在当今这个多种不同数据库混用，各种不同语言不同框架融合的年代（一切为了降低成本并高效的提供服务），知识点多如牛毛。虽然大部分SQL脚本可以使用标准SQL来写，但在实际中，效率就是一切，因而每种不同厂商的SQL新特性有时还是会用到，这部分内容更是让人抓瞎，常常会由于一些很简单的问题花很久来搜索准确答案。赶脚俺弱小的智力已经完全无法记清楚常见的命令了，即使是用的最熟悉的T-SQL（SQL Server），因此决定将最常见且有效的T-SQL做个简单的总结，便于开发时的查找。实话实说，现在开发中较少写SQL了，但有时需要给测试团队提供一些便利还是需要的。

预祝大家新年快乐，万事如意！ I believe: 万丈高楼平地起，大海无边百川融。☺

这部分中重要的概念就是要弄清楚SQL语句具体的执行顺序，记得在南京做一个短期培训讲师期间，就发现这部分是一个很容易被忽视的基础，一旦弄清这部分内容，基本的标准SQL的编写基本上就没有很大问题了。此外关于SQL的一个非常关键的概念是，尽可能的使SQL语句进行的是整体的集合操作，而不是类似游标的循环迭代操作，这一点也是SQL优化的一个核心概念。

|  |  |
| --- | --- |
| 语句 | 执行顺序 |
| SELECT empid, YEAR(orderdate) AS orderyear, COUNT(\*) AS numorders  FROM Sales.Orders  WHERE custid = 71  GROUP BY empid, YEAR(orderdate)  HAVING COUNT(\*) > 1  ORDER BY empid, orderyear [ASC, DESC] | 1. **FROM**，从Sales.Orders表查询行 2. **WHERE**， 仅筛选ID为71的客户 3. **GROUP BY**，按固原ID和订单年度分组 4. **HAVING**， 筛选大于1个订单的组 5. **SELECT**，返回每组成员ID、订单年度、订单数量 6. **ORDER BY**，按固原ID和订单顺序排序输出行 |

不知道这儿的执行顺序和你心中的是否相同，记得了解到这部分知识时，自己也花了很久去理解， 不过从形式上可以看到实际的执行顺序很像LINQ，有木有？SQL只所以语句顺序和实际执行顺序不同是因为SQL设计师将该高级语言作为声明式语言来定义的，“可以按照类似英语的方法提供自己的请求“。之所以说这部分重要，不知道大家遇到过自己给字段起的别名在where中不能使用的情况没有，那是因而where执行时，select还未执行，那么select中给字段其的别名还不存在好，但在order by字句中就可以正常使用。接下来，补充说明一下以上六个字句中的相关知识。

**FROM字句**：在From字句中的对象中需要附加上schema架构限定，如dbo.Sales， hr.Employee等。

**WHERE字句**：该字句中字段的选择对于查询性能影响很大，如果符合索引（包括组合索引，需要正确的顺序）条件，那么查询就会通过索引而不是全表扫描。例如建立的组合索引为(name, time),那么如果查询中使用where time =xx and name = xx会造成索引不起作用，而造成全表扫描，当然由于内置查询优化器的存在，实际的查询可能与教科书上说的不同， 一定要实际查看执行计划，一起以事实说话。在实际项目中，数据库的设计需要保证基本不犯明显的错误即可，其他的到出现性能问题时通过查询计划和查询统计信息才去优化，不用过度设计，因为数据量没变化一个量级可能调优的方式就会出现不同。才外，需要记住，在TSQL中使用三值谓词逻辑，逻辑表达式可以计算为TRUE、FALSE和UNKNOWN，而如果数据字段为空，需要使用IS [NOT] NULL判断。

**GROUP BY字句：**当涉及分组时，其后续的所有操作都是对组的操作而不是对单个行的操作，每组均是一个单个行，这些操作中表达式需要保证返回一个标量。不参与到group by中的字段仅允许作为一个聚合函数的输入，如COUNT、SUM等。注意，除了Count(\*)外，所有的聚合函数忽略NULL标记，DISTINCT可以包含在聚合函数中，针对不重复且有值的项。

