计算机科学1992 Vol.19 No.5

数据库查询理论

Ashok K. Chandra

Chan, AK 范明

一、引 言

从Codd引进关系数据库开始，在理解怎样才能从数据库中提取数据方面已做了大量工作。已经提出和开发了许多数据库系统和查询语言，包括SQL，QBE，QUEL，L-DL等。与此同时，数据库的查询理论已成长为一个丰满的技术领域，这一发展把更重实效的开发建立在对问题的全面理解之上。确实，现在逻辑程序设计方面开展的研究就是这种情况。本文的目的不是全面地综述数据库查询理论，而是指出随着该领域的发展，使得该领域成型的一些思想。

二、一阶查询和关系代数

第一个重要的思想来自Codd，他考察了查询关系数据库的两种基本方法：一种是使用一阶逻辑，另一种是使用关系代数。一个一阶查询可以表示为$\{x \mid \varphi(x)\}$，它的输出是使得一阶公式在给定的数据库下为真的所有元组x的集合。公式φ可以使用$\forall$、$\exists$、$\wedge$、$\vee$、$\neg$、$=$和给定的数据库关系。此外，可以允许常数和有限制的算术运算。关系代数由使用诸如投影、选择、连接、差等运算符构造的项组成。

Codd指出可用一阶逻辑表示的任意查询也可以用关系代数表示，并且其逆成立。类似的结果早已被逻辑界认识，但是，在数据库界，Codd的结果具有很大的影响。表示查询的两种不同的方法等价这一事实意味着这是一个相当健壮的概念。此外，一阶逻辑看来是相当有表达能力的，并且术语"完备性"时常被用来表明一种查询语言能够表示所有的一阶查询。事实上，当时的查询语言的设计者们认为，他们的查询语言是"完备的"，被认为具有足够强的表达能力。

然而，这并不意味着表达能力比一阶逻辑弱的语言就不重要。事实上，数据库系统的许多查询常常是简单的，并且可以更有效地回答。此外，一阶查询的子集可以充当数据库系统的内部界面。例如，合取查询（和密切相关的制表查询）对应于选择-投影-连接查询（等价地，$\exists$、$\wedge$查询）。它们可以被优化为唯一最小范式，并且可以充当回答一阶查询或逻辑程序设计的内部表示。一种更小的查询类，选择-投影查询（例如，系统R中的"sargable"查询）有时也被用作存取数据，处理索引、锁和日志等低层次数据库子系统的界面。这种查询对应于在单个关系中挑选满足某种性质的所有元组。例如，"列出计算机科学系年薪在五万至十万之间的所有副教授的姓名"（在这个例子中，数据库有一个关系，包含对应于姓名、职称、系别和年薪的列）。

现在回到一阶查询的表达能力，很早就清楚：尽管对于表示所有的一阶查询，它是一种理想的查询语言，但是仅仅具有这种表达能力是不够的。例如，传递闭包查询常常是有用的，但不能用一阶逻辑表示。传递闭包的一个例子是，"列出直接或间接为Smith

QL86")

[3] Database Language SQL with integrity enhancement, ANSI X3, 135-1989, ISO-9075-1989 (E), （即通常所称的"SQL89"）

[4] Database Language SQL2 and SQL3, ISO working draft.

工作的所有人”。（这里，数据库的关系给出了一个机构内部每个职员的经理。）的确，查询语言的设计者们曾试图把他们的查询语言与一种诸如PL/1或Pascal等通用程序设计语言相结合，以提供更强的表达能力。不幸的是，在接口处失去了关系的抽象。在接口处，关系被映射到数组或相反。一种较纯洁的做法是在查询语言本身包含一些附加的结构，如列出一个关系中元组的数目，或计算传递闭包。一个自然的问题是：确定某个结构的集合是否允许人们表达所有的查询。

三、可计算的查询

然而，在回答该问题之前，搞清该问题的真实含义是什么是有帮助的。显然，限于可计算的查询是充分的，这样可以不考虑停机问题，或者计算其它非部分递归的函数。看来更重要的是关于必做的纯数据的提取和操纵问题，而不是在值上做算术运算（例如，求数据库中一个数集的和）。然而，必须证明这种选择是适当的，并且如果必要的话，“不纯”的部分可以不太困难地添加到该理论中。例如，一阶查询和关系代数等价就是这种情况。

