SQL进阶教程

Advanced SQL Tutorial

Stephen CUI[1]

November 4, 2022

[1]cuixuanStephen@gmail.com

目录

Chapter 1

神奇的SQL

1.1 CASE表达式

1.1.1 写在前面

因为 CASE 表达式是不依赖于具体数据库的技术，所以可以提高 SQL 代码的可移植性。

1.1.2 CASE 表达式概述

CASE 表达式有简单 CASE 表达式（simple case expression）和搜索CASE表达式（searched case expression）两种写法。

在编写 SQL 语句的时候需要注意，在发现为真的 WHEN 子句时，CASE 表达式的真假值判断就会中止，而剩余的 WHEN 子句会被忽略。为了避免引起不必要的混乱，使用 WHEN 子句时要注意条件的排他性。

注意事项：

1. **统一各分支返回的数据类型：**一定要注意 CASE 表达式里各个分支返回的数据类型是否一致。某个分支返回字符型，而其他分支返回数值型的写法是不正确的。
2. **不要忘了写END：**使用 CASE 表达式的时候，最容易出现的语法错误是忘记写 END。
3. 养成写ELSE子句的习惯与 END 不同，ELSE 子句是可选的，不写也不会出错。不写 ELSE 子句时， CASE 表达式的执行结果是 NULL。但是不写可能会造成“语法没有错误，结果却不对”这种不易追查原因的麻烦，所以最好明确地写上 ELSE 子句（即便是在结果可以为 NULL 的情况下）。

1.1.3 将已有编号方式转化为新的方式并统计

建表的命令点这里

这里的关键在于将 SELECT 子句里的 CASE 表达式复制到 GROUP BY 子句里。需要注意的是，如果对转换前的列`pref_name`进行 GROUP BY，就得不到正确的结果（因为这并不会引起语法错误，所以容易被忽视）。

这个技巧非常好用。不过，必须在 SELECT 子句和 GROUP BY 子句这两处写一样的 CASE 表达式，这有点儿麻烦。后期需要修改的时候，很容易发生只改了这一处而忘掉改另一处的失误。所以，如果我们可以像下面这样写，那就方便多了。

没错，这里的 GROUP BY 子句使用的正是 SELECT 子句里定义的列的别称——district。但是严格来说，这种写法是违反标准 SQL 的规则的。因为 GROUP BY 子句比 SELECT 语句先执行，所以在 GROUP BY 子句中引用在 SELECT 子句里定义的别称是不被允许的。

1.1.4 用一条SQL语句进行不同条件的统计

进行不同条件的统计是 CASE 表达式的著名用法之一。（建表的命令点这里）

这里是将“行结构”的数据转换成了“列结构”的数据。除了 SUM，COUNT、AVG 等聚合函数也都可以用于将行结构的数据转换成列结构的数据。

这个技巧可贵的地方在于，它能将 SQL 的查询结果转换为二维表的格式。

1.1.5 用CHECK约束定义多个列的条件关系

1.1.6 在UPDATE语句里进行条件分支

如果 CASE 表达式里没有明确指定 ELSE 子句，执行结果会被默认地处理成 ELSE NULL。

这个技巧的应用范围很广。例如，可以用它简单地完成主键值调换这种繁重的工作。通常，当我们想调换主键值 a 和 b 时，需要将主键值临时转换成某个中间值。使用这种方法时需要执行 3 次 UPDATE 操作，但是如果使用 CASE 表达式，1 次就可以做到。

这里没有必要执行 3 次 UPDATE 操作，而且中间值 d 是否总能使用也是问题。而如果使用 CASE 表达式，就不必担心这些，1 次就可以完成调换[1]。

1.1.7 表之间的数据匹配

与 DECODE 函数等相比，CASE 表达式的一大优势在于能够判断表达式。也就是说，在 CASE 表达式里，我们可以使用 BETWEEN、LIKE 和 >、 < 等便利的谓词组合，以及能嵌套子查询的 IN 和 EXISTS 谓词。因此，CASE 表达式具有非常强大的表达能力。

