中篇的重点在于，在复杂情况下使用表表达式的查询，尤其是公用表表达式(CTE)，也就是非常方便的WITH AS XXX的应用，在SQL代码，这种方式至少可以提高一倍的工作效率。此外开窗函数ROW\_NUMBER的使用也使得数据库分页变得异常的容易，其他的一些特性使用相对较少，在需要时再查阅即可。

表表达式Table Expression是一种命名的查询表达式，代表一个有效的关系表与其他表的使用类似。SQL Server支持4种类型的表表达式：派生表、公用表表达式、视图等。

* 派生表

派生表也称为子查询表，非常的常见，之前介绍相关子查询时那些命名了的外部表均是表表达式。表表达式并没有任何的物理实例化，其优势在于使得代码逻辑清晰并可重用，但对性能并无影响。

|  |
| --- |
| 获取处理订单数超过100的订单年度及其客户数量：SELECT \* FROM (SELECT orderyear, COUNT(DISTINCT custid)) AS numcusts  FROM (SELECT YEAR(orderdate) AS orderyear, custid FROM sales.[order]) AS D1 GROUP BY orderyear) AS D2 WHERE numcusts > 100 |

* 公用表表达式CTE

其是T-SQL提供的一种表表达式的增强形式，使用起来非常的便捷方便（重用性很强），z而且代码非常的清晰，在数据库查询分页等场景下和开窗函数ROW\_NUMBER()配合的很好,这儿将之前介绍的派生表转化为CTE的形式。

|  |  |
| --- | --- |
| 嵌套的CTE | WITH D1 AS (  SELECT YEAR(orderdate) AS orderyear, custid FROM sales.[order] GROUP BY orderyear  ), D2 AS(  SELECT orderyear, COUNT(DISTINCT custid)) AS numcusts FROM D1  )  SELECT \* FROM D2 WHERE numcusts > 70 |
| 递归的CTE | 这个比较有意思，比如想在员工表中获取当前雇员的最大BOSS时很有效哦  WITH empsCTE AS(  SELECT \* FROM hr.employee WHERE empid = 6 --定位点元素  UNION ALL  SELECT c.\* FROM empsCTE AS p JOIN hr.employee AS c ON c.empid = p.manageid --递归元素  )  SELECT \* FROM empsCTE WHERE manageid IS NULL |

* 视图和内嵌表值函数（参数化视图）

|  |  |
| --- | --- |
| 视图 | IF OBJECT\_ID('sale.ChinaCusts') IS NOT NULL  DROP VIEW sale.ChinaCusts  GO  CREATE VIEW sale.ChinaCusts AS  SELECT \* FROM sale.Customer WHERE country = 'China' |
| 内嵌表值函数 | IF OBJECT\_ID('dbo.GetOrderByUID') IS NOT NULL  DROP FUNCTION dbo.GetOrderByUID  GO  CREATE FUNCTION dbo.GetOrderByUID  (@uid AS INT) **RETURNS** TABLE  AS  RETURN  SELECT \* FROM sales.[order] WHERE uid = @uid;  GO  SELECT \* FROM dbo.GetOrderByUID(8888) AS O; |

* APPLY操作符

该运算符也是一个表运算符，其支持CROSS APPLY和OUTER APPLY两种类型。其对两个输入表进行操作，右侧表往往是是一个派生表或者内联的TVF。其逻辑查询处理阶段将右侧表应用到左侧表的每一行，并生成组合的结果集。它**与JOIN操作符最大的不同**是右侧的表可以引用左侧表中的属性，例子如下。

|  |
| --- |
| 返回每个客户3个最近的订单：  SELECT c.custid, a.orderid, a.orderdate  FROM sales.customer as c CROSS[OUTER] APPLY  (SELECT TOP(3) orderid, empid, orderdate, requiredate FROM sales.[order] AS o WHERE o.custid = c.custid ORDER BY orderdate DESC, orderid DESC) AS a |

当使用CROSS APPLY操作符时会将orderid为空列去除，而OUTER APPLY则会在第二个逻辑阶段把其添加上，和外联接操作类似。

T-SQL支持集合运算符，除了常见UNION还支持INTERSECT和EXCEPT，也就是并集、交集和差集，其优先级顺序是INTERSECT > UNION = EXCEPT。需要注意的一点是，集合操作符默认认为两个NULL值是相等的，而不是之前逻辑操作符中提到的UNKNOWN。可能你会说使用外联接或者EXISTS运算符也可以达到相似效果，并在存在NULL比较的情况下必须添加相应处理代码，使用集合操作符可以简化SQL代码。

集合操作默认都存在一个隐式去除重复（即包含DISDINCT）的行为，只有UNION ALL支持重复数据。这儿补充一个关于集合概念，集合指不包含重复数据的集合，包含重复数据的情况我们称之为多元集合。在对两个（或多个）查询结果集进行集合操作时，需要注意其中的查询并不支持ORDER BY操作，如果还是需要这样的功能可以使用外部的ORDER BY或者是使用TOP等操作符将返回的游标转化为结果集。