第二种简化更多地是属于技术的而不是原理的，是用一个关系定义在其上的有限定义域来定义数据库。现在，数据库上的关系一般被认为是有限的，但是其值的选择取自一个无限域，譬如说U。定义域D是U的一个有限子集，它包括出现在关系中的所有元素（并且还可能包括其它元素）。查询的输出也是D上的一个关系。这种选择的结果是容易处理否定(一阶逻辑)或补(关系代数)。现在，无限集不会出现，因为补是对D，而不是对U定义的。这样，我们可以把数据库B想像为一个有限的关系结构：

$$B=(D,\ R_1,\ R_2,\ \cdots,\ R_K)$$

其中定义域D是域U的一个有限子集，而关系R_1是D^n的子集。这也导致数据库理论和模型论的紧密联系。关系结构上的可计算性已在逻辑方面被广泛研究，但是大部分注意力都放在无限模型上，因为这方面的工作大多是由逻辑、群论、递归论等推动的。事实上，与数据库的联系已经重新引起了对有限模型论的兴趣，并且导致了重大进展。

现在，我们可以问一个“合理的”查询的可能含义是什么。例如，问一个关系的第一个元素是什么可能是不合理的（这将违反关系是元组的无序集的抽象，并且在数据库中信息可能的重新组织下不是不变的）。看来查询的输出应当保持数据库中元素之间的对称性是自然的。谈及的一种方法是，它是数数据库上的一个应当保持输出不变的自同构映射。

一个可计算查询是一个部分递归函数，给定一个数据库作为输入，它产生该数据库定义域上的一个关系作为输出，并满足相容性准则：如果两个数据库是同构的，则它们的输出也是同构的（在同一同构下）。碰巧，这蕴涵了上面的自同构条件。

剩下的问题是，是否有某种查询语言能够表示所有的可计算查询。业已表明，这种语言事实上可以设计出来。一种方法是把关系代数推广到一种程序设计语言中。关系代数仅由项组成，项由给定的关系，使用关系运算符构造。对于程序设计语言，我们需要取关系为值的变量。对这些变量的赋值是关系代数中的项（当然，在项中允许变量），并且我们需要允许语句序列，if-then-else，和while-do结构。使这种语言可以表示所有可计算查询的唯一附加条件是，允许变量的值在计算时变得任意“宽”（具有无界秩）。

宽关系的要求使得这种查询语言可能低效，并且对于实际使用来说常常不直观。最近，Abiteboul和Vianu指出，如果这种语言具有一种机制，能创建数据库或计算中未出现的新元素，则宽关系的要求可以去掉。他们还强调了数据库查询的非确定输出问题。尽管如此，寻求一个有效的、直观有吸引力的查询语言，以表示所有可计算的查询，仍是一个尚待解决的设计问题。

但是，数据库查询的实质是否被抓住，或者，这些简化是否使得问题过于容易？结果是实际查询的附加复杂性可以相当容易地处理。例如，在上面的查询例子中，有一些像"计算机科学"、"副教授"等常量。人们并不期望在这些常量到数据库的其它元素的自同构映射下输出是不变的。而我们确实需要进行数的比较（年薪<100,000），并且处理其产生不在数据库中的元素的查询，如"求所有雇员年薪的和"。可计算查询的概念可以容易地加以扩充，以包括这样的部分解释，并且对查询语言的适当修改就足以表达这些推广。

可计算查询的概念已被推广到其它方面，例如，层次结构或词典和网络与层次数据库模式。

把一价查询看作最小的合理查询集，而可计算的查询为最大的，人们可能会问在它们中间有什么。或者，各种查询语言结构的表达能力是什么？

四、不动点、逻辑程序设计和否定

4.1 不动点查询

如前面注意到的，计算一个二元关系的传递闭包看来是一种需要增加到一阶查询的自然功能。不幸的是，在语言中仅包含传递闭包有点笨拙难用。要增加的一个更自然的结构是最小不动点。由

$$TC(x,y)\leftarrow x=y$$

$$TC(x,y)\leftarrow R(x,z)\wedge TC(z,y)$$

给出的查询TC是关系R的（自反）传递闭包。上式是用逻辑程序设计的记法给出的，并表示$TC(x,y)\equiv(x=y)\vee\exists z(R(x,z)\wedge TC(z,y))$的最小解。一般地，在关系T上单调的公式$\varphi(x,T)$（即，如果$T_1\subset T_2$，则$\varphi(x,T_1)\subset\varphi(x,T_2)$）具有一个最小不动点，即不动点$T(x)\equiv\varphi(x,T)$。这可以用标准的Tarski结构计算：设$T_0=\{\}$，且$T_{i+1}=\{x|\varphi(x,T_i)\}$，则$T_\infty=\bigcup_i T_i$是最小不动点。确保$\varphi(x,T)$单调的一个充分条件是T仅在$\varphi$中正出现（T的每一出现都在偶数个否定词下）。

