1、**substr(gmt\_start, 12, 5)**：这部分代码使用了 MySQL 的 substr 函数，从字段 gmt\_start 的第 12 个字符开始提取连续 5 个字符，即提取出 gmt\_start 字段的时间部分。假设 gmt\_start 的格式为 YYYY-MM-DD HH:MM:SS，那么这部分代码提取的是时间的小时和分钟部分。

2、**concat(left(gmt\_start, 10), ' ', a4, ':00')**：这部分代码使用了 MySQL 的字符串函数来构建新的日期时间值。

3、**ceil函数**是一个数学函数，用于向上取整一个数。具体来说，CEIL函数将一个数字参数作为输入，并返回大于或等于该数字的最小整数。

例如，CEIL(3.14)的结果是4，CEIL(5)的结果是5，CEIL(-2.5)的结果是-2。在SQL中，CEIL函数通常用于计算向上取整后的费用、数量、百分比等数字值。

4、**greatest**函数是一个函数，用于返回一组数值中的最大值。具体来说，GREATEST函数将多个参数作为输入，并返回这些参数中的最大值。如果所有参数均为NULL，则GREATEST函数返回NULL。

例如，GREATEST(3, 5, 2)的结果是5，GREATEST(10, 10, 5, 7)的结果是10，GREATEST(NULL, 1, 2)的结果是2。

5、**unix\_timestamp**函数是一个日期和时间函数，用于将日期和时间转换为UNIX时间戳。UNIX时间戳是指从1970年1月1日00:00:00 UTC（协调世界时）开始到指定日期和时间之间的秒数，通常用于计算机系统中表示时间的一种方式。

UNIX\_TIMESTAMP函数将日期和时间参数转换为UNIX时间戳，返回一个整数值。例如，UNIX\_TIMESTAMP('2021-08-01 12:00:00')的结果是1627824000，表示从1970年1月1日00:00:00 UTC到2021年8月1日12:00:00之间的秒数。

6、**inner join、left /right join、full [outer] join**的区别

inner join产生的结果是A与B的交集；

full [outer] join产生的结果是A与B的并集，但需要注意的是，对于没有匹配的记录，则会以null作为值，可用ifnull进行判断；

A left join B产生表A的完全集，而B表中没有匹配的则以null值取代（right同理）。

7、**ifnull**( left( right( d.gmt\_start, 8), 5), lpad( substr( b.gongzuojihua\_\_name, 6, 4), 5, '0') ) a4

这段代码是一个SQL查询中的表达式，用于计算一个名为a4的列的值。让我们逐步解释这段代码的含义：

d.gmt\_start：这是一个表达式，表示从表格 d 的列 gmt\_start 中获取值。假设这个值是一个日期或时间戳，例如 "2023-08-08 10:30:00"。

right(d.gmt\_start, 8)：这是一个函数调用，它从 d.gmt\_start 的右侧返回最后8个字符。假设返回的值是 "10:30:00"。

left(right(d.gmt\_start, 8), 5)：这是另一个函数调用，它从上一步返回的字符串中获取前5个字符。假设返回的值是 "10:30"。

ifnull(left(right(d.gmt\_start, 8), 5), ...)：这是一个函数调用，它检查前一步返回的值是否为 NULL（空值）。如果不是 NULL，则返回该值；否则，继续执行下一步。

substr(b.gongzuojihua\_\_name, 6, 4)：这是一个函数调用，它从字符串 b.gongzuojihua\_\_name 的第6个字符开始，获取长度为4的子字符串。假设返回的值是 "1234"。

lpad(substr(b.gongzuojihua\_\_name, 6, 4), 5, '0')：这是另一个函数调用，它将上一步返回的字符串用字符 '0' 左填充，直到总长度达到5个字符。假设返回的值是 "01234"。

ifnull(left(right(d.gmt\_start, 8), 5), lpad(substr(b.gongzuojihua\_\_name, 6, 4), 5, '0'))：这是最终的表达式。如果步骤 4 返回的值不是 NULL，则返回步骤 4 的值；否则，返回步骤 6 的值。最终的结果将赋给列 a4。

