遇到 X情况 就 执行X 遇不到就 执行B 结束  
case when …… then …… else …… end

Case具有两种格式。简单Case函数和Case搜索函数。

--简单Case函数

CASE sex

WHEN '1' THEN '男'

WHEN '2' THEN '女'

ELSE '其他' END

--Case搜索函数

CASE WHEN sex = '1' THEN '男'

WHEN sex = '2' THEN '女'

ELSE '其他' END

这两种方式，可以实现相同的功能。简单Case函数的写法相对比较简洁，但是和Case搜索函数相比，功能方面会有些限制，比如写判断式。   
**还有一个需要注意的问题，Case函数只返回第一个符合条件的值，剩下的Case部分将会被自动忽略。**

--比如说，下面这段SQL，你永远无法得到“第二类”这个结果

CASE WHEN col\_1 IN ( 'a', 'b') THEN '第一类'

WHEN col\_1 IN ('a') THEN '第二类'

ELSE'其他' END

下面我们来看一下，使用Case函数都能做些什么事情。   
  
**一，已知数据按照另外一种方式进行分组，分析。**   
  
有如下数据:(为了看得更清楚，我并没有使用国家代码，而是直接用国家名作为Primary Key)

|  |  |
| --- | --- |
| **国家（country）** | **人口（population）** |
| 中国 | 600 |
| 美国 | 100 |
| 加拿大 | 100 |
| 英国 | 200 |
| 法国 | 300 |
| 日本 | 250 |
| 德国 | 200 |
| 墨西哥 | 50 |
| 印度 | 250 |

根据这个国家人口数据，统计亚洲和北美洲的人口数量。应该得到下面这个结果。

|  |  |
| --- | --- |
| **洲** | **人口** |
| 亚洲 | 1100 |
| 北美洲 | 250 |
| 其他 | 700 |

想要解决这个问题，你会怎么做？生成一个带有洲Code的View，是一个解决方法，但是这样很难动态的改变统计的方式。   
如果使用Case函数，SQL代码如下:

SELECT SUM(population),

CASE country

WHEN '中国' THEN '亚洲'

WHEN '印度' THEN '亚洲'

WHEN '日本' THEN '亚洲'

WHEN '美国' THEN '北美洲'

WHEN '加拿大' THEN '北美洲'

WHEN '墨西哥' THEN '北美洲'

ELSE '其他' END

FROM Table\_A

GROUP BY CASE country

WHEN '中国' THEN '亚洲'

WHEN '印度' THEN '亚洲'

WHEN '日本' THEN '亚洲'

WHEN '美国' THEN '北美洲'

WHEN '加拿大' THEN '北美洲'

WHEN '墨西哥' THEN '北美洲'

ELSE '其他' END;

同样的，我们也可以用这个方法来判断工资的等级，并统计每一等级的人数。SQL代码如下；

SELECT

CASE WHEN salary <= 500 THEN '1'

WHEN salary > 500 AND salary <= 600 THEN '2'

WHEN salary > 600 AND salary <= 800 THEN '3'

WHEN salary > 800 AND salary <= 1000 THEN '4'

ELSE NULL END salary\_class,

COUNT(\*)

FROM Table\_A

GROUP BY

CASE WHEN salary <= 500 THEN '1'

WHEN salary > 500 AND salary <= 600 THEN '2'

WHEN salary > 600 AND salary <= 800 THEN '3'

WHEN salary > 800 AND salary <= 1000 THEN '4'

ELSE NULL END;

**二，用一个SQL语句完成不同条件的分组。**   
  
有如下数据

|  |  |  |
| --- | --- | --- |
| **国家（country）** | **性别（sex）** | **人口（population）** |
| 中国 | 1 | 340 |
| 中国 | 2 | 260 |
| 美国 | 1 | 45 |
| 美国 | 2 | 55 |
| 加拿大 | 1 | 51 |
| 加拿大 | 2 | 49 |
| 英国 | 1 | 40 |
| 英国 | 2 | 60 |

