4	1

2.左外连接

```
-- 左外连接
SELECT s1.*,s2.* FROM student s1 LEFT JOIN student2 s2 ON s1.stu_id=s2.stu_id;
```

执行结果

信息 结果1 概况 状态

stu_id	stu_name	age	class	grade	stu_id1	gender
1	小明	17	1	60	1	1
2	小花	18	2	90	2	0
3	小王	16	2	70	3	1
4	小李	20	1	20	4	1
5	小陈	17	2	60	(Null)	(Null)
6	小赵	15	1	80	(Null)	(Null)

3.右外连接

```
-- 右外连接
SELECT s1.*,s2.* FROM student s1 RIGHT JOIN student2 s2 ON
s1.stu_id=s2.stu_id;
```

执行结果

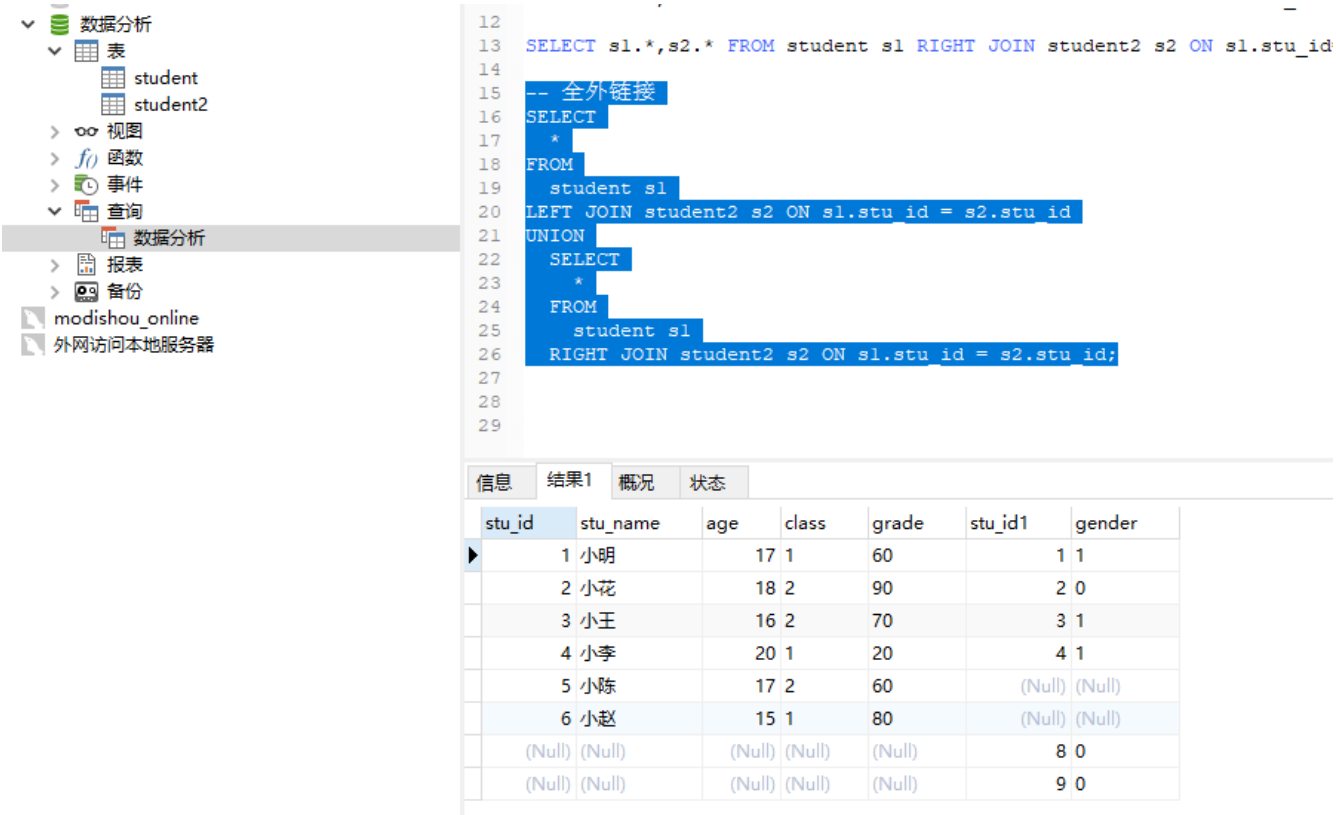
信息 结果1 概况 状态

stu_id	stu_name	age	class	grade	stu_id1	gender
1	小明	17	1	60	1	1
2	小花	18	2	90	2	0
3	小王	16	2	70	3	1
4	小李	20	1	20	4	1
(Null)	(Null)	(Null)	(Null)	(Null)	8	0
(Null)	(Null)	(Null)	(Null)	(Null)	9	0