8、表达式**if**(substr(gmt\_start,12,5)<=a4, concat(left(gmt\_start,10),' ',a4,':00'), gmt\_start) 返回一个值，要么是经过处理的日期时间字符串，要么是原始的 gmt\_start 值，具体取决于 substr(gmt\_start,12,5) 是否小于等于 a4。

9、**cast(123 as string)**

CAST(expression AS data\_type)

其中，expression是要转换的表达式或列，data\_type是要转换为的目标数据类型。

在上述示例中，expression是整数值123，data\_type是字符串类型（string）。所以，CAST函数将整数123转换为字符串类型。

10、

FLOOR(DATEDIFF(CURDATE(), birthday) / 365) 是一个SQL表达式，用于计算当前日期与出生日期之间的年龄。

CURDATE() 是一个日期函数，返回当前日期。

DATEDIFF() 是一个日期函数，用于计算两个日期之间的差异。在这个表达式中，它用于计算当前日期与出生日期之间的天数差异。

FLOOR() 是一个数学函数，用于向下取整，将小数部分舍去，保留最接近的整数。

所以，FLOOR(DATEDIFF(CURDATE(), birthday) / 365) 表达式将计算出从出生日期到当前日期之间的天数差异，并将其除以365，得到一个带有小数的年龄值。然后，使用FLOOR()函数将小数部分舍去，得到最接近的整数，表示实际年龄。这个表达式返回的是一个整数值，表示以年为单位的实际年龄。

11、cast(floor(DATEDIFF(FROM\_UNIXTIME(UNIX\_TIMESTAMP(CAST(dt AS STRING), 'yyyyMMdd'), 'yyyy-MM-dd'), to\_date(a.gbdat)) / 365) as string) employee\_age --年龄

上述语句做了以下操作：

1. 使用 CAST(dt AS STRING) 将列 dt 转换为字符串类型。这假设你有一个名为 dt 的列，其中包含整数格式的日期数据。
2. 使用 UNIX\_TIMESTAMP(CAST(dt AS STRING), 'yyyyMMdd') 将字符串日期转换为 UNIX 时间戳。这里假设你的日期格式是 'yyyyMMdd'。
3. 使用 FROM\_UNIXTIME 函数将 UNIX 时间戳转换回日期格式，并指定输出日期的格式为 'yyyy-MM-dd'。
4. 使用 to\_date(a.gbdat) 将列 a.gbdat 转换为日期类型。这里假设你有一个名为 a.gbdat 的列，其中包含日期数据。
5. 使用 DATEDIFF 函数计算两个日期之间的天数差异。
6. 使用 FLOOR 函数将天数差异除以 365，得到年龄的近似值。
7. 最后，使用 CAST(... AS STRING) 将年龄值转换回字符串类型。

这个查询的结果将返回一个名为 employee\_age 的列，其中包含计算得到的年龄近似值。

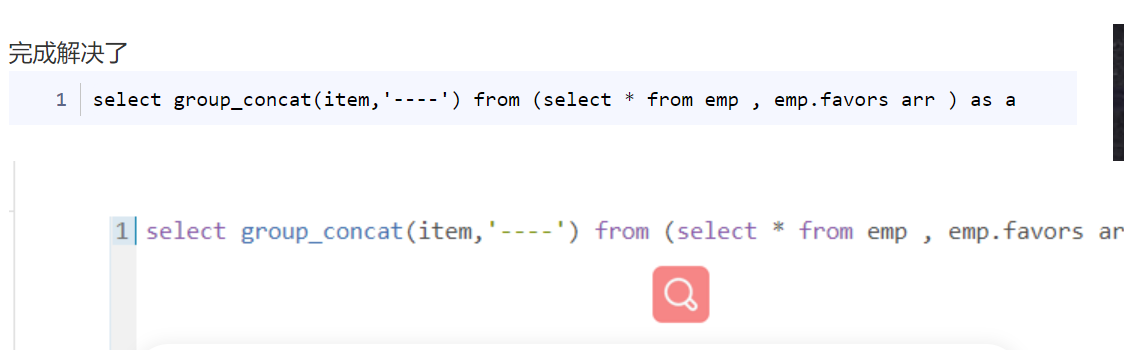
请确保在运行查询之前，仔细检查列名、表名和数据类型，以确保查询逻辑和上下文的正确性。

