**Over子句**

Over子句用于为行定义一个窗口（window），以便进行特定的运算。可以把行的窗口简单地认为是运算符的一个行的集合。例如聚合函数和排名函数都是可以支持Over子句的运算类型。由于Over为这些函数提供了一个行的接口，所以这些函数也被称之为开窗函数（window function）。

聚合开窗函数使用Over子句提供的窗口作为上下文，对窗口中的一组值进行操作，而不是使用Group By子句提供的上下文。这样就不必对数据进行分组，还能够在同一行中同时返回基础行的列和聚合列。

如果我们想对行进行限制或分区，则可以使用Partition By子句。例如，现在不想返回所有的行，只是想返回当前用户的总价格，则可以指定Sum(val) Over(Partition By custid)

Over子句也支持四种排名函数：Row\_Number(行号) ，Rank(排名)，DENSE\_RANK(密集排名)，以及NTILE.

Row\_Number函数用于为查询的结果集中的各行分配递增的序列号，其逻辑顺序通过Over子句中的Order By语句进行指定。

NTILE函数可以把结果中的行关联到组，并为每一行分配一个所属组的编号。NTILE接收一个表示组数量的输入参数，并要在Over子句中指定逻辑顺序。

**谓词**

T-SQL支持的谓词包括IN，Between和Like等。

IN这个谓词用于检查一个值是否与一组元素中的至少一个相等。

Between这个谓词用于检查一个值是否在指定范围内，包括两个指定的边界值。

Like这个谓词用于检查一个字符串值是否与指定模式匹配。

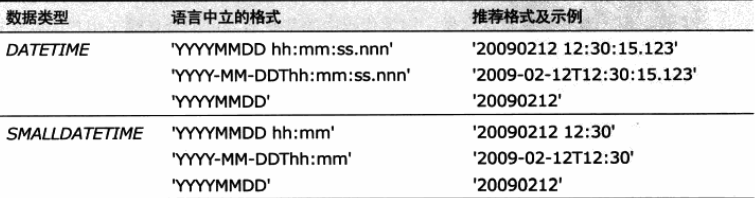
**NULL值**

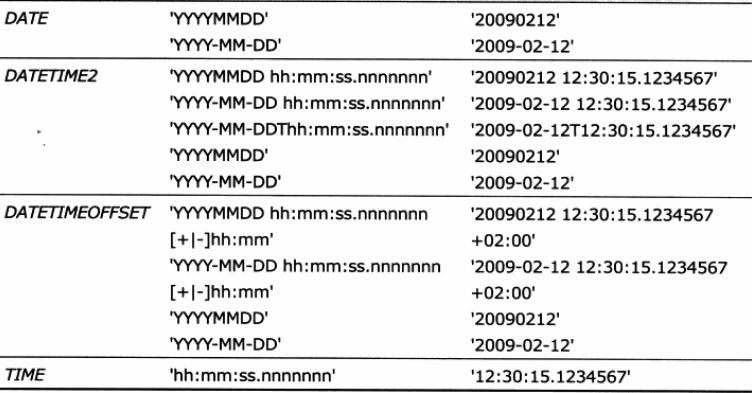
在where子句中，如果结果需要包含NULL，则必须指定：ISNULL(字段)。因为条件查询时会过滤掉UNKNOWN的值。

**日期格式和转换，比较**

**日期格式转换**

字符串转日期格式时，强烈推荐按照语言无关性的方式来编写日期和时间字符串文字。下面列举了每种日期和时间类型的语言无关的字符串格式：





SQL语句如下：

SELECT CAST('2018-01-23T10:23:01' AS DATETIME)

--2018-01-23 10:23:01.000

SELECT CAST('20181001' AS SMALLDATETIME)

--2018-10-01 00:00:00

SELECT CAST('20000901 07:30:00' AS DATETIME2)