4.全外连接

```sql
-- 全外连接
SELECT
    *
FROM
    student s1
LEFT JOIN student2 s2 ON s1.stu_id = s2.stu_id
UNION
    SELECT
        *
    FROM
        student s1
    RIGHT JOIN student2 s2 ON s1.stu_id = s2.stu_id;
```

执行结果



三、子查询

子查询是指把一个查询的结果放到另一个查询里面使用，功能十分丰富，既可以放在from语句中作为临时表供另一个查询使用，也可以放在where语句子句后面进行过滤。

放在where子句后面的子查询返回结果可以是单行，也可以是多行。

单行子查询一般与>,<,=,<=,>=等比较符一起使用。

而多行子查询与in(等于子查询返回结果的任意一个)，any(比较符满足子查询中的任意一个情况)，all(比较符满足子查询中所有情况)等配套使用

1.where单行子查询

student表中查询grade大于平局分的学生id，姓名，分数

```
-- where单行子查询
SELECT
    s.stu_id,
    s.stu_name,
    s.grade
FROM
    student s
WHERE
    s.grade > (
        SELECT
            AVG(s.grade)
        FROM
            student s
```

```
    )
```

执行结果

```
-- where单行子查询,student表中查询grade大于平均分（73.33）的学生id，分数
SELECT
  s.stu_id,
  s.stu_name,
  s.grade
FROM
  student s
WHERE
  s.grade > (
    SELECT
      AVG(s.grade)
    FROM
      student s
  )
```

信息 | 结果1 | 概况 | 状态

stu_id	stu_name	grade
2	小花	90
3	小王	70
6	小赵	80

2.where/from多行子查询

```
-- student表中查询属于班级平均分最高班级的学生信息
SELECT
    s.stu_id,
    s.stu_name,
    s.grade,
    s.class
FROM
    student s
WHERE
    s.class IN (
        SELECT
            class
        FROM
            (
                SELECT
                    class,
                    AVG(grade) avg_grade
                FROM
                    student
                GROUP BY
                    class
                ORDER BY
                    AVG(grade) DESC
                LIMIT 1
            ) sub_1
    )
```

执行结果

```
-- student表中查询属于班级平均分最高班级的学生信息
SELECT
  s.stu_id,
  s.stu_name,
  s.grade,
  s.class
FROM
  student s
WHERE
  s.class IN (
    SELECT
      class
    FROM
      (
        SELECT
          class,
          AVG(grade) avg_grade
        FROM
          student
        GROUP BY
          class
        ORDER BY
          AVG(grade) DESC
        LIMIT 1
      ) sub_1
  )
```

信息 结果1 概况 状态

stu_id	stu_name	grade	class
2	小花	90	2
3	小王	70	2
5	小陈	60	2

上面这个嵌套子查询，一共用了两层子查询：

首先最里层的子查询计算了每个班级的平均分，然后用order by子句按班级平均分倒序排列(最高分排第一个)，然后limit 1限制输出一行数据，得到了最高分的班级行，接着把这层子查询的结果放到外面一层子查询的from语句后面。

第二层子查询得到class，最后外查询的where子句判断class是不是等于第二层子查询中的某一个

3.all

```
-- student表中最高分的学生信息
SELECT * from student s WHERE s.grade>=ALL(SELECT s.grade FROM student s)
```

执行结果

```
-- student表中中的最高分的学生信息
SELECT * from student s WHERE s.grade>=ALL(SELECT s.grade FROM student s)
```

信息 结果1 概况 状态

stu_id	stu_name	age	class	grade
2	小花	18	2	90

4.any

```
-- student表中非最低分的学生信息
SELECT * from student s WHERE s.grade>ANY(SELECT s.grade FROM student s)
```

执行结果

```
-- student表中非最低分的学生信息
SELECT * from student s WHERE s.grade>ANY(SELECT s.grade FROM student s)
```

信息 结果1 概况 状态

stu_id	stu_name	age	class	grade
1	小明	17	1	60
2	小花	18	2	90
3	小王	16	2	70
5	小陈	17	2	60
6	小赵	15	1	80