一个自然的查询类——不动点查询通过用不动点扩充标准的一阶运算符$\exists$、$\forall$、$\wedge$、$\vee$、$\neg$得到。这里，我们可以取一个一阶公式的不动点，运用某些一阶运算符于其上（包括可能的否定），在一个不同的关系符号上取另一个不动点，如此下去，只要保持正条件。

还有另外的定义不动点的方法，结果导致相同的查询集，从而也暗示该集合相当健壮。例如，只$\varphi(x,T)$要在T中单调，而不必是正的，我们就可以使用不动点运算符。不幸的是，单调性条件并不容易检查。事实上，它是不可判定的。尽管如此，也许有理由相信，允许某种类型的单调不动点可能产生比正不动点更大的类，因为单调的一阶公式比正一阶公式表达力更强（在有限结构上）。一个表面上更强的不动点结构是膨胀不动点。这里，我们对任意公式$\varphi(x,T)$取不动点T_∞是通过把Tarski结构修改为$T_0=\{\}$，且$T_{i+1}=T_i\cup\{x|\varphi(x,T_i)\}$，并且$T_\infty=\bigcup_i T_i$（这里$T_\infty$是$T(x)\equiv T(x)\vee\varphi(x,T)$的一个不动点，但不必是最小的）。膨胀不动点看上去比最小不动点更为一般，因为如果是单调的，则两不动点一致。然而，Gurevich和Shelah表明，正不动点，单调不动点和膨胀不动点表示相同的查询集。

不动点查询还有一些有趣的性质。Immerman给出了一个令人吃惊的结果：一个不动点的补本身也可以表示成一个不动点（在无限结构下，有些事不成立）。事实上，所有的不动点查询都可以通过取单个不动点（后随某些简单的一阶操作）表示。换言之，为表示所有的不动点查询，没有必要取一次以上不动点。最近，已表明，在取膨胀不动点时，甚至没有必要使用全称量词。不动点查询可以表示成存在的一阶公式的膨胀不动点（的投影）。在量词后的公式体中，允许否定词。

4.2 逻辑程序设计

逻辑程序设计是不动点出现的一个重要领域。一个自然的查询集通过仅考虑数据操纵（不使用函数符号或算术运算符）得到。这些称为Horn子句查询（也称作datalog查询）。它们由一组Horn子句组成，这些Horn子句用给定的数据库关系和新关系，通过取最小不动点定义新的关系。Horn子句查询实际上有两种版本（通过允许或不允许≠得到），但是这里我们将不关心其不同。上面的查询TC就是一个Horn子句查询，下面的T也是（这里，T用T_1定义，而T_1用T定义）。Horn子句右端的自由变量被看作是存在量化的。

$$T(x,y)\leftarrow R(x,z)\wedge T_1(z,y)$$

$$T_1(x,y)\leftarrow R(x,y)$$

$$T_1(x,y)\leftarrow T(x,z)\wedge R(z,y)$$

可以证明Horn子句查询的相互递归可以归结为单个关系上的递归。因此，这些查询可以看作取仅包含∃、∧、∨（即，没有全称量词或否定）的一阶公式的不动点。这些Horn子句查询包括某些非一阶查询（例如，传递闭包），但是它们不能表示所有的一阶查询（包括一个简单的全称量词或一个关系的补）。考虑到早期的查询语言设计者期望至少要表示所有的一阶查询，看来对许多情况来说，Horn子句查询表达能力太弱了。

4.3 逻辑程序设计的拓广

几个方向看来有希望扩充纯Horn子句查询，使得它们具有更强的表达能力，并且仍然保持将查询看作"子句"（或许用某种方式加以扩充）集合的程序设计风格。可能性包括使用否定，全称量词，或使用嵌套关系（关系的每个元素本身可以是关系，如此下去）。