**HAVING字句：**可以指定一个谓词来筛选组而不是单个的行，比如使用集合函数count(\*)>1表示筛选组成员大于1的组。

**SELECT字句：**指定返回到查询结果表中列的地方，可以包含表达式，推荐给表达式创建一个易懂的别名，比如Year(orderdate) AS OrderYear，尤其是新增一些与列无关的表达式，如current\_timestamp()函数，使关系模型变得完善。这人再次提及SELECT字句中别名的使用范围，只能是SELECT字句执行之后的部分，也就是Order by字句。此外，有一点曾经困扰了我很久，就是如果我在where字句中使用YEAR(orderdate)，还在select中使用YEAR(orderdate)，那样不是重复计算了？其实，**SQL SERVER能够识别查询中重复使用的相同表达式**，也就是说在一个查询，出现多次相同的表达式，实际上只会运算一次，简直赞赞哒。不过在同层中使用别名还是没有被支持的，例如，

~~SELECT YEAR（orderdata）AS orderYear, orderYear + 1 AS nextyear~~是不正确的。常见的，我们在一般的查询中，比如检验数据等，是推荐使用SELECT \*，包括加上top 1000的，但在项目代码中，是严禁这样的操作的。

补充一点关系代数的知识，我们知道在关系模型中，所有操作均基于关系代数，并且操作结果是一个关系集合，但实际上我们返回的结果集还是会出现重复行的情况，不过可以通过**DISTINCT**关键字删除重复行。

**ORDER BY字句：**按序输出行，需要理解的是，在SQL中，表中没有确定的顺序，表假定为一个集合，集合是没有顺序（这个观念如果是半路出家，需要很久才能真正理会的到）。因此，Order by之后的有序结果，其实失去表资格，一般将这种结果称之为游标，“一个具有确定行顺序的非关系型结果”，这部分概念在之后的还会有介绍。此外，该字句中可以使用不在SELECT列表中的字段排序，但如果使用了DISTINCT关键字，则必须使用SELECT列表中的列，否则由于单个结果行可能代表多个原行，造成排序的不清晰。

这儿有点需要补充的是，在同样的ORDER BY条件下，可能会得到不一样结果的问题，这个其实和数据结构中排序的概念一样。在某个条件（比如order by日期）下，有多个符合条件的记录时，这几个结果集的顺序是不一定的（已实际访问的物理记录行的顺序为准），属于**不稳定排序**。常见排序算法中，[快速排序](http://baike.baidu.com/view/115472.htm)、[希尔排序](http://baike.baidu.com/view/178698.htm)、[堆排序](http://baike.baidu.com/view/157305.htm)、[直接选择排序](http://baike.baidu.com/view/3632419.htm)不是稳定的[排序算法](http://baike.baidu.com/view/297739.htm)，而[基数排序](http://baike.baidu.com/view/1170573.htm)、[冒泡排序](http://baike.baidu.com/view/254413.htm)、[直接插入排序](http://baike.baidu.com/view/1193395.htm)、折半插入排序、[归并排序](http://baike.baidu.com/view/90797.htm)是稳定的排序算法。那么有没有稳定的情况呢，那么就需要排序条件中的每一项都是独一无二的，比如是主键列，唯一列，这种属性也称之为排序的**决胜属性(tiebreaker)**。