建表命令点击这里Course

我们需要做的是，检查表 OpenCourses 中的各月里有表 CourseMaster 中的哪些课程。这个匹配条件可以用 CASE 表达式来写。

这样的查询没有进行聚合，因此也不需要排序，月份增加的时候仅修改 SELECT 子句就可以了，扩展性比较好。

无论使用 IN 还是 EXISTS，得到的结果是一样的，但从性能方面来说， EXISTS 更好。

1.1.8 在 CASE 表达式中使用聚合函数

接下来介绍一下稍微高级的用法。这个用法乍一看可能让人觉得像是语法错误，实际上却并非如此。

[1]如果在 PostgreSQL 和 MySQL 数据库执行这条 SQL 语句，会因主键重复而出现错误。但是，约束的检查本来就发生在更新完成后，因此更新途中主键一时出现重复也没有问题。事实上，在 Oracle、 DB2、SQL Server 数据库执行都没有问题。

建表命令点击这里StudentClub

这样做也能得到正确的结果，但需要写多条 SQL 语句。而如果使用 CASE 表达式，下面这一条 SQL 语句就可以了。

这条 SQL 语句在 CASE 表达式里使用了聚合函数，又在聚合函数里使用了 CASE 表达式。

1.1.9 本节小结

1. 在 GROUP BY 子句里使用 CASE 表达式，可以灵活地选择作为聚合的单位的编号或等级。这一点在进行非定制化统计时能发挥巨大的威力。
2. 在聚合函数中使用 CASE 表达式，可以轻松地将行结构的数据转换成列结构的数据。
3. 相反，聚合函数也可以嵌套进 CASE 表达式里使用。
4. 相比依赖于具体数据库的函数，CASE 表达式有更强大的表达能力和更好的可移植性。
5. 正因为 CASE 表达式是一种表达式而不是语句，才有了这诸多优点。

1.2 用SQL进行集合运算

SQL 语言的基础之一是集合论。但是，很长一段时间内，由于 SQL 没能很好地支持集合运算，所以相关功能并没有被人们充分地利用。过去这些年，SQL 凑齐了大部分基础的集合运算，人们终于可以真正地使用它了。

1.2.1 写在前面

集合论是 SQL 语言的根基——这是贯穿全书的主题之一。因为它的这个特性，SQL 也被称为面向集合语言。

1.2.2 导入篇：集合运算的几个注意事项

顾名思义，集合运算符的参数是集合，从数据库实现层面上来说就是表或者视图。因为和高中学过的集合代数很像，所以理解起来相对比较容易。但是，SQL 还是有几个特别的地方需要注意一下。

1. SQL 能操作具有重复行的集合，可以通过可选项 ALL来支持
   一般的集合论是不允许集合里存在重复元素的，因此集合 {1, 1, 2, 3, 3, 3} 和集合 {1, 2, 3} 被视为相同的集合。但是关系数据库里的表允许存在重复的行，称为多重集合（multiset, bag）。
   因此，SQL 的集合运算符也提供了允许重复和不允许重复的两种用法。如果直接使用 UNION 或 INTERSECT，结果里就不会出现重复的行。如果想在结果里留下重复行，可以加上可选项 ALL，写作 UNION ALL。ALL 的作用和 SELECT 子句里的 DISTINCT 可选项刚好相反。
   除了运算结果以外，这两种用法还有一个不同。集合运算符为了排除掉重复行，默认地会发生排序，而加上可选项 ALL 之后，就不会再排序，所以性能会有提升。这是非常有效的用于优化查询性能的方法，所以如果不关心结果是否存在重复行，或者确定结果里不会产生重复行，加上可选项 ALL 会更好些。
2. 集合运算符有优先级
   标准 SQL 规定，INTERSECT 比 UNION 和 EXCEPT 优先级更高。因此，当同时使用 UNION 和 INTERSECT，又想让 UNION 优先执行时，必须用括号明确地指定运算顺序

3. 各个 DBMS 提供商在集合运算的实现程度上参差不齐
4. 除法运算没有标准定义

   四则运算里的和（UNION）、差（EXCEPT）、积（CROSS JOIN）都被引入了标准 SQL。但是很遗憾，商（DIVIDE BY）因为各种原因迟迟没能标准化。

1.2.3 比较表和表：检查集合相等性之基础篇

在迁移数据库的时候，或者需要比较备份数据和最新数据的时候，我们需要调查两张表是否是相等的。这里说的“相等”指的是行数和列数以及内容都相同，即“是同一个集合”的意思。