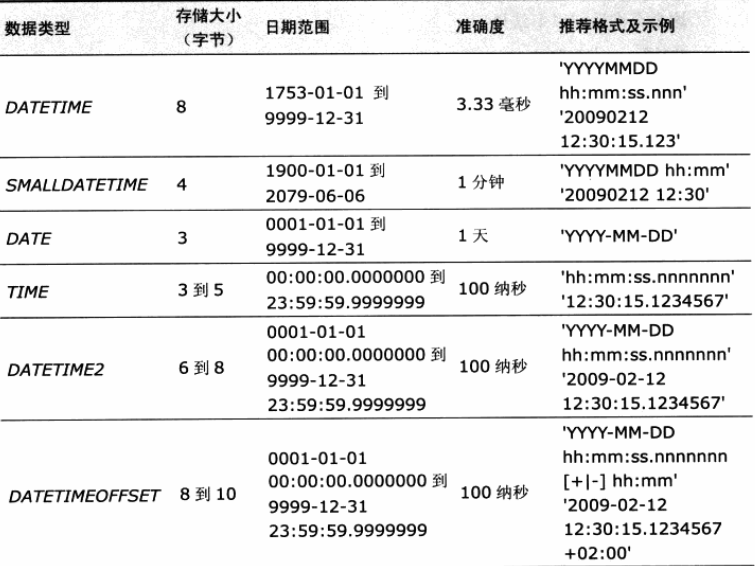
2000-09-01 07:30:00.0000000

SELECT CAST('19:45:00' AS TIME)

--19:45:00.0000000

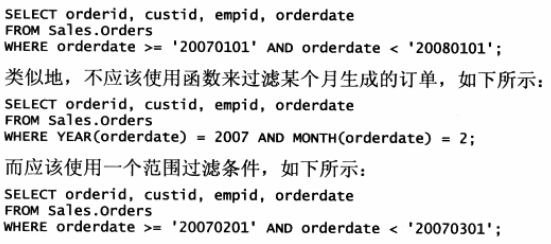
**日期类型**

SQLServer2008引入了新的日期类型：DateTime2，一种比DateTime具有更大的日期范围和更好精度的日期类型；以及DATETIMEOFFSET它具有一个时区的组成部分。



**过滤日期条件**

SQLS如果要过滤日期条件，比较自然的方法是调用Month和Year之类的函数。不过应该注意的一点是：在大多数情况下，当对过滤条件的列做了一些处理之后，就不能以有效的方式使用索引了。所以我们需要对谓词进行调整，以便对过滤条件中的列进行处理。如下所示：



**联接查询**

Join表操作符对两个输入表进行了操作。联结有三种基本类型：交叉连接，内连接，外链接。这三种连接的区别是它们采用的逻辑查询处理步骤各部相同，每种连接都有一套不同的步骤。交叉连接只有一个步骤—笛卡尔积；内连接有两个步骤—笛卡尔积，过滤；外链接有三个步骤—笛卡尔积，过滤，添加外部行。

**交叉连接**

交叉联接是一种最简单的联接，交叉联接只实现一个笛卡尔积即可。这一步就是对两张表进行操作，把它们联接起来，生成两者的笛卡尔积。也就是将一个输入表的每行和另一张表的每行进行匹配。如果一张表有n行，另一张表有m行，将得到m \* n行。

ANSI SQL-92：Cross Join

ANSI SQL-89：在两张表间放置一个逗号

**内连接**

内连接要应用两个逻辑查询处理步骤：它首先像交叉联接一样，对两个输入表进行笛卡尔积运算：然后根据用户指定的谓词对结果进行过滤。和交叉联接一样，内联接也有两种标准语法。

ANSI SQL-92：Inner Join 或者 Join（由于内连接是默认的联接方式，所以也可以只写Join）

ANSI SQL-89：在两张表间放置一个逗号，在查询的where子句中写过滤条件。

强烈推荐使用Inner Join，因为假如你忘记写过滤条件(on)，执行时SQL会报错。而如果使用SQL-89语法，则不会报错。那么这个内连接就变成交叉连接了! 而且实际开发中SQL语句会非常的复杂，写错的概率就很大了。

**组合连接**

有时候过滤条件不止一个，这时候就要用and来连接多个条件，这就叫做组合连接。例如：Table2表中定义了一个外键（col1和col2），引用了Table1表的col1和col2列，现在要写一个根据主外键关系来连接两个表的查询语句。该条件如下：