此外注意在SQL的关键字和系统函数名使用大写，涉及多表查询时需要给表起别名方便理解。以上是最核心的部分，接下来以列表的形式阐述与基本查询相关的SQL关键字。

|  |  |
| --- | --- |
| 关键字 | 解释与示例 |
| TOP | T-SQL特有功能，用于限制查询返回的行数或行的百分比  获取前5行记录：SELECT TOP 5 userid FROM HR.Employee ORDER BY userid  获取前5%的记录：SELECT TOP 5 PERCENT userid FROM HR.Employee ORDER BY userid  这儿其实隐含了一个问题，就是这儿**TOP返回的结果是表结果还是游标**，因为之前有提到，使用order by之后返回的游标，这个问问的解释会放到表表达式部分。 |
| OFFSET-FETCH | 这是标准SQL的选取行数的语法，并且支持跳过功能，免得我们需要使用开窗函数或者两个TOP取交来实现该功能，等价于C#中 XXX.Skip(m).Take(n)  查询第51到75条记录：SELECT userid FROM HR.Employee ORDER BY userid **OFFSET 50 ROWS FETCH NEXT 25 ROWS ONLY**  Tip:ROW和ROWS关键字等价，该关键字不支持PERCENT和WITH TIES选项 |
| ROW\_NUMBER | 是一种开窗函数，理解起来有点困难，其实就是对分组中的数据做更加细粒度的操作，方便实现复杂的查询，尤其是为报告服务(SSAS报表分析服务)。为了之后能更好的理解该知识点，提前拿出来给大家见见面，本文下篇还会具体介绍  开窗函数的定义：对于一个查询中的每一行，按行的窗口组进行运算，并计算一个标量结果值，行的窗口使用OVER字句定义  SELECT userid, **ROW\_NUMBER() OVER(PARTITION BY** userid ORDER BY name) AS rownum  这儿的PARTITION分区其实就是分组条件，ROW\_NUMBER函数实际用于对分组后小组内成员标上行号，同时OVER窗中的ORDER BY是组内的排序，规则和正常排序一致 |

* 谓词、常见运算符和系统函数

谓词（Predicate，这个委托熟悉不？）主要出现在WHERE、HAVING查询筛选中，包括TRUE、FALSE和UNKNOWN中逻辑结果，这儿一定不能忘记UNKNOWN未知结果这种情况，比如说让两个NULL作比较等，接下来用一个简单的表格展示。

|  |  |
| --- | --- |
| 关键字 | 解释与示例 |
| BETWEEN, IN, LIKE | BETWEEN AND表示属于什么之间；IN表示在枚举出来的几个值中；LIKE可以使用%作为万能替代符，主要注意的LIKE中默认使用的Unicode的字符类型，并且在使用LIKE关键字时一定要谨慎，会造成很大的查询消耗，如果实在需要大量字符串的查询，考虑使用全文检索或选用其他类型数据库等解决方案 |
| NOT, AND, OR | 分别表示非、与、或的逻辑，优先级依次递减 |
| % | 取余操作符或是之前介绍的万能占位符 |
| CAST(col1 AS NUMERIC(12, 2)) | 在数值运算时，如果出现两个整型相除，需要修改其类型避免丢失小数点后位数 |
| CASE | CASE是一个标量表达式，返回一个基于条件逻辑的值，需要注意CASE不是语句不能用于控制逻辑(比如IF ELSE)，实际中，CASE的使用场景还是很多的,比如行列转换等，才外，ORANGE有一个叫做的decade的简化版操作符。  尤其需要注意的是，CASE具有“简单“和”搜索“两种格式，后者非常的灵活  简单格式： SELECT studentid, **CASE score WHEN 59 THEN ‘Fail’ WHEN 60 THEN ‘Alive’** FROM dbo.testScore  搜索格式： SELECT studentid, **CASE WHEN score < 60 THEN ‘Bad’ WHEN score > 60 THEN ‘Good’ELSE ‘UNKNOWN’** FROM dbo.testScore  此外，在SQL SERVER2012中还增加几个简化操作符, **COALESCE** ， CHOOSE，ISNULL等，比如ISNULL(col1, ‘’)~~表示Nullable<T> ?? ‘’~~，若col1不为空就取其值，为空就是’’空字符串，不过均不推荐使用。 |
| NULL | NULL标记的理解在SQL中非常重要，很多细微的SQL错误都来之于此。其根源仍然是之前提到的3值逻辑，NULL标记表示不知道是什么值（在现实生活中，就像登记时缺失了），它与除了IS [NOT] NULL逻辑操作以外的逻辑运算结果均是UNKNOWN。  为了更好的理解应用这个概念，来看看接下来的例子  SELECT COUNT(\*) FROM Address WHERE region <> ‘Beijing’, 假设该表中100条记录，10条的region是Beijing，20条的region为NULL，70为其他城市，那么这个查询的结果将是70，而不是我们想要的90，修改查询如下即可。  SELECT COUNT(\*) FROM Address WHERE region <> ‘Beijing’ **OR region IS NULL** |

