第18章 信息空缺

18.1 引言

在现实世界中，信息是经常空缺的；比如“不知道生日”、“等待宣布的发言人”，“当前住址不祥”，等等，对此我们大家都已习以为常。为此，很明显在数据库系统里需要某些方法处理这样的信息空缺问题。这个问题的解决方法是基于空值(null)和三值逻辑(three-valued logic,3VL)，并已很普遍地运用于实际系统中，尤其是SQL及许多相关商业产品中。举个例子，我们可能不知道某些零件的重量，比如零件P7，于是可能会很随意地说这个零件的重量“是空”——这句话更准确的意思是：(a)我们知道这个零件存在；(b)也知道这个零件有重量；(c)但我们不知道重量是多少。

在此情况下，显然不能在零件P7这个元组上放一个真正的WEIGHT值。于是，替代的做法是在这个元组的WEIGHT位置标上“null”标记，把这个标记的精确意思解释为我们并不知道这个标记的真正值是什么。这里说WEIGHT位置“包含一个null”，或认为WEIGHT值“是null”，是一种不十分严格的说法，虽然在实际中经常这样讲。严格地讲，说某些元组的WEIGHT字段值“是null”，其实是说那个元组根本没有包含WEIGHT值。这里不赞成使用“null value(空的值)”这种说法，原因是：精确地讲，null不是值——重复一下，它们是标记或标志。

接下来，我们将在下一节看到任何包括空值操作数的标量比较式(scalar comparison)的值都等于真值unknown(未知)，而不是true(真)或false(假)。这样做的理由是把null解释成“unknown值”：如果*A*的值不知道，那么很明显，*A*是否大于*B*(比如说)也是unknown，不管*B*的值是多少(甚至——也许比较特殊——*B*的值也是unknown)。于是，尤其需要注意的是，两个空值之间不能互相相等；即，如果*A*和*B*都是null，则比较式*A*=*B*的值等于unknown，而不是true(它们也不能被认为是不相等，即比较式*A*≠*B*的值也是unknown)。因此就有了术语“三值逻辑”(3VL)：空值概念导致这样的逻辑，它包括三个真值，true、false和unknown。

在进一步讨论之前，应当很清楚的是，以我们的观点来看(顺便提一下，其他很多作者也持这个观点)，空值和3VL是非常严重的错误，在像关系模型这样清晰规范的系统中应该没有它们的位置[⊖]。但是，对空值和3VL不加介绍是不合适的；因此本书保留这部分内容。

本章的内容是这样安排的。引言之后，在18.2节里，先描述空值和3VL的基本思想，但不对这些思想提出很多的批评(很显然，如果不介绍这些思想是怎么回事，就不可能对

⊖ 比如，某个零件元组没有包含WEIGHT值是说——根据定义——那个元组根本不是一个零件元组。相应地，也就是说该元组不是零件谓词的一个实例化。事实上，正是在试图准确说明空值为何物的过程中，显示了这一想法的不一致性。从而导致了很难对该问题给出一个一致性的解释。引用文献[18.19]中的一句话：“如果只是一带而过不假思索地看待这一问题，那么一切看起来都是合理的”。

它们进行适当和公正地批评)。然后在18.3节里，我们讨论了这些思想所带来的一些重要的后果，以此来证明我们的立场是正确的，即空值是一个错误。 18.4节考虑了空值在主码和外码上的潜在影响。在 18.5节里，考虑了在空值和 3VL环境中所遇到的操作，即外连接操作。18.6节非常简短地介绍了信息空缺的一种替代解决方法，即使用特殊值。 18.7节略述了SQL的有关空值方面的内容。最后，在 18.8节里进行了小结。

最后一个预先要引起注意的问题是：当然有很多原因不能在某些元组的某些位置放置真正的数据值 —— “ 未知值 ”仅仅是其中一个原因。其他原因包括“ 值不可应用 ”、“ 值不存在 ”、“ 值没有定义 ”、“ 值没有提供 ”、等等[18.5] ⊖。其实，在文献 [5.2]里，Codd提出关系模型应当包括两个空值而不是一个空值：一个意味着“ 值不知道 ”；另一个意味着“ 值不可应用 ”。于是他进一步提出 DBMS应当处理的是四值逻辑，而不是三值逻辑。在[18.5]中我们对这种提法已提出异义；在本章里，我们仅仅讨论单一种类的空值，即“ 值不知道 ”型空值，以后我们按照定义，经常 —— 而不是不变 —— 把它称为UNK（对于值不知道类型）。