集合操作符涉及的查询应该有相同列数，并对应列具有兼容类型（即低级别数据可以隐式的转化为高级别数据，如int->bigint），查询的列名称由第一次查询决定（在其中设置列别名）。

|  |  |
| --- | --- |
| 元数据查询类型 | 解释与示例 |
| UNION [ALL], INTERSECT, EXCEPT | SELECT country, region, city FROM address UNION SELECT country, region, city FROM user order by country |
| 复杂情况 | 对前置查询进行复杂操作，获取1、6号员工最近的2个订单,使用表表达式：  SELECT empid, orderid, orderdate FROM (SELECT TOP 2 empid, orderid, orderdate FROM [order] WHERE empid = 1 ORDER BY orderdate DESC) AS O1  UNION ALL  SELECT empid, orderid, orderdate FROM (SELECT TOP 2 empid, orderid, orderdate FROM [order] WHERE empid = 6 ORDER BY orderdate DESC) AS O2 |
| INTERSECT[EXCEPT] ALL的替代方案 | 实际SQL SERVER还不支持这种类型的操作，理解起来有点复杂，简单来说就是如果我的子查询A, B都有重复数据，一个是3条，一个是5条， 那么其INTERSECT ALL操作结果应该为3条，EXCEPT ALL的结果是2条。代码如下，重点是熟悉开窗函数的使用。  SELECT row\_number() OVER(PARTITION BY country, region, city ORDER BY (SELECT 0)) AS rownum, country, region, city FROM address  INTERSECT  SELECT row\_number() OVER(PARTITION BY country, region, city ORDER BY (SELECT 0)) AS rownum, country, region, city FROM user  这儿注意的是**ORDER BY (SELECT 0)**的用法，表示告诉系统不用排序的意思，减少不必要的开销。 |

这部分内容主要涉及T-SQL自身的一些新特性，例如开窗函数、透视数据等概念，相对来说比以前的内容难理解一些，不过经常几次简单的实践，你会发现它的强大和有效。

* 开窗函数

其根据基础查询的行子集计算，为子集中每行计算一个标量结果值，行子集被称为“窗口”，通过OVER字句进行相关操作，简单来说以前对分组查询操作GROUP BY的粒度仅限于一个聚合函数（子查询操作也类似），比如SUM（Amount），但现在想对分组内的行记录进行排序，这个更小的操作粒度在过去的SQL中是难以实现的，这是开窗函数却可以完成这部分的工作。常见的分组查询实际在查询中定义集合或组，因此在查询中的所有计算都要在这些组中完成，还记得那个逻辑顺序吧，GROUP BY是在SELECT之前的，因此一旦分组后，自然的就丢失了很多细节信息，但现在开窗函数是在SELECT字句阶段，那么也就是说所有的信息仍然都在，可以支持各种细粒度的操作。此外，开窗函数能够定义顺序，并不会和显示数据时的排序混淆。

|  |
| --- |
| 计算每个雇员每月的销售总计值：  SELECT empid, ordermonth, val, SUM(val) OVER (PARTITION BY empid ORDER BY ordermonth ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS runval FROM Sales.EmpOrders |

以上的窗口函数包括三个部分：分区、排序和框架。

分区字句，PARTITION BY：限定聚合函数运算的行子集，比如这个用empid分区，那么每个窗口自会包含该empid的计算（类似一个分组子集）。

顺序字句，ORDER BY：定义窗口中的排序，但不要和显示排序混淆，窗口排序是针对之后的窗口框架的，无论如何不要忘记字句的逻辑处理顺序，外部的ORDER BY字句是在SELECT字句后的。

框架字句，ROWS BETWEEN <top delimiter> AND <bottom delimiter>：进一步筛选之前的行子集（类似在子集中使用TOP操作），这儿的UNBOUNDED PRECEDING表示分区开始，CURRENT ROW表示当前行，使用UNBOUNDED FOLLOWING表示分区中的最后一行。