* 同时操作

同时操作(all-at-once operations)表示出现在同一逻辑处理阶段的所有表达式在同一时间点进行逻辑运算，即并行运算，这个概念是之前查询顺序概念的补充，之前讲的纵向的顺序，这儿讲的是横向的并行。是不是觉得很拗口，其实重点就是SQL在同一层中运算的顺序不固定，所以之后运算一定不能依赖于之前的运算，并且不支持C#等常见语言中的短路: if(result != null && result == true)，前者失败后者不运算，接下举两个**错误**的例子。

|  |
| --- |
| SELECT orderID, YEAR(orderdate) AS orderyear, orderyear + 1 AS nextyear FROM [order] |
| SELECT num1, num2 FROM dbo.tableB WHERE num1 <> 0 AND num2/num1 > 5 |

那么不禁要问，SQL中能提供同层次的逻辑运算的解决方案么？其实是有的，一种技巧性的，一种是通过合理的规划。

|  |  |
| --- | --- |
| 技巧性 | 通过使用CASE表达式来实现，形式上有一些奇怪  SELECT num1, num2 FROM dbo.tableB CASE WHEN num1 = 0 THEN 0 WHEN num1/num2 > 2 THEN 1 ELSE 0 END = 1。 其中1表示true，0 表示false |
| 合理规划 | SELECT num1, num2 FROM dbo.tableB WHERE (num1 < 0 AND num2/num1 > 5) OR (num1 > 0 AND num2/num1 > 5) |

* 字符数据类型及其函数

最基础的字符类型包括 ASCII（American Standard Code for Information Interchange，这单词还是第一次认真看）和Unicode类型，在T-SQL中就是(CHAR, VARCHAR)和(NCHAR, NVARCHAR)。ASCII每个字符用一个字节存储，用’’单引号括起来，Unicode用2个字节存储，用N’’包起来。CHAR字符类型是固定大小的，效率高但空间浪费率高，VARCHAR灵活节省空间，有2个字节偏移数据，但在字段值变长时，可能出现行扩展导致分页等，更新效率较低。字符类型默认最长(max)为8000个字符，若超过使用LOB存储，放在行外，此外数据在压缩（Data Compression）是会有一些变化。

介绍一个不常见的知识点，排序规则，知道以下概念即可

|  |
| --- |
| 获取数据库支持的排序规则：SELECT \* FROM sys.fn\_helpcollations() |
| 筛选条件区分大小写; SELECT \* FROM user WHERE name COLLATE Latin1\_General\_CS\_AS = N‘xionger’ |

接下来是最重要的字符函数使用示例列表

|  |  |
| --- | --- |
| 关键字 | 解释与示例 |
| +, CONCAT | 连接字符串， SELECT firstname + lastname AS fullname FROM user，需要注意的是null与任何字符串连接操作的结果还是NULL |
| SUBSTRING | 获取world子串：SELECT SUBSTRING('hello world', 7, 5)，注意index从1开始，不是0 |
| LEFT, RIGHT | SUBSTRING的简化形式，获取字符串左边/右边指定的字符数，有个一个很经典的应用场景如下  对ID值补0操作：SELECT RIGHT('0000000000' + CAST(1973 AS char(10)), 10)， 1973可以为任何的类似ID的变量 |
| LEN, DATALENGTH | 前者返回字符长度，后者返回字节长度，SELECT LEN('abcde'), LEN(N'abcde'), DATALENGTH(N'abcde')的结果为： 5, 5, 10 |
| CHARINDEX, PATINDEX | 前者返回子串第一次出现的位置，后者返回匹配的子串第一次出现的位置，SELECT CHARINDEX(' ', 'xiong er 1'), PATINDEX('%[1-9]', 'xiong er 1')，结果6, 10 |
| REPLACE, REPLICATE, STUFF | 分别是替换、复制和删除后新增，SELECT REPLACE('1e11', 'e', '1'), REPLICATE('0', 10), STUFF('1e11', 2, 2, '222')，结果为 1111, 0000000000, 12221 |
| LIKE | 包括几个常见的通配符，% 表示任意大小字符串 ，\_ 表示单个字符，以及其他常见正则表达式，如[ABC]、[A-Z]、[^1-9] |