FROM TABLE1 AS T1

JOIN TABLE2 AS T2

ON T1.col1 = T2.col1 AND T1.col2 = T2.col2

**不等连接**

如果连接条件包含等号运算符，那么这样的连接叫做等值连接（equi join）。如果连接条件包含除等号以外的其他运算符，那么这样的连接叫做不等连接（non-equi join）。

**多表连接**

一个联接表运算符只对两个表进行操作，而一条查询语句可以包含多个联接。通常，当FROM子句中包含多个表运算符时，表运算符在逻辑上是按从左到右的顺序处理的。也就是说，第一个表运算符的结果将作为第二个表运算符的输入，第二个表运算符的结果将作为第三个运算符左边的输入，以此类推。

**外连接**

外连接会应用内连接所应用的两个逻辑处理步骤（笛卡尔积和On过滤），此外还多一个外连接特有的第三步：添加外部行。

在外联接中，耍把一个表标记为“保留的”表，可以在表名之间使用关键字LEFT OUT JOIN、RIGHT OUTER JOIN ,以及 FULL OUTER JOIN,其中 OUTER 失键字是可选的。 LEFT关键字表示左边表的行是保留的关键字表示右边表的行是保留的，而FULL 关键字则表示左右两边表的行都是保留的。外联接的第三个逻辑杏询处理步骤就是要识别保留表中按照ON条件在另一个表找不到与之匹配的那些行，再把这些行添加到联接 的前两个步骤生成的结果表中：对于来自联接的非保留表的那些列，追加的外部行中的 这些列则用NULL作为占位符。

**外连接常见问题**

**1.对外联接中非保留表的列进行过滤**

当检查涉及外联接的代码时，查找逻辑错误时，应该检查的一个地方就是WHERE子句。如果WHERE子句的条件是以<列><运算符><值>的形式引用了联接中的非保留表的列。这通常就是存在错误的一个标志。因为外联接得到的外部行中来自非保留字的列值均为NULL，而NULL <运算符><值>这种格式只会得到UNKNOWN（除非使用IS NULL运算符，显示查找NULL值）。这种结果相当于抵消了外联接的作用。

**2.在多表连接中使用外连接**

在From子句中运行的表运算符并不适用与“同时操作”，表运算符逻辑上是从左到右计算的。对外联接的处理顺序的调整，可能会影响到输出结果，所以不能随便改变它们的顺序。

例如：假设现在要写一条多表联接查询语句，先对两张表进行外联接，再和第三种表进行内连接。如果在内连接ON子句中的条件是对来自外联接非保留表的列和第三张表的列进行比较，那么所有的外部行都会被过滤掉。

记住，对于外部行，其来自非保留表的列值为NULL；把NULL和任何值进行比较，得到的都是UNKNOWN;UNKNOWN都会被ON过滤掉。

那么我们如何避免这种情况呢？我们可以采用调换顺序的方式来解决这个问题，或者采用（）来让内连接先执行，最后在外联接。

**2.随外联接一起使用COUNT聚合函数**

另一个常见的逻辑错误是与外联接一起使用COUNT聚合函数有关。当对外连接的结果进行分组，再使用COUNT(\*)聚合函数时，聚合操作会把内部行和外部行都计算在内，因为它不计算行数，而不管行的内容。但通常是不希望把外部行也纳入计数的考虑范围之内。

为了解决这一个问题，应该用COUNT(<COLUMN>)来代替COUNT(\*)，并从联接的非保留表中选择一个列。这样一来 COUNT就会忽略外部行，因为这些行在计数列上的值为NULL。记住要选择只有在外部行中列值为NULL的列。

**子查询**

SQL支持在查询语句中编写查询，或者嵌套其他查询。最外层查询的结果集会返回给调用者，称为外部查询。

内部查询的结果是供外部查询使用的，也称为子查询。

子查询也可以分为独立子查询（self-contained subquery）和相关子查询（correlated subquery）两类。独立子查询不依赖于它所属的外部查询，而相关子查询则依赖与它所属的外部查询。

**独立子查询**

所有子查询都有所属于的外部查询。独立子查询是独立于其外部查询的子查询。独立子查询调试起来非常方便，因为总可以把子查询代码独立出来单独运行，并确保它能够正确实现默认的功能。