接下来介绍三类开窗函数，其中排序和聚合使用的场景比较多。

|  |  |
| --- | --- |
| 开窗函数类型 | 解释与示例 |
| 排名开窗函数 | 其中包含4种类型的排名函数，ROW\_NUMBER()、RANK()、DENSE\_RANK()、NTILE()，最常用的是ROW\_NUMBER,介绍一个分页场景  WITH CTE AS(  SELECT ROW\_NUMBER() OVER(ORDER BY custid) AS rownum, \* FROM Sales.Customers)  SELECT \* FROM CTE WHERE rownum > 10 AND rownum <= 20  接下来介绍一个分区内排序，要求选取每个雇员最大的3单金额及其排名  WITH CTE AS(  SELECT ROW\_NUMBER() OVER(PARTITION BY empid ORDER BY freight DESC) AS rownum\_ingroup, \* FROM Sales.Orders)  SELECT empid, freight, rownum\_ingroup FROM CTE WHERE rownum\_ingroup >= 1 AND rownum\_ingroup <= 3 |
| 偏移开窗函数 | 涉及LAG、LEAD、FIRST\_VALUE、LAST\_VALUE四个函数，这儿就介绍LEG和LEAD，表示当前记录的前一个记录和后一个记录，记得在上篇的子查询有写过一种“小于该值的最大值”的方式，这儿使用函数更加的简单。  SELECT orderid, freight, LAG(freight) OVER(ORDER BY orderid) AS pre\_freight, LEAD(freight) OVER(ORDER BY orderid) AS next\_freight FROM Sales.Orders  这儿比较奇葩的是LAG用于获取前一条记录，LEAD获取后一条记录，不得不说设计的小伙伴那天“脑袋不小心被门夹了下”，哈哈 |
| 聚合开窗函数 | 看到之后的例子，你会感觉开窗函数和人类的自然语言很像，获取每个订单、所有订单的运费总和  SELECT orderid, freight, SUM(freight) OVER() AS freightTotal FROM Sales.Orders |

* 透视和逆透视数据

透视实际上就是常说的“行转列”，而逆透视就是常说的“列转行”，由于这种操作实际上已有标准SQL的解决方案，不过很复杂和繁琐，这儿将SQL标准的解决方案和PIVOT、UNPIVOT函数的解决方案都描述出来。

|  |  |
| --- | --- |
| 透视/逆透视解决方案 | 解释与示例 |
| 标准透视 | 相信大家都很熟悉这种写法，因为面试中经常问到  SELECT empid, SUM(CASE WHEN custid = 'A' THEN qty END) AS A,  SUM(CASE WHEN custid = 'B' THEN qty END) AS B,  SUM(CASE WHEN custid = 'C' THEN qty END) AS C,  SUM(CASE WHEN custid = 'D' THEN qty END) AS D  FROM dbo.orders  GROUP BY empid;  这儿需要强调的重点是这个解决方案其实涉及3个阶段：第一个阶段为GROUP BY empid分组阶段；第二阶段为扩展阶段通过在SELECT字句中使用针对目标列的CASE表达式；最后一个阶段聚合阶段通过对每个CASE表达式结果聚合，例如SUM。 |
| PIVOT透视 | PIVOT实际是一个表运算符，包含分组、扩展、聚合三个逻辑阶段  SELECT empid, A, B, C, D  FROM ( SELECT empid, custid, qty FROM dbo.Orders) AS D PIVOT(SUM(qty) FOR custid IN (A, B, C, D)) AS P  以上可以发现子查询D中，包含empid、custid、qty三个属性，其中custid作为分组属性,qty作为聚合属性，那么剩下的empid就是扩展属性（不显示的指出但可以推算出） |
| 标准逆透视 | WITH CTE AS(  SELECT empid, custid, CASE custid WHEN 'A' THEN A WHEN 'B' THEN B WHEN 'C' THEN C END AS qty  FROM dbo.EmpCustOrders CROSS JOIN (VALUES('A'), ('B'), ('C'), ('D')) AS Custs(custid) )  SELECT \* FROM CTE WHERE qty IS NOT NULL  逆透视包括也包括三个逻辑阶段：第一阶段需要通过交叉联接生成每一列对应的一个副本；第二阶段通过CASE运算符生成列(qty)；最后一个阶段通过去qty IS NOT NULL删除不相关的交叉点，这一点一定不能忘了。 |
| UNPIVOT逆透视 | SELECT empid, custid, qty FROM dbo.EmpCustOrders UNPIVOT(qty FOR custid IN(A, B, C, D)) AS U ，有没有觉得超简单？ |

* 分组集

分组集就是一个属性集，分组GROUP BY字句只支持在一个查询中使用一种分组方式，如果需要多种分组的结果就需要通过UNION ALL将多个分组聚合起来，为了字段对应，需要为部分列设置NULL占位符。这部分的使用场景主要是在报表分析中，分组集提供4类操作符用于增强原有的GROUP BY字句，这儿就介绍GROUPING SETS操作符，CUBE和ROLLUP是对它的简化，可以通过语义理解，CUBE是立方即包含提供的分组属性的所有组合，ROLLUP是归纳，按照层次对分组属性进行组合，最后的GROUPING和GROUPING\_ID是对分组的标识。

|  |  |
| --- | --- |
| GROUPING SETS | SELECT empid, custid, SUM(qty) AS sumqty  FROM dbo.Orders GROUP BY GROUPING SETS((empid, custid), (empid), (custid), ()); |

最后推荐一个学习T-SQL的网站，<http://tsql.solidq.com/>，有空可以去看看，有英文原版的学习视频和资料。

参考资料：

1. (美)本咁. SQL Server 2012 T-SQL基础教程[M]. 北京:人民邮电出版社, 2013.