在逻辑程序中任意地使用否定的一个明显问题是不动点可能不存在。例如，下式就没有不动点：

$$T(x)\leftarrow\neg T(x)$$

然而，如果只否定数据库关系，但不否定新关系（对新关系取不动点），则这种困难不会出现。例如：

$$T(x,y)\leftarrow\neg R_1(x,y)\wedge\neg R_2(x,y)\wedge R_4(x,y)$$

$$T(x,y)\leftarrow\neg R_3(x,z)\wedge T(z,y)$$

一种自然的可能性是对上例进行拓广。（除R_3，R_4之外）上面使用了数据库关系R_1，R_2，但是这些可能被替换成其本身先需用不动点计算的关系。例如，在下面的查询中，S_1，S_2先计算，然后在T的计算中使用。这符合将否定看作失败的观点。

$$T(x,y)\leftarrow\neg S_1(x,y)\wedge\neg R_3(x,y)\wedge R_4(x,y)$$

$$T(x,y)\leftarrow\neg S_2(x,z)\wedge T(z,y)$$

$$S_1(x,y)\leftarrow x=y$$

$$S_1(x,y)\leftarrow\neg R_5(x,z)\wedge S_2(x,y)$$

$$S_2(x,y)\leftarrow R_6(x,z)\wedge S_1(z,y)$$

这里的关系R_3至R_6又可以被计算出的关系替换，如此下去。这种用"分层"的方式定义查询曾多次提出，并称作分层查询。它们看来是合理、自然、有表达力和可以有效实现的，并且已被包括在一些数据库系统中，例如，Nail!和LDL!。

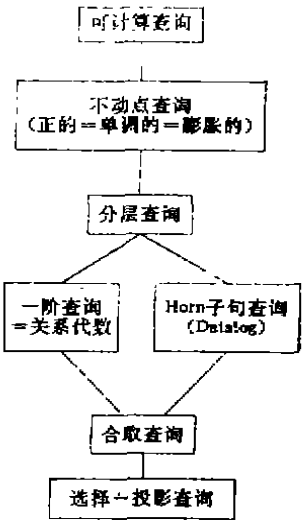

图1. 一阶和不动点查询

显然，分层查询能够表达所有的一阶查询，因为在任何一层都可以使用∃，∧，并

且在层之间允许否定，然而，最近的结果表明，分层查询不能表示所有的不动点查询，特别是它们在全称量词上取不动点很困难。例如，想像定义域由二人游戏的棋局组成，关系Move(x,y)表明一个游戏者可以从棋局x到棋局y，而关系G(x)给出第一个游戏者已取胜的棋局，则第一个游戏者可以由之取胜的棋局由以下方程的最小不动点给出：

$$Win(x)\equiv G(x)\vee\exists y\ (Move(x,y)\wedge\forall z(Move(y,z)\rightarrow Win(z)))\ .$$

并且它不能被分层查询表示。

也已考虑逻辑程序设计的其它推广，包括全称量化，和嵌套关系的使用。它们的准确特征尚不清楚，尽管已证明在逻辑程序设计中增加嵌套关系的几种方法具有等价的表达能力。值得注意的是，嵌套关系可以用来表达基本层上任意复杂的查询——需要指数时间，双指数时间，三指数时间等等。

还有些可以处理否定的替代方法，尽管它们看来不具备有效的实现。例如，我们可以取$S(x)\equiv\varphi(x,S)$，表示所有不动点的交，所有极小不动点的并（有时称作广义封闭世界假设），或存在时表示唯一的最小不动点。为了理解否定的表达能力和如何在逻辑程序设计中使用它，仍有许多工作要做。

数据库查询理论的目的不仅是考虑使用一定的结构能够表达什么，而且要考察它不能表达什么。就这一点而论，有一些非常简单的问题不能被不动点查询表示。不动点查询不具备的一种能力是计数。例如，不动点查询不能指出一个关系的长度是奇还是偶：如果数据库B不包含定义域D之外的关系，则下面的查询EVEN不是一个不动点查询：

$$EVEN=\begin{cases}\text{True} & \text{如果 }|D|\text{ 为偶数}\\ \text{False} & \text{否则}\end{cases}$$