* 时间日期数据类型及其函数

在T-SQL中，常见的时间类型仅仅包含DATETIME，其实记住这个基本上足够用了。其他的都是一些更高精度和便捷的选择，包括SMALLDATETIME,DATE, TIME, DATETIME2,DATETIMEOFFSET等，精度达到了ns纳秒级，需要时再查阅即可。需要提及的一点时，在SQL中经常使用字符串常量格式的日期实际上最终是通过一个隐式转化为变为DATETIME类型的，如ordedate = ‘20160203’等价于orderdate = CAST(‘20160203’AS DATETIME)，这儿的转化是基于当前会话的语言格式的，在实际中为了兼容，推荐使用与语言无关的常量格式： YYYYMMDD hh：mm：ss.nnn或YYYT-MM-DD。

接下来，将一个很容易忽视的知识点，筛选时间范围，涉及查询优化，比如我们想选择今年的全部订单，很自然的会想到如下SQL语句。

|  |  |
| --- | --- |
| 错误的方式 | SELECT \* FROM [order] WHERE YEAR(orderdate) = 2016 |
| 正确的方式 | SELECT \* FROM [order] WHERE **orderdate >= '20160101' AND orderdate < '20170101'** |

对于所有的查询条件，尽可能的不要在其上使用表达式，这样查询优化器更可能通过索引的方式查找，此外想说的是，查询的条件的顺序也很重要哦，其需要和你所建立组合索引的顺序一致。

时间日期函数看起来比较简单，但在实际的使用中，由于不同的时间格式，往往会让人非常的困扰，毕竟那么多的API使用起来选择比较多，这儿将最常见的罗列了出来。

|  |  |
| --- | --- |
| 关键字 | 解释与示例 |
| GETDATE(), CURRENT\_TIMESTAMP | 均是获得当前时间，后者遵循ANSI SQL规范 |
| CAST, CONVERT, PARSE | 将输入值转换为目标类型，CAST(value AS datatype)最简单，CONVERT（datatype, value [,style\_number]）足够完美，PARSE(value AS datatype [USING culture])  在SQLSERVER2008之后版本获取时间、日期方式：CAST(GETDATE() AS DATE), CAST(GETDATE() AS TIME)  老版本兼容方式：**SELECT CONVERT(CHAR(8),GETDATE(),112), SELECT CONVERT(CHAR(12),GETDATE(),114)**  此外如果想的到今天的午夜时间可以用：**SELECT CAST(CONVERT(CHAR(8),GETDATE(),112) AS DATETIME)** |
| DATEADD | 增加一年, SELECT DATEADD(year, 1, CURRENT\_TIMESTAMP), month, day,second, quarter季度, week星期等类似 |
| DATEDIFF | 返回两个日期间的差值，SELECT DATEDIFF(day, '20160101', CURRENT\_TIMESTAMP)  常见的，获得本月(年)的开始: **SELECT DATEADD(MONTH, DATEDIFF(month, '19000101', CURRENT\_TIMESTAMP), '19000101')**  获得本月(年)的结束: **SELECT DATEADD(day, -1, DATEADD(MONTH, DATEDIFF(month, '19000101', CURRENT\_TIMESTAMP) + 1, '19000101'))**  这部分如果和之前的筛选时间日期的知识点结合在一起就能写出非常灵活的SQL代码 |
| DATEPART, YEAR, MONTH, DAY | 获取时间日期中的部分整数，DATEPART(month, CURRENT\_TIMESTAMP)， 后面的3个函数是前面的简化版 |