此时做法有两种。我们先看一个简单的，只用 UNION 就能实现的方法。这里先假设已经事先确认了表 `tbl_A` 和表 `tbl_B` 的行数是一样的（如果行数不一样，那就不需要比较其他的了）。

这种做法的原理是什么呢？请回忆一下导入篇：集合运算的几个注意事项里的注意事项 1。如果集合运算符里不加上可选项 ALL，那么重复行就会被排除掉。因此，如果表 `tbl_A` 和表 `tbl_B` 是相等的，排除掉重复行后，两个集合是完全重合的。

当然，我们也可以只比较表里的一部分列或者一部分行。只需要指定一下想要比较的列的名称，或者在 WHERE 子句里加入过滤条件就可以比较了。

1.3 必知必会的窗口函数

1.3.1 什么是窗口

实际上没有哪句代码表明了“这是窗口[2][3]”，只看到那些使用了PARTITION BY子句或ORDER BY子句的查询。

显式定义了窗口，并对其应用了 AVG 函数。这里所说的窗口，就是针对通过 FROM 子句选择的记录集，**使用 ORDER BY 排序和使用 ROWS BETWEEN 定义帧之后所形成的数据集**，其优点是可以重复使用[4]。

窗口函数让人难以理解的原因之一是 1 个窗口函数中包含多个操作:

1. 使用 PARTITION BY 子句[5]分割记录集合。
2. 使用 ORDER BY 子句对记录排序。
3. 使用帧子句定义以当前记录为中心的子集。

Q1:除向前移动之外，帧还可以向“后”移动吗？ 可以，使用FOLLOWING
Q2:可以设置基于列值（而不是行）的帧吗？ 可以，使用RANGE而不是ROWS

下面是帧子句中可以使用的选项：

- ROWS：按行设置移动单位

[2]窗口函数出现于 20 世纪 90 年代到 21 世纪初，当时它也被称为“OLAP 函数”。人们打算把它用于OLAP（联机分析处理，online analytical processing），所以才取了这个名称，但现在已经不怎么使用该名称了。

[3]窗口（window）在英语中原本就有“范围”“宽度”的意思，在系统开发领域也存在“批处理窗口”“维护窗口”这样的术语。在这种情况下，“窗口”与普通意义上的窗口并无直接关系。用这个词表示时间段含义时也是如此。一般来说，该术语不仅用作将集合分割开的子集，还用作默认内含某种顺序性的“范围”。

[4]有名称的窗口函数可以用在 PostgreSQL 和 MySQL 中，但在 Oracle 中使用就会发生错误。

[5]PARTITION BY 子句只用来分割窗口，并不会像 GROUP BY 子句那样对记录进行聚合，因此在应用窗口函数时，记录的行数不会发生改变，这一点与 GROUP BY 子句的功能并不完全一样。

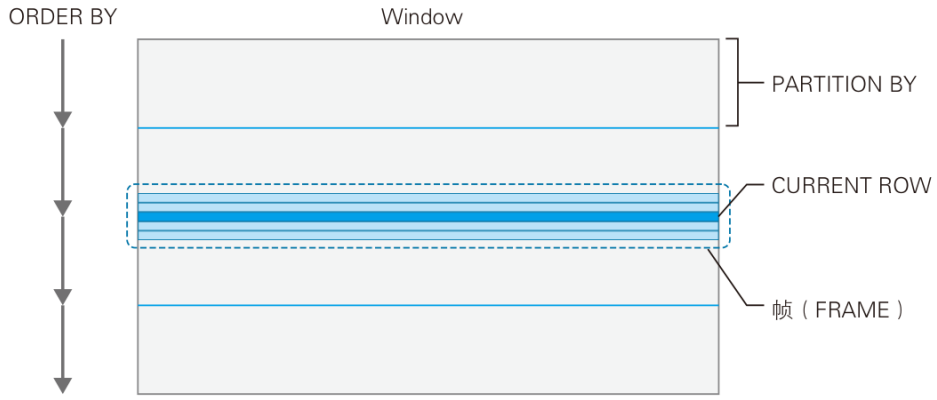

图 1.1: 一张图看懂窗口函数

- RANGE：按行值设置移动单位。使用 ORDER BY 子句来指定基准列
- n PRECEDING：仅向后（行号较大的方向）移动n行，n为正整数
- n FOLLOWING：仅向后（行号较大的方向）移动n行，n为正整数
- UNBOUNDED PRECEDING：一直移动到最前面
- UNBOUNDED FOLLOWING：一直移动到最后面
- CURRENT ROW：当前行