在逻辑上，独立子查询在执行外部查询之前只要先执行一次，接着外部查询再使用子查询的结果继续进行查询。

标量子查询是返回单个值的子查询。

多值子查询是在一个列中返回多个值的子查询。一些谓词如IN可以处理多值子查询。

在许多情况下，我们既可以使用子查询也可以使用表联接来解决问题。在一些情况下，子查询和联接是一样的；而在另一些情况下，对两者的解释是不同的。我的方法是：对于给定的任务，先用直观的形式写出能解决问题的语句，如果对它的性能不满意，调整方法之一就是重构查询。这样的查询重构可以包括用子查询代替联接，也可以是联接代替子查询。

**相关子查询**

相关子查询是指引用了外部查询中出现的表的列的子查询。这就意味着子查询要依赖于外部查询，不能独立地调用它。在逻辑上，子查询会为外部行单独计算一次。

**EXISTS谓词**

EXISTS谓词它的输入是一个子查询；如果子查询能够返回任何行，该谓词则返回TRUE，否则返回FALSE。

和其他谓词一样，我们可以使用NOT逻辑运算符来否定EXISTS。

性能优化方面

如果SQLServer引擎什么也没有找到，也足以决定子查询是否返回了。而无须处理所有满足条件的行。可以把这种处理方式看做是一种“短路（short-circuit）”，它能提高处理效率。

EXISTS另一个有趣的方面是：与T-SQL大多数谓词不同，EXISTS谓词使用的是二值逻辑，而不是三值逻辑。如果稍加考虑就能明白：不知道查询是否有返回行的情况是不存在的。

**子查询常见问题**

**NULL问题**

当对至少返回一个NULL值的子查询使用NOT IN谓词时，外部查询总会返回一个空集。因为NOT （FALSE OR UNKNOWN）的计算结果为UNKNOWN他会被过滤掉。

解决方案：

1. 在子查询中排除NULL值
2. 使用EXISTS谓词，因为EXISTS是二值逻辑，天生就能过滤UNKNOWN。所以使用NOT EXISTS比NOT IN安全。

**集合运算**

集合运算是对输入的两个或多个集合进行的运算，最终输出一个结果集。

T-SQL支持3种集合运算：并集（UNION），交集（INTERSECT）和差集（EXCEPT）。

集合运算的基本格式为：

*输入的集合1*

*<集合运算>*

*输入的集合2*

*[Order by …]*

需要注意的是，集合运算涉及的两个查询不能包含ORDER BY 子句。但可以为整个集合运算的结果集增加一个ORDER BY 子句。

参与集合运算的两个查询生成的结果集必须包含相同的列数，而且相应列必须具有兼容的数据类型。这里指的是优先级较低的数据类型必须能隐式地转换为较高的数据类型。

集合运算结果中的列名由第一个查询决定，因此如果要分配列名，应该在第一个查询中分配相应的列名。

集合运算还有一个特点就是它任务两个NULL值是相等的 。

**交并差集合运算**

UNION ALL集合运算返回在输入的多集中出现的所有行，它实际上不会对行进行比较，也不会删除重复行。

UNION （DISTINCT）集合通过删除重复记录，可以把两个输入的多集转变为一个集合。

从物理处理过程来看，SQLServer 不一定先删除输入多集中的重复行，在进行集合运算。相反，它可以先把两个多集组合到一起，然后再删除重复行。

运用场景：如果在集合运算中可能存在重复行，且要返回重复行，则使用UNION ALL。如果可能存在重复行，但是返回互不相同的行，则使用UNION。如果在输入两个集合中不可能出现重复行，推荐使用UNION ALL，因为可以避免SQLServer为检测重复行而带来的性能损失。

INTERSECT (DISTINCT)如果一个行在两个输入多集中都至少出现一次，那么交集返回的结果中将包含这一行。

EXCEPT (DISTINCT)集合运算是在逻辑上先删除输入多集中的重复行，然后返回只在一个集合中出现，在第二个集合中不出现的所有行。换句话说，一个行能够被返回，仅当这个行在第一个输入的多集中至少出现一次，而且在第二个集合中没有出现过。需要注意EXCEPT是不对称的。