* 基本元数据查询

对于.NET程序员来说，元数据这个概念一点也不陌生，这儿指的是数据库本身以及其中对象的结构信息，接下来介绍最简单的几种元数据的查询。

|  |  |
| --- | --- |
| 元数据查询类型 | 解释与示例 |
| 目录视图 | 获取表信息：SELECT SCHEMA\_NAME(schema\_id) AS table\_schema\_name, name AS table\_name FROM sys.tables  获取列信息：SELECT name AS columnName, TYPE\_NAME(system\_type\_id) AS columnType FROM SYS.columns WHERE object\_id = object\_id(N'dbo.tableA')  此外还有很多的目录信息都在sys这个schema之下，这儿只选择最常见的表和列 |
| 信息架构视图（推荐使用） | 这其实是前面目录视图的标准化版本，功能基本类似，相对更加的简洁，推荐使用这种方式查询，相关视图均在INFORMATION\_SCHEMA这一schema下  表信息： SELECT TABLE\_CATALOG, TABLE\_NAME, TABLE\_SCHEMA, TABLE\_TYPE FROM INFORMATION\_SCHEMA.TABLES  列信息：SELECT COLUMN\_NAME, DATA\_TYPE FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_NAME = 'tableA' AND TABLE\_SCHEMA = 'dbo' |
| 系统存储过程和函数 | 还可以通过存储过程来查询相关信息，比如EXEC sys.sp\_tables; EXEC sys.sp\_columns; EXEC sys.sp\_help @objname = N'dbo.tableA'等，这种方式不是特别推荐，但sp\_help在俺的SQL生涯中确实出现了很多次哦。此外SELECT SERVERPROPERTY('productlevel')可以查询当前数据库实例的版本，这不是重点，想说的是本机的查询结果是RTM，这表示Release to Manufacturing发布到生产制造，即正式版，有时遇到RTM、SP1等是注意理解。 |
| DBCC命令 | 查看数据库的隔离级别的信息：DBCC USEROPTIONS |

在介绍联接前先引出一个概念--**表运算符**，我们知道FROM字句是第一个被逻辑处理的字句，其中包含表信息，那么对表进行操作的运算符就是表运算符，其中本节要介绍的**JOIN**是最重要的，很多时候，工作中可能仅仅使用它就足够，此外还是APPLY、PIVOT、UNPIVOT等操作符，之后会介绍。其中JOIN操作符对两个输入表进行操作，类型包括交叉联接、内部联接和外部联接，它们之间的差别在于其逻辑查询处理阶段，这是本节的最需要理解的概念，是真正理解联接操作的基础，通过一个表格来做一个初步的了解（这部分初学时，很容易忽视，但非常重要）。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 逻辑查询阶段 | 笛卡尔乘积 | 筛选 | 添加外部行 | 示例 |
| 联接类型 |  |
| 交叉联接 | | Y | N | N | SELECT u.userid, s.studentid FROM user AS u CROSS JOIN student AS s |
| 内部联接 | | Y | Y | N | SELECT u.userid, s.studentid FROM user AS u  INNER JOIN student AS s ON u.name = s.name |
| 外部联接 | | Y | Y | Y | SELECT u.userid, s.studentid FROM user AS u INNER JOIN student AS s  LEFT JOIN student AS s ON u.name = s.name |

之前一直强调的逻辑查询阶段其实相对应与物理查询阶段的，由于数据库查询分析器的存在，有时看起来有性能问题的联接也能运行的很好，所以当遇到查询性能问题时，查看执行计划和分析统计数据非常的重要。