注释：只有最低分的学生成绩不大于任何一个人的分数，被排除在外

四、case when

case when语句时sql中的一个非常重要的功能，可以完成很多复杂的计算，相当于一个表达式。可以放在任何可放表达式的地方。

语法：

case when 条件 then 结果 when 条件 then 结果 else end

else可不加，是缺省条件下的值，如果不加，有缺省情况则为null。

case when还可以和group by语句搭配使用，用在sum，count，max等聚合函数内部

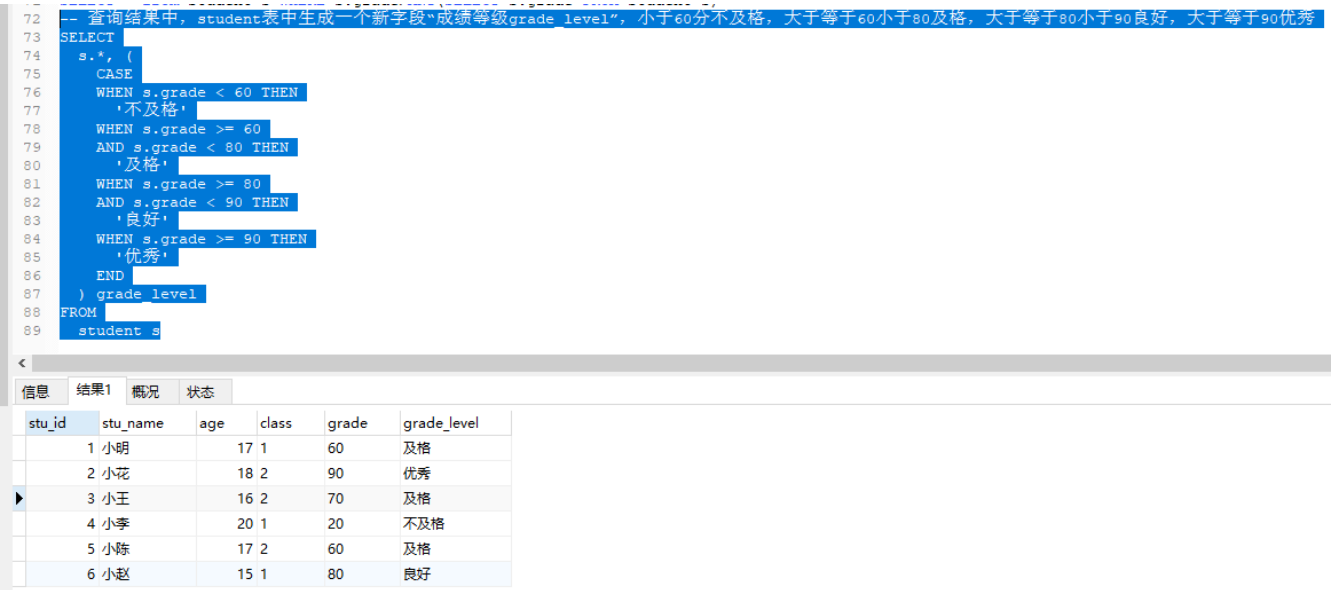
1.例1

查询结果中，student表中生成一个新字段“成绩等级grade_level”，小于60分不及格，大于等于60小于80及格，大于等于80小于90良好，大于等于90优秀

```
SELECT
    s.*, (
        CASE
```

```
        WHEN s.grade < 60 THEN
            '不及格'
        WHEN s.grade >= 60
        AND s.grade < 80 THEN
            '及格'
        WHEN s.grade >= 80
        AND s.grade < 90 THEN
            '良好'
        WHEN s.grade >= 90 THEN
            '优秀'
        END
    ) grade_level
FROM
    student s
```

执行结果



```
-- 查询结果中，student表中生成一个新字段"成绩等级grade_level"，小于60分不及格，大于等于60小于80及格，大于等于80小于90良好，大于等于90优秀
SELECT
  s.*, (
    CASE
    WHEN s.grade < 60 THEN
      '不及格'
    WHEN s.grade >= 60
    AND s.grade < 80 THEN
      '及格'
    WHEN s.grade >= 80
    AND s.grade < 90 THEN
      '良好'
    WHEN s.grade >= 90 THEN
      '优秀'
    END
  ) grade_level
FROM
  student s
```

信息 结果1 概况 状态

stu_id	stu_name	age	class	grade	grade_level
1	小明	17	1	60	及格
2	小花	18	2	90	优秀
3	小王	16	2	70	及格
4	小李	20	1	20	不及格
5	小陈	17	2	60	及格
6	小赵	15	1	80	良好