18.2 3VL方法概述

在这一节里，简要叙述应用于信息空缺上的 3VL方法的基本组成部分。下面首先讨论空值（特指UNK）对布尔表达式的影响。

1. 布尔表达式

前面已经说过，只要标量比较式中的两个操作数之一是 UNK，那么这个比较式的值就等于真值 *unkown*，而不是 *true*或*false*。接下来讨论三值逻辑（ 3VL）。*Unknown*（以后经常 —— 但不是不变地 —— 把它缩写为 *unk*）就是“ 第三个真值 ”。下面是AND、OR和NOT在3VL上的真值表（ *t=true, f =false, u =unk*）：

AND	t	u	f
t	t	u	f
u	u	u	f
f	f	f	f

OR	t	u	f
t	t	t	t
u	t	u	u
f	t	u	f

NOT	
t	f
u	u
f	t

举例，假设A=3，B=4，C是UNK，那么下面的表达式具有所示的真值：

```
A > B AND B > C  :  false
A > B OR  B > C  :  unk
A < B OR  B < C  :  true
NOT ( A = C )    :  unk
```

然而AND、OR和NOT并不是所需要的唯一布尔操作符 [18.1]；另一个重要的操作符是MAYBE[18.5]，它的真值表如下：

MAYBE	
t	f
u	t
f	f

为了说明为什么 MAYBE是需要的，现在来考虑这样一个查询：“ 查出可能是 —— 但不确切知道是 —— 在1971年1月18日之前出生的、工资少于 50 000美元的程序员 ”。用

⊖ 需要说明的是，其实并不存在上述所说的各种各样的“ 空缺信息 ”。比如，如果我们说佣金对于雇员Joe“ 不适用 ”，可以非常清楚地说佣金属性不应用于 Joe；这里并不存在信息空缺（然而还是这种情况，如果，比如在Joe的“ 雇员元组 ”的佣金属性中“ 包含 ”一个“ 不适用的 null ”，那么这个元组就不是一个雇员元组 —— 即它不是一个“ 雇员 ”谓词的实例。

MAYBE操作符，这个查询可以像下面这样简洁地表达出来：

```
EMP WHERE MAYBE ( JOB = 'Programmer' AND
                  DOB < DATE '1971-1-18' AND
                  SALARY < 50000.00 )
```

（假定表EMP的属性JOB、DOB和SALARY的类型分别是CHAR、DATE和RATIONAL（有理数））。然而，如果没有MAYBE操作符，这个查询可表达如下：

```
EMP WHERE ( IS_UNK ( JOB ) AND
            DOB < DATE '1971-1-18' AND
            SALARY < 50000.00 )
    OR    ( JOB = 'Programmer' AND
            IS_UNK ( DOB ) AND
            SALARY < 50000.00 )
    OR    ( JOB = 'Programmer' AND
            DOB < DATE '1971-1-18' AND
            IS_UNK ( SALARY ) )
    OR    ( IS_UNK ( JOB ) AND
            IS_UNK ( DOB ) AND
            SALARY < 50000.00 )
    OR    ( IS_UNK ( JOB ) AND
            DOB < DATE '1971-1-18' AND
            IS_UNK ( SALARY ) )
    OR    ( JOB = 'Programmer' AND
            IS_UNK ( DOB ) AND
            IS_UNK ( SALARY ) )
    OR    ( IS_UNK ( JOB ) AND
            IS_UNK ( DOB ) AND
            IS_UNK ( SALARY ) )
```

（假定存在另一个真值操作符叫IS_UNK，它是一元操作符，如果操作数的值是UNK，则返回true；否则返回false。）

顺便提一句，上述内容并不表明MAYBE是3VL需要的唯一新的布尔操作符。其实，比如TRUE_OR_MAYBE操作符也是非常有用的[18.5]。见[18.11]的注释。