**交叉联接**：只包含笛卡尔乘积阶段，比如一张表A有m行，表B有n行，其结果集有m\*n行记录。该类型使用场景非常少，但其中有2个场景还是需要知道的。

|  |  |
| --- | --- |
| 自交叉联接 | SELECT e1.empid, e2.empid FROM hr.employee AS e1 CROSS JOIN hr.employee AS e2 |
| 生成数字表（有点像数学的辅助函数） | 首先在DB中创建一张包含1到10的数字表,之后通过这张表来构建1到1000的数字表  SELECT d3.digit \* 100 + d2.digit \* 10 + d1.digit + 1 AS n  FROM dbo.digits AS d1 CROSS JOIN dbo.digits AS d2 CROSS JOIN dbo.digits AS d3 ORDER BY n  这儿介绍这个的原因是，在实际工作中，为处理异构数据或者按指定格式呈现时，可能需要构建辅助表，埋下这样一个种子就好 |

**内部联接**：最常见和基础的联接方式，包含笛卡尔乘积和筛选两个步骤，相对复杂的情形包括复合联接、不等联接和多联接查询，如下表所示。

|  |  |
| --- | --- |
| 情形 | 解释与示例 |
| 复合联接 | 一般在查流水、履历时会遇到这样的场景，因为这时并没有一个唯一的主键标识，需要组合的候选键来查询  SELECT dbo.tableA AS t1 JOIN dbo.tableB AS t2 ON t1.col1 = t2.col1 AND t1.col2 = t2.col2 |
| 不等联接 | 用到不等联接的场景不算太多，一种比较有意思婚配的婚配场景，找到一组人中所有婚配组合（不重复，不并剔除自己和自己配对）  SELECT e1.empid, e2.empid FROM hr.employee AS e1 INNER JOIN hr.employee AS e2 ON e1.empid < e2.empid |
| 多连接查询 | SELECT \* FROM sales.[order] AS o INNER JOIN sales.orderdetail od ON o.orderid = od.orderid INNER JOIN sales.product p ON od.productid = p.id |
| 自联结 | 有一个比较典型的例子，就是在员工表找到自己的直属领导  SELECT e1.\*, e2.name AS manageName FROM hr.employee AS e1 INNER JOIN hr.employee AS e2 ON e1.manageid = e2.empid |

**外部联接**：除了包含内联接的两个逻辑处理阶段，还包含一个“添加外部行”的第三个阶段。外联接包含LEFT OUTER JOIN、RIGHT OUTER JOIN和FULL OUTER JOIN三种类型，分别表示左侧表为保留表、右侧表为保留表和两侧表均为保留表。这儿的**保留表**也就是我们常说的基准表，即其中的每一条记录实际上都会在最终的结果中显示出来，之后会通过一个内部行、外部行的概念对其进行阐述。在逻辑处理的第二阶段，也就是筛选阶段，将筛选出符合ON谓词的行，称之为内部行（由内部联接返回）。在以后的第三阶段，将识别出保留表中基于ON谓词未能与另一张表匹配的行，称之为外部行，此阶段会将这些外部行添加到之前的结果集中，在这些外部行中，其非保留表字段将使用NULL作为占位符。

接下来介绍外联接中容易忽视的知识难点，其与之前介绍的查询顺序密切相关，就是在查询中ON关键字和WHERE关键字的区别的问题，通过之前的介绍可知，ON关键字属于FROM字句，其作用时间在WHERE字句之前，也就是说WHERE操作是基于之前结果的，理清逻辑就好理解了。之前提到外联接查询的结果包含内部行和外部行，如果我们想进一步将外部行或内部行剔除，那么就可以使用WHERE进行筛选，需要注意非保留表中的属性一定要选择非NULL属性，这时的查询真正对应到NULL占位符（NULL有两种含义，一种是数据不存在，另一种是联接的占位符），如下所示。

|  |
| --- |
| 只获取外部行，即订单不存在的用户记录：SELECT c.custid, c.companyname FROM sale.customer AS c LEFT OUTER JOIN sale.order AS o on c.custid = o.custid WHERE o.orderid IS NULL |