按照国家和性别进行分组，得出结果如下

|  |  |  |
| --- | --- | --- |
| **国家** | **男** | **女** |
| 中国 | 340 | 260 |
| 美国 | 45 | 55 |
| 加拿大 | 51 | 49 |
| 英国 | 40 | 60 |

普通情况下，用UNION也可以实现用一条语句进行查询。但是那样增加消耗(两个Select部分)，而且SQL语句会比较长。   
下面是一个是用Case函数来完成这个功能的例子

SELECT country,

SUM( CASE WHEN sex = '1' THEN

population ELSE 0 END), --男性人口

SUM( CASE WHEN sex = '2' THEN

population ELSE 0 END) --女性人口

FROM Table\_A

GROUP BY country;

这样我们使用Select，完成对二维表的输出形式，充分显示了Case函数的强大。   
  
**三，在Check中使用Case函数。**   
  
在Check中使用Case函数在很多情况下都是非常不错的解决方法。可能有很多人根本就不用Check，那么我建议你在看过下面的例子之后也尝试一下在SQL中使用Check。   
下面我们来举个例子   
公司A，这个公司有个规定，女职员的工资必须高于1000块。如果用Check和Case来表现的话，如下所示

CONSTRAINT check\_salary CHECK

( CASE WHEN sex = '2'

THEN CASE WHEN salary > 1000

THEN 1 ELSE 0 END

ELSE 1 END = 1 )

如果单纯使用Check，如下所示

CONSTRAINT check\_salary CHECK

( sex = '2' AND salary > 1000 )

女职员的条件倒是符合了，男职员就无法输入了。

**四，根据条件有选择的UPDATE。**   
  
例，有如下更新条件

1. 工资5000以上的职员，工资减少10%
2. 工资在2000到4600之间的职员，工资增加15%

很容易考虑的是选择执行两次UPDATE语句，如下所示

--条件1

UPDATE Personnel

SET salary = salary \* 0.9

WHERE salary >= 5000;

--条件2

UPDATE Personnel

SET salary = salary \* 1.15

WHERE salary >= 2000 AND salary < 4600;

但是事情没有想象得那么简单，假设有个人工资5000块。首先，按照条件1，工资减少10%，变成工资4500。接下来运行第二个SQL时 候，因为这个人的工资是4500在2000到4600的范围之内， 需增加15%，最后这个人的工资结果是5175,不但没有减少，反而增加了。如果要是反过来执行，那么工资4600的人相反会变成减少工资。暂且不管这个 规章是多么荒诞，如果想要一个SQL 语句实现这个功能的话，我们需要用到Case函数。代码如下:

UPDATE Personnel

SET salary = CASE WHEN salary >= 5000

　 THEN salary \* 0.9

WHEN salary >= 2000 AND salary < 4600

THEN salary \* 1.15

ELSE salary END;

这里要注意一点，最后一行的ELSE salary是必需的，要是没有这行，不符合这两个条件的人的工资将会被写成NUll,那可就大事不妙了。在Case函数中Else部分的默认值是NULL，这点是需要注意的地方。   
这种方法还可以在很多地方使用，比如说变更主键这种累活。   
一般情况下，要想把两条数据的Primary key,a和b交换，需要经过临时存储，拷贝，读回数据的三个过程，要是使用Case函数的话，一切都变得简单多了。

|  |  |  |
| --- | --- | --- |
| **p\_key** | **col\_1** | **col\_2** |
| a | 1 | 张三 |
| b | 2 | 李四 |
| c | 3 | 王五 |

假设有如上数据，需要把主键a和b相互交换。用Case函数来实现的话，代码如下

UPDATE SomeTable

SET p\_key = CASE WHEN p\_key = 'a'

THEN 'b'

WHEN p\_key = 'b'

THEN 'a'

ELSE p\_key END

WHERE p\_key IN ('a', 'b');