形式地，我们可用空集表示False，而用包含零长度的向量的单值集合表示True。

如果某查询具有较低的计算复杂性，则人们可能愿意对它提问。因此，从一种语言中排除容易计算的查询多少有点危险。这把我们引向查询的计算复杂性问题。

五、查询的复杂性

不难看出，不动点查询可以在多项式时间，即在数据库长度的多项式时间内计算。然而，正如我们看到的，有一些简单的多项式查询（如EVEN）不是不动点查询。我们可以简单地把多项式时间查询(P-查询）定义为可以在多项式时间内计算的可计算查询。P和P-查询的基本不同是P-查询必须满足第三节提到的相容性准则。类似地，对于任意的复杂性类C(如NP,NC,PSPACE等)，我们有一个C-查询集。这些查询复杂性类的行为与纯复杂性类非常相同，即（在一定条件下）C_1-查询$\subset C_2$-查询，当且仅当$C_1\subset C_2$。这样，P-查询=NP-查询，当且仅当P=NP，这在当前尚不知道。

一阶查询具有相当好的性质——它们都是NC可计算的，即，它们可以使用多项式个处理机，在多项式-对数时间内(事实上，对数时间内）计算。不幸的是，不动点查询（甚至Horn子句查询）看来不具有这种性质——它们看来是固有顺序性的（有些不动点查询是P完全的）。

事实上，由复杂性类定义的查询和由逻辑结构定义的查询之间甚至有更密切的联系。Fagin由有限模型论得到的一个有趣结果是：NP-查询恰是存在二阶查询(使用存在二阶量词，后随一个一阶公式定义的查询)。这一结果被拓广，表明二阶查询恰是具有多项式时间分级复杂性的查询（图2）。

如果我们将数据库配上一个线性序，则复杂性和逻辑的附加的紧密联系就会显露出来。线性序是数据库定义域上的一个特殊的二元关系<，它线性地确定定义域的次序。在数据库中，关系<不必显式提供，它可以在需要时加以计算，因为数据库元素可以被看作位串，这些位串可以进行比较，提供次序。使用线性序，Sazonov，Immerman和Vardi独立地证明不动点查询与多项式时间查询（P-查询）相同。一阶查询（使用线

性序）还与AC^0-查询相同（AC^0-查询是使用多项式长度的常数深度的电路，允许无界扇入可计算的函数类）。

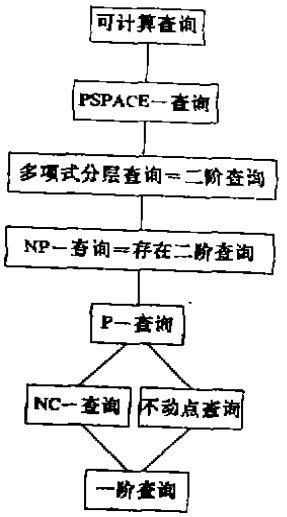

图2. 查询的复杂性

线性序的使用看来是获得数据库查询表达能力的有效办法。例如，查询EVEN可以用不动点和线性序表达（简单的原因是它在多项式时间内可计算）。然而，线性序可能对程序员看来不太直观，并且在实践中可能难以有效地使用。似乎考虑其它具有更强表达能力的程序设计风格的原语也是相关的。

六、程序设计原语

一种明显的程序设计风格式查询语言可以使用取（有界宽度的）关系为值的变量，将一阶表达式（等价地，关系代数表达式）赋值给变量的赋值语句，和允许语句序列构造。但仅用这些结构只能表达一阶查询。

对这些基本结构，我们可以增加循环语句，如

For all tuples t in relation S do Body od

（本质上这类似于SQL中的Select-from和光标原语）。这里，S是变量，而Body对S中每个元组t执行（当进入循环时，使用S的值）。如果循环的体（Body）能够涉及t，则存在一个困难：S中元组t的选择次序的选取必须使用非确定机制。这里有一些选择——其一是在Body中不允许引用t；另一种是接受非确定的查询；第三种是引进使得循环的结果确定化的机制。

第一种情况导致有界循环查询。这里，循环结构可以表示成：For |s| do Body od.

使用这种结构的一个例子是：

TC←{(x,x) |x∈D};
For |D| do
  TC_1←{(x,y)| (x,y)∈TC∨∃z((x, z)∈TC∧(z,y)∈R);
  TC←TC_1
od.

现在有界循环查询都是多项式时间可计算的（由于每个循环都只能执行多项式次——注意，变量具有有界宽度），并能够表示所有的不动点查询。事实上，有界循环也可以被看作一种逻辑结构，其能力至少象不动点结构一样强（事实上更强）。有界循环的另一个例子是第四节提到的查询EVEN：

EVEN←true;
FOR |D| do EVEN←¬EVEN od.