最后，补充两个使用外联接容易出错的场景。第一个是在一个查询中同时包含内联接和外联接的情况，由于表运算符的处理是有逻辑顺序的（其他为同时操作，之前有介绍），因而不同的联接顺序可能造成不同的结果集，比如在使用LEFT JOIN之后使用INNER JOIN很有可能就会把添加上的外部行再次除去掉，一般推荐确定好保留表的先后顺序，均使用LEFT JOIN为宜。另外一个问题是在有外联接的情况下使用COUNT聚合操作符，会将外部行业作为计数目标，然而有时这是不合理的，此时需要使用指定的非空列，COUNT(col)。

SQL支持在查询内编写查询，外部查询返回结果集，内部查询的结果集被外部查询使用，称之为子查询。如果子查询不依赖其所属外部查询，称之为无关子查询，反之为相关子查询。子查询的返回值可以是一个标量、多个值和一张表。

* 无关子查询

|  |  |
| --- | --- |
| 标量子查询 | 获取当前最大订单的相关信息: SELECT \* FROM sale.[order] WHERE orderid = (SELECT MAX(o.orderid) FROM sale.[order] AS o) |
| 多值子查询 | 返回中国客户的订单: SELECT \* FROM sales.[order] WHERE custid in (SELECT C.custid FROM sales.customer AS c WHERE c.country = N'China') |

无关子查询最为简单常见，在之前的IN谓词查询中，可能会想到不是给custid加上DISCTINCT会更高效，实际上查询分析器会默认考虑删除重复记录，此外多值查询还有ALL、ANY和SOME关键字。

* 相关子查询

相关子查询引用的列位于外部查询中，该子查询依赖于外部查询，自身无法单独调用。

|  |
| --- |
| 获取每个用户返回最大订单ID的订单：SELECT \* FROM sales.[order] AS o1 WHERE o1.orderid = (SELECT MAX(o2.orderid) FROM sales.[order] AS o2 WHERE o2.uid = o1.uid) |

* EXISTS谓词

接受一个子查询作为输入，如果子查询返回任意行返回TRUE，否则FALSE，这也是SQL中少有的2值逻辑场景。此外，EXISTS在查询优化方面也有一定应用，在之后优化的专题中会涉及。

|  |
| --- |
| 查询没有下订单的美国客户: SELECT \* FROM sales.customer AS c WHERE country = ‘American’ AND NOT EXISTS (SELECT \* FROM sales.order AS o WHERE o.custid = c.custid) |

是不是觉得上面的部分很简单，那么接下来将带来两个相对复杂的查询场景。

|  |  |
| --- | --- |
| 返回前一个值 | 返回每个当前订单及其前一个订单ID的信息，前一个逻辑上等价于“小于当前值中最大的那个值”  SELECT orderid, orderdate, empid, custid, ( SELECT MAX(T2.orderid) FROM sales.[order] AS T2 WHERE T2.orderid < T1.orderid ) AS prevorderid  FROM sales.[order] AS T1 |
| 随时间累积值的聚合 | 返回每年的订单年度、数量和经过年度的运行总量  SELECT orderyear, qty, ( SELECT SUM(T2.qty) FROM sales.orderTotalByYear AS T2 WHERE T2.orderyear <= T1.orderyear ) AS runqty  FROM sales.orderTotalByYear AS T1 ORDER BY orderyear |

本节最后仍然是来介绍一些易错的查询场景，第一个是NULL故障，在查询中存在NOT IN (某个子查询)，如果这个子查询的结果集中存在NULL，则无论如何其外部查询的结果也是空结果集，仍然是3值逻辑的理解。第二是只要涉及多个表一定要习惯与使用别名，不然很容易出现因细微字母差异而造成的SQL语句错误。

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），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。 |

最后推荐一款数据库管理工具Navicat，支持SQL Server、Mysql和Oracle等主流数据库，不用切换不同的数据库管理工具，整体使用流畅性和用户体验也比默认的管理工具好一些哦，赞赞哒。

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.