1.4 自连接的用法

1.4.1 可重排列、排列、组合

一种是有顺序的有序对（ordered pair），另一种是无顺序的无序对（unordered pair）。有序对用尖括号括起来，无序对用花括号括起来。

交叉连接的特征是没有连接条件，因为交叉连接是通过遍历两张表来列举出所有记录的所有组合的。

在 SQL 里，只要被赋予了不同的名称，即便是相同的表也应该当作不同的表（集合）来对待。

1.5 三值逻辑和 NULL

大多数编程语言都是基于二值逻辑的，即逻辑真值只有真和假两个。而 SQL 语言则采用一种特别的逻辑体系——三值逻辑，即逻辑真值除了真和假，还有第三个值“不确定”。

1.5.1 理论篇

两种NULL、三值逻辑还是四值逻辑

两种 NULL 分别指的是“ 未知”（unknown）和“ 不适用”（not applicable, inapplicable）。以“不知道戴墨镜的人眼睛是什么颜色”这种情况为例，这个人的眼睛肯定是有颜色的，但是如果他不摘掉眼镜，别人就不知道他的眼睛是什么颜色。这就叫作未知。而“不知道冰箱的眼睛是什么颜色”则属于“不适用”。因为冰箱根本就没有眼睛，所以“眼睛的颜色”这一属性并不适用于冰箱。

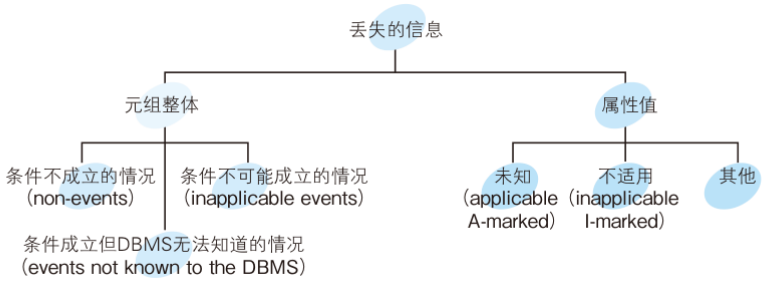

图 1.2: 关系数据库中“丢失的信息”的分类

为什么必须写成“IS NULL”，而不是“= NULL”

对 NULL 使用比较谓词后得到的结果总是 unknown。

为什么对 NULL 使用比较谓词后得到的结果永远不可能为真呢？这是因为，NULL 既不是值也不是变量。NULL 只是一个表示“没有值”的标记，而比较谓词只适用于值。因此，对并非值的 NULL 使用比较谓词本来就是没有意义的。

unknown、第三个真值

真值 unknown 和作为 NULL 的一种的 UNKNOWN（未知）是不同的东西。前者是明确的布尔型的真值，后者既不是值也不是变量。

请注意这三个真值之间有下面这样的优先级顺序。

- AND 的情况： $false > unknown > true$
- OR 的情况： $true > unknown > false$

优先级高的真值会决定计算结果。例如 true AND unknown，因为unknown 的优先级更高，所以结果是 unknown。而 true OR unknown 的话，因为 true 优先级更高，所以结果是 true。记住这个顺序后就能更方便地进行三值逻辑运算了。特别需要记住的是，当 AND 运算中包含 unknown 时，结果肯定不会是 true（反之，如果 AND 运算结果为 true，则参与运算的双方必须都为 true）。

1.5.2 实践篇

比较谓词和NULL(1)：排中律不成立

把命题和它的否命题通过“或者”连接而成的命题全都是真命题”这个命题在二值逻辑中被称为排中律（Law of Excluded Middle）。顾名思义，排中律就是指不认可中间状态，对命题真伪的判定黑白分明，是古典逻辑学的重要原理。“是否承认这一原理”被认为是古典逻辑学和非古典逻辑学的分界线。遗憾的是，在 SQL 的世界里，排中律是不成立的。

比较谓词和NULL(2)：CASE表达式和NULL

CASE表达式的判断方法和WHERE子句一样，只认可真值为TRUE的条件。