2.例2

计算每个班级中，及格和不及格的人数

```
SELECT
    s.class,
    COUNT(
        CASE
        WHEN s.grade < 60 THEN
            s.stu_id
        END
    ) failed_num,
    COUNT(
        CASE
        WHEN s.grade >= 60 THEN
            s.stu_id
        END
```

```
    ) pass_num
FROM
    student s
GROUP BY
    s.class
```

执行结果

```
-- 计算每个班级中，及格和不及格的人数
SELECT
  s.class,
  COUNT(
    CASE
    WHEN s.grade < 60 THEN
      s.stu_id
    END
  ) failed_num,
  COUNT(
    CASE
    WHEN s.grade >= 60 THEN
      s.stu_id
    END
  ) pass_num
FROM
  student s
GROUP BY
  s.class
```

信息 | 结果1 | 概况 | 状态

class	failed_num	pass_num
1	1	2
2	0	3

五、组合查询

sql中可用union或union all将多个查询结果拼接起来，两者的区别是union会对结果重新排序，而union all不会，所以一般为了节省内存在不需要排序的情况下用union all更好

eg.

```
-- 组合查询，查询两个班都及格的学生信息
SELECT * FROM student s1 WHERE s1.grade >=60 ORDER BY s1.class ASC

SELECT * FROM student s1 WHERE s1.class=1 AND s1.grade>=60
UNION ALL
SELECT * FROM student s1 WHERE s1.class=2 AND s1.grade>=60
```

执行结果

```sql
-- 查询两个班都及格的学生信息
SELECT * FROM student s1 WHERE s1.grade >=60 ORDER BY s1.class ASC

SELECT * FROM student s1 WHERE s1.class=1 AND s1.grade>=60
UNION ALL
SELECT * FROM student s1 WHERE s1.class=2 AND s1.grade>=60
```

信息 | 结果1 | 概况 | 状态

stu_id	stu_name	age	class	grade
1	小明	17	1	60
6	小赵	15	1	80
2	小花	18	2	90
3	小王	16	2	70
5	小陈	17	2	60

六、常用函数

```sql
-- 字符串函数
-- 截取字符串string，从start开始的length个字符，类似excel的mid
substring(string,start,length)
-- 截取字符串string，从最左边开始的length个字符，类似excel的left
left(string,length)
-- 截取字符串string，从最右边开始的length个字符，类似excel的right
right(string,length)
-- mysql中查找string1在string2中出现的位置
instr(string1,string2[start_location])
-- mysql中删除指定个数字符，并在指定位置处插入新字符
insert(string1,start,length,string2)
```

eg.

```sql
SELECT
    substring(
        '中国浙江省杭州市',
        3,
        3
    ) AS province,
    LEFT (
        '中国浙江省杭州市',
        2
    ) AS country,
    RIGHT (
        '中国浙江省杭州市',
        3
    ) AS city,
    instr(
        '中国浙江省杭州市',
        '杭州市'
    ) AS index_city,
```

```
    INSERT (
        '中国浙江省杭州市',
        6,
        3,
        '宁波市'
    ) AS replace_city
```

执行结果

```
SELECT
  substring(
    '中国浙江省杭州市',
    3,
    3
  ) AS province,
  LEFT (
    '中国浙江省杭州市',
    2
  ) AS country,
  RIGHT (
    '中国浙江省杭州市',
    3
  ) AS city,
  instr(
    '中国浙江省杭州市',
    '杭州市'
  ) AS index_city,
  INSERT (
    '中国浙江省杭州市',
    6,
    3,
    '宁波市'
  ) AS replace_city
```

信息 | 结果1 | 概况 | 状态

province	country	city	index_city	replace_city
浙江省	中国	杭州市	6	中国浙江省宁波市

七、日期函数

```sql
-- 日期函数，不同数据库会有一些区别，此处以mysql为例
-- 当前日期
current_date()
-- date加减expr的unit(年月日周)的date
date_add(date,interval expr unit)
-- expr1减去expr2的天数
datediff(expr1,expr2)
-- expr1减去expr2的时间
timediff(expr1,expr2)
--格式化日期
date_format(date,format)
-- 将字符转换成日期
str_to_date(str,format)
```

eg.

```sql
SELECT
    CURRENT_DATE () AS now,
    date_add(
        CURRENT_DATE (),
        INTERVAL - 1 DAY
    ) AS yesterday,
    datediff(
        CURRENT_DATE (),
        DATE_ADD(
            CURRENT_DATE (),
            INTERVAL - 1 DAY
        )
    ) AS date_cha,
    date_format(CURRENT_DATE(), '%Y/%m') AS ym,
    str_to_date('2020-07-02', '%Y-%m-%d') AS strdate
```