这表明有界循环查询比不动点查询能力更强，它们至少可以对任意常数进行计数。不幸的是，他们不能象P-查询那样计数。事实上，可以证明下面的查询EQUAL（它是NC-查询）不能使用有界循环查询表示（这里数据库有两个一元关系R_1和R_2），

$$EQUAL\begin{cases}\text{True} & \text{如果} R_1=R_2\\ \text{False} & \text{否则}\end{cases}$$

一种不同的循环语句是while语句，它可以记作：

while S={ } do Body od.

由这种结构（用有界宽度的变量，如上面的S，一阶赋值和顺序）表示的查询形成了一个相对自然的类——称之为while查询（它们和RQ或LE相同）。这些while查询也能表示所有的不动点查询，但是它们看来不是多项式时间可计算的。有一些while查询，其复杂性是多项式空间完全的，并且，

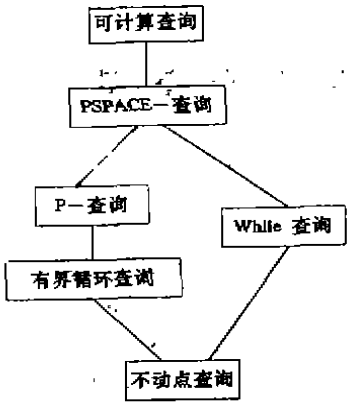

图3. 程序设计原语

事实上若给定数据库定义域上的线性序，则while查询与PSPACE查询相同。此外，Abiteboul 和Vianu已表明另一个查询类 (Sdet TL) 等于这些while查询。尽管已发现相当简单的查询，如EVEN都不是while查询，但是该查询类尚未被很好认识。例如，尚不知道是否存在一个while查询，它不是不动点查询（不做诸如P≠PSPACE的假定）。

许多附加的程序设计结构已被考虑，并且一些特性已被证明，但是许多工作尚待去做。

七、结束语

本文集中讨论了数据库查询的表达能力，许多问题未曾涉及。一个重要的问题是优化。的确，优化问题由于查询语言的层次不同而显得很不相同。例如，选择-投影查询是相对低层次的查询，优化大多涉及存取关系的索引的选择。对于合取查询，存在可执行的叠合的全局优化问题，尽管其复杂性是NP完全的。对于一阶查询和Datalog查询，可以使用诸如代数化简等各种试探方法，或使用魔集，判定一个Datalog查询是否有界等等。关于如何优化分层查询、不动点查询或更大的查询类，所知甚少。

更新数据库通常被认为是查询语言设计后加上的。为了理论上的原因，数据库的改变（或数据库本身的更新）通常可以简单地看作在原数据库上回答查询的结果，并因此认为没有什么新东西。然而，当我们考虑更新的复杂性对回答查询本身的复杂性时，可能出现不同。并且，甚至在不调用更新的查询语言中，更新语句的使用也可能是自然的。

在第五节，我们考虑了作为数据库大小的函数计算查询输出的复杂性。还有一种复杂性概念：使数据库固定，作为查询大小的函数计算输出的复杂性。对我们讨论过的数据复杂性，这种复杂性有时称为表达复杂性。通常表明，表达复杂性比对应的数据复杂性高指数阶。

考虑这里讨论的数据库概念（即，有限关系结构）和演绎数据库概念之间的联系是有趣的。在后者，人们认为数据库不仅包含一些事实（如在有限结构中），而且包含一些附加的性质，它们可能是一阶或高阶的（例如，当使用封闭世界假定时。一个演绎数据库可以简单地看作一个关系结构和一个附着在其上的不动点查询就是这种情况。当事实和附加的性质不是仅看作一个关系的简洁表示，而是看作代表某种现实的逼近（其本身也可以是一个关系结构）时，就会出现一个更重要的区别。当我们允许标准数据库中有空值时，也会出现这种情况。逼近可以看作代表与给定的事实相容的现实的集合（可能无限）。在这种逼近下的查询的含义是什么是相当基本的。在泛关系的情况下，该问题已被考察，但一般性问题仍尚未解决。

在结束本文时，我们注意到，今天的大部分查询语言都基于一阶查询，或基于不动点查询的某个子集。然而，正如上面所指出的，某些非常简单的查询不属于这些类，并且看来查询语言应当增补，以提供这些能力。清醒地认识到增强数据库查询理论对于查询语言的设计将是有帮助的。但是该理论本身还有待发展完善。查询理论的目标之一是提供对查询语言结构的理解，以便能够设计出使用自然，表达力强，并且在实践中有效的查询语言。其另一个目标总是优雅。

（参考文献略）