12、窗口函数

【窗口函数】  over （partition by 【分组依据的字段】 order by 【排序依据的字段】）



13、impala中的复杂类型数据查询，如ARRAY<类型>数组类型数据的查询，需要将复杂类型当作表处理



14、在Impala中，GROUP\_CONCAT()函数用于将组内的多个值连接成一个字符串。这个函数将组内的多个值按照指定的分隔符连接起来，并返回一个字符串结果。

以下是GROUP\_CONCAT()函数的用法示例：

SELECT id, GROUP\_CONCAT(name, ',') AS concatenated\_names

FROM my\_table

GROUP BY id;

在上面的示例中，我们使用了GROUP\_CONCAT()函数来连接my\_table表中每个组（按照id列分组）中的name列值，并使用逗号作为分隔符。结果集中的每一行包含一个id值和相应组中name列值的连接字符串。

你可以根据需要使用不同的分隔符，只需将分隔符作为GROUP\_CONCAT()函数的第二个参数传递。例如，如果想要使用空格作为分隔符，可以这样写：

SELECT id, GROUP\_CONCAT(name, ' ') AS concatenated\_names

FROM my\_table

GROUP BY id;

这将使用空格将name列值连接起来。

另外，GROUP\_CONCAT()函数还支持对连接结果进行排序。你可以通过在函数中使用ORDER BY子句来指定排序的列和顺序。以下是一个示例：

SELECT id, GROUP\_CONCAT(name, ',' ORDER BY name DESC) AS concatenated\_names

FROM my\_table

GROUP BY id;

在上面的示例中，我们按照name列的逆序对连接结果进行排序，并使用逗号作为分隔符。

总而言之，GROUP\_CONCAT()函数可用于将组内的多个值连接成一个字符串，并可选地指定分隔符和排序方式。

15、doris中建表语句后面需要加上如下代码  
ENGINE=OLAP   
DUPLICATE KEY(`update\_tm`)  
COMMENT "OLAP"  
PARTITION BY RANGE(`dt`)  
()  
DISTRIBUTED BY HASH(`表里面的字段1`, `表里面的字段2`, `表里面的字段3`) BUCKETS 5  -- 不一定要3个 以表的主键为准  
PROPERTIES (  
"replication\_num" = "1",  
"in\_memory" = "false",  
"storage\_format" = "V2"  
);

16、**hive sql中拆分+合并字符串**

--获取员工信息表sdl.s\_hr\_tb\_sap2hr\_staffinfo\_n中的部门全称full\_name

--@sql(createOrReplaceTempView=cache\_s\_hr\_tb\_sap2hr\_staffinfo\_n\_fullname)

--@sql(cache=cache\_s\_hr\_tb\_sap2hr\_staffinfo\_n\_fullname)

select

d.pernr,

d.full\_path,

concat\_ws(',',collect\_set(d.short)) as full\_name

from

(

select

b.pernr,

b.full\_path,

b.id,

c.objid,

c.short

from(

select

a.pernr,a.full\_path,t.id

from sdl.s\_hr\_tb\_sap2hr\_staffinfo\_n a

lateral view explode(split(a.full\_path,',')) t as id

where a.dt=${statisdate}

) b left join sdl.s\_hr\_tb\_sap2hr\_orginfo\_n c

on b.id=c.objid and c.dt=20230822

) d

group by d.pernr,d.full\_path;