执行结果

```sql
SELECT
  CURRENT_DATE () AS now,
  date_add(
    CURRENT_DATE (),
    INTERVAL - 1 DAY
  ) AS yesterday,
  datediff(
    CURRENT_DATE (),
    DATE_ADD(
      CURRENT_DATE (),
      INTERVAL - 1 DAY
    )
  ) AS date_cha,
  date_format(CURRENT_DATE(), '%Y/%m') AS ym,
  str_to_date('2020-07-02', '%Y-%m-%d') AS strdate
```

信息 结果1 概况 状态

now	yesterday	date_cha	ym	strdate
2020-07-02	2020-07-01	1	2020/07	2020-07-02

```sql
-- 其他一些常用函数
-- 对某个数据列进行指定小数位四舍五入
round(columns,decimals)
-- 返回某个字段长短
len(str)
-- 将字符全部小写
lower(str)
-- 将字符全部大写
upper(str)
-- 返回第一行记录
first()
-- 返回最后一个记录的值
last()
-- 类型转化，比如cast(str as bigint)
cast(expr as stype)
-- 将时间戳转换为时间
from_unixtime(timestamp)
```

八、窗口函数

窗口函数也称为OLAP函数，可以对数据库数据进行实时分析处理。语法如下：

```sql
<窗口函数> over ([partition by <分组columns>]
            order by <排序columns> [窗口子句])
```

窗口函数经常用来计算排序，也可以用在一些聚合函数上，下面举例说明，一张新的学生表student_new

对象 | 数据分析 @数据分析 (localho... | student_new @数据分析 (loc...

开始事务 | 备注 | 筛选 | 排序 | 导入 | 导出

stu_id	stu_name	age	class	grade
1	小明	17	1	60
2	小花	18	2	90
3	小王	16	2	70
4	小李	20	1	20
5	小陈	17	2	60
6	小赵	15	1	80
7	小钱	17	1	90
8	小冯	19	2	90
9	小孙	17	1	80
10	小周	18	2	70

执行结果

```sql

-- 用于排序，row_number,rank,dense_rank mysql8.0支持窗口函数
SELECT a.*,
row_number() over(PARTITION by class order by grade desc)as row_num,
rank() over(PARTITION by class order by grade desc)as rank_num,
dense_rank() over(PARTITION by class order by grade desc)as dense_num,
FROM student_new a;-- 这里会报错

-- 用union实现排序
SELECT * FROM
(SELECT * FROM student_new a WHERE a.class=1 ORDER BY a.grade) a1
UNION ALL
SELECT * FROM
(SELECT * FROM student_new a WHERE a.class=2 ORDER BY a.grade) a2; -- order by不能直接出现在union的子句中，但是可以出现在子句的子句中
```

信息 | 结果1 | 概况 | 状态

stu_id	stu_name	age	class	grade
4	小李	20	1	20
1	小明	17	1	60
6	小赵	15	1	80
9	小孙	17	1	80
7	小钱	17	1	90
5	小陈	17	2	60
3	小王	16	2	70
10	小周	18	2	70
2	小花	18	2	90
8	小冯	19	2	90

【举个栗子_12】窗口函数-排序

```
--用于排序，row_number,rank,dense_rank
SELECT a.*,
       row_number() over(partition by class order by grade desc) as row_num,
       rank() over(partition by class order by grade desc) as rank_num,
       dense_rank() over(partition by class order by grade desc) as dense_num,
FROM student_new a
```

stu_id	stu_name	age	class	grade	row_num	rank_number	dense_rank
7	小钱	17	1	90	1	1	1
6	小赵	15	1	80	2	2	2
9	小孙	17	1	80	3	2	2
1	小明	17	1	60	4	4	3
4	小李	20	1	20	5	5	4
2	小花	18	2	90	1	1	1
8	小冯	19	2	90	2	1	1
3	小王	16	2	70	3	3	2
10	小周	18	2	70	4	知乎 @呆呆玩数据	2
5	小陈	17	2	60	5	5	3

由上面这个例子可以清晰地看出三个窗口排序函数的区别，row_number是在每个分组窗口中给定唯一序号，而rank碰到相等值序号一样，会跳过之后的位次，而dense_rank碰到相等值序号也一致，但不跳过之后的位次。