VisualFoxPro9中新的数据处理方式 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/273/2021_2022_VisualFoxP_c97_273096.htm Visual FoxPro 9.0与以前的版本相比，在数据引擎上做了很大的改进。从增强的SQL语言到支持新的数据类型和索引都作了增强，本文阐述了这个最新版本作为一个成熟开发平台的魅力。 数据引擎的改变主要体现在以下5个方面： 增强的SQL语言：取消了很多硬编码的限制，增强了子查询和关联查询的支持，支持更复杂的表达式，以及增强了对UNION的支持。 性能方面：加入了一个全新的索引方式，增加了过滤型索引的性能，提高了了TOP n，MIN()/MAX()以及LIKE这些查询子句的性能。 命令和函数：对数据操作的更具灵活性，增强对SQL中showplan的支持，增加ICASE()来代替IIF（）函数。 新的数据类型：支持VarChar、VarBinary和BLOB等新的数据类型，并提供相应的类型转换函数：CAST()。增强了现有函数对数据类型的控制和转换能力 远程数据：增强了事务控制的能力，游标机制使得代码逻辑更加清晰，并且对CursorAdapter作了加强，使开发者只需数行代码就可以方便地访问远程视图。 由于提供了与SQL Server强有力的互操作性，Visual FoxPro 9对客户端/服务器模式做了很大的改进。通过支持新的数据类型，并取消了SQL语言的诸多限制，同一套代码可同时运行在本地数据引擎和SQL Server这两种不同的数据源上。 以上是大致的描述，下面让我们深入剖析这些新增功能。 SQL子查询的增强 如果要用一句话来表示SQL子查询的增强程度，那就是：“太多了”！SQL语句中再没有了元素数量的硬编码限制。一

个简单的SELECT语句能包括更多的表、连接、子查询、嵌套子查询和联结。 SQL语句中的IN子句中再也没有数量限制。在以前的版本中IN实际被映射到了一个名字INLIST（）函数中，但现在这种依赖取消了。这个改变使得IN子句能使用更多的参数来生成非常复杂的SQL语句。与原来版本不同，只要找到相应记录，Visual FoxPro 9会自动停止计算IN子句中的表达式，这将带来性能的提高。 完全无限制？ IN参数表的元素也不是完全无限的，它的最大数量等于函数SYS（3055）的返回值，而这个函数的返回值与实际可用内存有关，因此如果你的可用内存越大，那么IN子句支持的元素就越多。无硬编码的限制并不等于完全无限制。像可用内存以及表达式的复杂性都能决定是否能运行一个非常长而且复杂的语句，你要花很大的功夫才能找出它们在你的机器上的真实极限。 增强的子查询功能 子查询在SQL语言中是一个很有用的功能。它一般处于WHERE子句中的右边，充当一个选择器的作用。在Visual FoxPro 9中，子查询还可以处于SELECT的参数列表中（称为投影）以及FROM子句中（称为派生表）。 当你使用投影时，如果子查询没有返回任何记录，那将返加一个空值（NULL）。投影还允许互相关联，以后我们会讲到这点。以下使用投影的SQL语句的一个例子：SELECT .C.CustomerID, .C.CompanyName, .(SELECT YTD_Sales FROM Sales_02 WHERE .C.CustomerID = Sales_02.CustomerID) AS Y02,.(SELECT YTD_Sales FROM Sales_03 WHERE .C.CustomerID = Sales_03.CustomerID) AS Y03,.(SELECT YTD_Sales FROM Sales_04 WHERE .C.CustomerID = Sales_04.CustomerID) AS Y04 .FROM Customers C 这个SELECT

语句返回最后三个会计年度的客户ID和公司名称。 使用投影的限制是子查询只能查询一个字段，并且子查询返回的记录数不能大于1。 投影的另一个有价值的使用方法是它可以成为表达式的一部分，如下所示：SELECT .C.customerID, .C.companyname, .SUM(D.quantity*D.unitprice) AS CustTotal ,. (SUM(D.quantity*D.unitprice) / .(SELECT SUM((quantity*unitprice)-discount) .FROM OrderDetails D2) .)*100 AS PctTotal .FROM Customers C .INNER JOIN Orders O .ON C.customerID = O.customerID .INNER JOIN OrderDetails D .ON O.orderid = D.orderid .GROUP BY C.customerID, C.companyname, O.orderID .ORDER BY pctTotal DESC 这个SELECT语句返回客户ID、公司名称、销售额以及销售额占总销售额的百分比。 注意在以上语句中，子查询充当SELECT列表中的一个复杂表达式，并且它还包含一个聚集函数SUM（），这里我们可以看到使用它的灵活性。 子查询的另一种使用场景即派生表，实际你可以将它看作一个逻辑表。 考虑以下的例子：SELECT .C.customerid, .P.product_count AS p_count.FROM Customers C, .(SELECT c2.customerid, . COUNT(DISTINCT D.productID) AS p_count .FROM Customers C2 .INNER JOIN Orders O .ON C2.customerid = O.customerid .INNER JOIN OrderDetails D .ON O.orderid = D.orderid .GROUP BY c2.customerid) AS P .WHERE C.customerID = p.customerID .AND P.p_count >= .(SELECT (COUNT(*)*.50) FROM Products) .ORDER BY p.product_count DESC 这个SELECT语句返回客户ID、所有购买了50%产品的客户，以及它们所购买的产品数量。 观察以上的语句，你可