解释说明：

lateral view explode(split(a.full\_path,',')) t as id

在这个表达式中，lateral view explode(split(a.full\_path,',')) t as id 是一种在Hive中使用Lateral View进行扁平化操作的语法。

1. split(a.full\_path, ',')：这部分使用了字符串函数split，它将字符串按照指定的分隔符进行拆分，并返回一个数组。在这里，a.full\_path是需要拆分的字符串，逗号（,）被用作分隔符。
2. explode(split(a.full\_path, ','))：这部分使用了表生成函数explode，它的作用是将数组中的元素展开为多行数据。在这里，explode函数被用于展开split函数返回的数组，将数组中的每个元素拆分为一行。
3. lateral view explode(split(a.full\_path, ',')) t as id：这部分使用了lateral view语法，它允许在查询中将一行数据展开为多行数据。在这里，lateral view explode(split(a.full\_path, ',')) 将a.full\_path列按照逗号进行拆分，并将拆分后的每个元素展开为多行数据。t 是展开后的表的别名，id 是展开后的每行数据的列名。

综上所述，lateral view explode(split(a.full\_path, ',')) t as id 的作用是将a.full\_path列中的字符串按照逗号进行拆分，并将拆分后的每个元素展开为多行数据，每行数据包含一个拆分后的元素，并将该列命名为id。

以full\_path =（’ 00001,00002,00003’）为例上述代码得到的结果如下：

a.pernr，a.full\_path，t.id

xx xx 00001

xx xx 00002

xx xx 00003

concat\_ws(',',collect\_set(d.short)) as full\_name

…

group by d.pernr,d.full\_path

1.collect\_set(d.short)：这部分使用了聚合函数collect\_set，它用于收集并去重一个列（在这里是d.short列）的所有值，并将它们组成一个集合。集合是一个无序的数据结构，不包含重复的元素。

2.concat\_ws(',', collect\_set(d.short))：这部分使用了字符串函数concat\_ws，它的作用是将多个值按照指定的分隔符进行连接。在这里，逗号（,）被用作分隔符。concat\_ws的第一个参数是分隔符，后面跟着需要连接的值。

因此，concat\_ws(',', collect\_set(d.short)) 将集合中的所有值以逗号为分隔符进行连接，形成一个字符串。

collect\_list函数与collect\_set函数类似，也是用于收集一列中的值并生成一个集合。不同之处在于，collect\_list函数会按照数据的顺序将值收集到集合中，保留元素的顺序。

以下是使用collect\_list函数生成有序集合的示例：

concat\_ws(',', collect\_list(d.short)) as ordered\_set

在上述示例中，我们使用collect\_list函数来替代原来的collect\_set函数，将d.short列中的值收集到一个有序的集合中。然后，使用concat\_ws函数将集合中的值以逗号为分隔符连接成一个字符串。

注意，使用collect\_list函数生成有序集合可能会导致性能上的一些影响，因为它需要保留元素的顺序。如果你的数据量很大，可能需要评估是否真正需要有序集合，以及对性能的影响是否可以接受。

17、impala中临时表创建及引用流程

解决方法：

（1） 建立关联Mysql数据库的临时表需要在临时表sql语句前加如下注解，

--@sql(mysql=true)

--@sql(createOrReplaceTempView=cache\_i\_oa\_check\_work)

--@sql(cache=cache\_i\_oa\_check\_work)

--@sql(jdbcimport\_partition\_lowerBound=0)

--@sql(jdbcimport\_partition\_upperBound=1000)

--@sql(jdbcimport\_partitions=1)

--@sql(jdbcimport=app)

（2） 建立关联impala本身数据库的临时表需要在临时表sql语句前加如下注解，

--@sql(createOrReplaceTempView=cache\_i\_oa\_check\_work)

--@sql(cache=cache\_i\_oa\_check\_work)