同样的也可以交换两个Unique key。需要注意的是，如果有需要交换主键的情况发生，多半是当初对这个表的设计进行得不够到位，建议检查表的设计是否妥当。   
  
**五，两个表数据是否一致的检查。**   
  
Case函数不同于DECODE函数。在Case函数中，可以使用BETWEEN,LIKE,IS NULL,IN,EXISTS等等。比如说使用IN,EXISTS，可以进行子查询，从而 实现更多的功能。   
下 面具个例子来说明，有两个表，tbl\_A,tbl\_B，两个表中都有keyCol列。现在我们对两个表进行比较，tbl\_A中的keyCol列的数据如果 在tbl\_B的keyCol列的数据中可以找到， 返回结果'Matched',如果没有找到，返回结果'Unmatched'。   
要实现下面这个功能，可以使用下面两条语句

--使用IN的时候

SELECT keyCol,

CASE WHEN keyCol IN ( SELECT keyCol FROM tbl\_B )

THEN 'Matched'

ELSE 'Unmatched' END Label

FROM tbl\_A;

--使用EXISTS的时候

SELECT keyCol,

CASE WHEN EXISTS ( SELECT \* FROM tbl\_B

WHERE tbl\_A.keyCol = tbl\_B.keyCol )

THEN 'Matched'

ELSE 'Unmatched' END Label

FROM tbl\_A;

使用IN和EXISTS的结果是相同的。也可以使用NOT IN和NOT EXISTS，但是这个时候要注意NULL的情况。   
  
**六，在Case函数中使用合计函数**   
  
假设有下面一个表

|  |  |  |  |
| --- | --- | --- | --- |
| **学号(std\_id)** | **课程ID(class\_id)** | **课程名(class\_name)** | **主修flag（main\_class\_flg)** |
| 100 | 1 | 经济学 | Y |
| 100 | 2 | 历史学 | N |
| 200 | 2 | 历史学 | N |
| 200 | 3 | 考古学 | Y |
| 200 | 4 | 计算机 | N |
| 300 | 4 | 计算机 | N |
| 400 | 5 | 化学 | N |
| 500 | 6 | 数学 | N |

有的学生选择了同时修几门课程 (100,200)也有的学生只选择了一门课程(300,400,500)。选修多门课程的学生，要选择一门课程作为主修，主修flag里面写入 Y。只选择一门课程的学生，主修flag为N(实际上要是写入Y的话，就没有下面的麻烦事了，为了举例子，还请多多包含)。   
现在我们要按照下面两个条件对这个表进行查询

1. 只选修一门课程的人，返回那门课程的ID
2. 选修多门课程的人，返回所选的主课程ID

简单的想法就是，执行两条不同的SQL语句进行查询。   
条件1

--条件1：只选择了一门课程的学生

SELECT std\_id, MAX(class\_id) AS main\_class

FROM Studentclass

GROUP BY std\_id

HAVING COUNT(\*) = 1;

执行结果1

STD\_ID MAIN\_class

------ ----------

300 4

400 5

500 6

条件2

--条件2：选择多门课程的学生

SELECT std\_id, class\_id AS main\_class

FROM Studentclass

WHERE main\_class\_flg = 'Y' ;

执行结果2

STD\_ID MAIN\_class

------ ----------

100 1

200 3

如果使用Case函数，我们只要一条SQL语句就可以解决问题，具体如下所示

SELECT std\_id,

CASE WHEN COUNT(\*) = 1 --只选择一门课程的学生的情况

THEN MAX(class\_id)

ELSE MAX(CASE WHEN main\_class\_flg = 'Y'

THEN class\_id

ELSE NULL END

)

END AS main\_class

FROM Studentclass

GROUP BY std\_id;

运行结果

STD\_ID MAIN\_class

------ ----------

100 1

200 3

300 4

400 5

500 6

通过在Case函数中嵌套Case函数，在合计函数中使用Case函数等方法，我们可以轻松的解决这个问题。使用Case函数给我们带来了更大的自由度。   
最后提醒一下使用Case函数的新手注意不要犯下面的错误

CASE col\_1

WHEN 1 　 THEN 'Right'

WHEN **NULL** THEN 'Wrong'

END

在这个语句中When Null这一行总是返回unknown，所以永远不会出现Wrong的情况。因为这句可以替换成WHEN col\_1 = NULL，这是一个错误的用法，这个时候我们应该选择用WHEN col\_1 IS NULL。