以发现派生表有一个名为“ P ”的别名，这与普通字段别名的语法一样——都必须用AS子句来表达。而且这个子查询很复杂（在本例中，它关联了两个表），这个派生表还可以作为WHERE子句的一个条件或者将它放在ORDER BY子句中。与投影不同，派生表能返回多个字段以及多条记录，但它不能相互关联。另外，所有的子查询都会在主句中的SELECT执行之前运行。 子查询还可以充当UPDATE语句中的SET列表。但SET子句只允许使用一个子查询，并且当SET子句使用子查询后，那WHERE子句中就不允许再使用子查询了。 更好的关联支持 新版本中的UPDATE语句和DELETE语句支持关联。这样，一条语句可以引用不同的表，如下所示：DELETE products .FROM mfg .WHERE mfg.productID = products.productID.AND mfg.discontinued = .t. 这个DELETE语句删除mfg表中所有不再生产的产品。 另一个关联UPDATE语句示例如下：UPDATE products .SET unitprice = mfg.msrp *.90 .FROM mfg .WHERE mfg.productID = products.productID 这条UPDATE语句将零售产品的价格打了九折。可能你会问：它支持子查询吗？当然支持。但使用的时候要格外小心。因为如果子查询没有返回任何记录，那将会返回一个值为NULL的空记录。而这恰恰不是你所希望得到的结果。如下所示：UPDATE products .SET unitprice = . (SELECT (msrp *.90) .FROM mfg .WHERE mfg.productID = products.productID) 这条UPDATE语句的作用与上条基本相似，但如果子查询中的产品没有找到的话，那unitprice将被置为NULL。 视图与查询设计器 尽管新版本增强了子查询功能，但不幸的是在视图与查询设计器中并不支持这种增强功能。并且由于SQL中的IN

子句中的元素数目取消了硬编码的限制，但视图与查询设计器中并不知道，因此如果你使用设计器，那IN子句还是只支持24个元素。 增强的UNION操作符 由于联结的数量没有了硬编码的限制，你可以在INSERT INTO子句的结果集中使用UNION。也可以在使用UNION的同时使用ORDER BY子句。 性能 不管你是访问远程数据还是依赖于它强大的本地数据库引擎，Visual FoxPro始终将性能考虑在第一位。Visual FoxPro 9进一步地增强了数据引擎的功能。 二进制索引 这种新型的索引使用方法如下：INDEX ON DELETED() TAG DELETED BINARY 这种索引能与任何非空的逻辑表达式一起使用。除此之外，FOR表达式、ASCENDING、DESCENDING、UNIQUE或CANDIDATE关键字不能与它一起使用。 二进制索引并不支持SET ORDER TO命令，而且INDEX ON命令会将当前的order设为0。但你可在Seek语句中使用二进制索引。 二进制索引最大的优势在于它的索引文件大小。以一个包含800万条记录的表为例，它的二进制索引文件的大小差不多为表大小的三十分之一。尺寸越小意味着在执行APPEND和REPLACE操作时I/O处理速度越快，但遗憾的是所有的Rushmore优化技术目前都还不支持二进制索引。如果SQL语句返回的记录起过表中总记录的三分之一的话，那Rushmore将得到很好的发挥（当所有的记录都命中的话会提高92%的速度）。如果SQL语句返回的记录低于总记录的三分之一，那Rushmore将变得很慢（如果没有记录命中的话会降低两倍的速度）。 在Visual FoxPro 9中，将DELETE语句设置为二进制索引是一种增强性能的简单方法。但要注意版本的问题，因为以前的版本并不支持这种新的索引方式。

Rushmore优化技术 新版本中有使用了大量的Rushmore优化技术，例如对TOP N[PERCENT]作了优化，这样便能提供更好的性能。它一般与ORDER BY子句配合使用，返回前N个或者前百分比数目的记录。在Visaul FoxPro 9中它减少了内排序操作和文件I/O操作，使得执行这种类型的语句占用更少的内存，并且正如此语句所言，它只返回确切的前N个记录。 在以前的版本中，如果要统计一个班级总分排在前十名的学生，但中间存在并列名次的话，那结果会返回多于10个的记录。在适当的情况下，Visual FoxPro 9将在FOR DELETED()子句和FOR NOT DELETED()子句中使用过滤索引来对MIN（）和MAX（）这两个聚集函数进行优化，可以提高MIN()和MAX()的计算速度。 如果LIKE模糊查询语句中的条件以“%”为结尾，那查询速度也将得到很大提高（请注意，“%”只能出现在查询条件的结尾，而不能出现在其它地方，否则优化无效）。这种优化的结果等同于普通WHERE子句查询的速度。 更多的智能索引技术 Visual FoxPro 9比以前的版本更智能化，它充分地让索引获得Rushmore的优化技术，如下所示：INDEX ON DELETED() TAG DELETED 以上语句自动对NOT DELETED（）和DELETED（）条件进行优化，不用你添加一条INDEX ON NOT DELETED()来实现。 这里有个特例，就是当索引过过滤表达式采用了FOR NOT DELETED（）作为Rushmore的优化，并且SET DELETED设为ON的话，那就无须为 NOT DELETED（）作优化。