2. EXISTS和FORALL

尽管本书的大部分例子是基于代数而不是演算，但我们还需考虑在演算量词（calculus quantifer）EXISTS和FORALL上的3VL的潜在影响。如在第7章介绍的一样，我们分别把EXISTS和FORALL定义为迭代的OR和AND。换句话说，如果（a）*r*是一个关系，它具有元组*t1*，*t2*，...，*tm*；（b）*V*是在这个关系上的范围变量；并且（c）*p*(*V*)是一个布尔表达式，其中*V*是一个自由变量，那么表达式

```
EXISTS V( p (V))
```

被定义成等价于下列表达式

```
false ORp(t1) OR ...ORp(tm)
```

同样，表达式

```
FORALL V(p(V))
```

被定义成等价于如下表达式

```
true AND p(t1) AND...ANDp(tm)
```

因此，如果对于某些*i*，*p*(*i*)等于*unk*，这将会怎么样？现通过一个例子来说明，假设关系*r*正好等于下列元组：

```
(1,     2,      3 )
(1,     2,    UNK)
(UNK, UNK, UNK)
```

为了简单化，假定（a）上述从左到右的三个属性分别叫做 *A*、*B*和*C*；（b）每一个属性都是INTERGER类型。于是下面的表达式具有如下所示的值：

```
EXISTS V ( V.C > 1 )              :  true
EXISTS V ( V.B > 2 )              :  unk
EXISTS V ( MAYBE ( V.A > 3 ) )    :  true
EXISTS V ( IS_UNK ( V.C ) )       :  true

FORALL V ( V.A > 1 )              :  false
FORALL V ( V.B > 1 )              :  unk
FORALL V ( MAYBE ( V.C > 1 ) )    :  false
```

3. 计算表达式

考虑下面数字表达式

```
WEIGHT * 454
```

这里WEIGHT表示某些零件的重量，比如 P7。如果零件 P7的重量是 UNK会怎么样？——于是，这个表达式的值会是什么？答案是它必须为 UNK。通常，对于任何数字表达式，只要其中任何一个操作数本身是 UNK，那么这个数字表达式就等于 UNK。于是，例如，如果WEIGHT是UNK，那么所有的下列表达式也都等于 UNK：

```
WEIGHT + 454        454 + WEIGHT        + WEIGHT
WEIGHT - 454        454 - WEIGHT        - WEIGHT
WEIGHT * 454        454 * WEIGHT
WEIGHT / 454        454 / WEIGHT
```

注意：这里应当指出，按照上述对数字表达式的处理方式会造成某些异常。比如，表达式WEIGHT-WEIGHT应当等于零，但其实等于 UNK。再如表达式 WEIGHT/0应当会产生一个“除零”错误，但结果也是 UNK（当然首先假定在上述两种情况里的 WEIGHT是UNK）。但在未进一步给出说明之前，我们先忽略这些异常。

类似的考虑也适用于所有其他标量类型（ scalar type）和操作符，但以下情况例外（ a）比较操作符（参见前两小节）；（ b）以前讨论过的操作符 IS_UNK；和（c）下面要讨论的操作符 IF_UNK。这样，比如，如果 *A*是UNK或*B*是UNK或两者都是 UNK，则字符串表达式A||B返回UNK（同样，这里也会有某些异常情况，在此略去了这些情况的细节。）

IF_UNK操作符在两个标量表达式操作数上进行操作。如果第一个操作数不等于 UNK，那么它返回第一个操作数的值；否则返回第二个操作数的值（换句话说，这个操作符提供了一种有效的途径，它把一个 UNK转换为某些非 UNK值）。比如，供应商的 CITY属性允许是UNK。于是表达式

```
EXTEND S ADD IF_UNK (CITY, 'City unknown') AS SCITY
```

产生这样的结果：如果 S中的供应商的 CITY属性是UNK，那么这个供应商的 SCITY属性的值是‘ City unknown ’。

注意，顺便提一句， IF_UNK可以用IS_UNK来定义。为了清楚起见，表达式

```
IF_UNK(exp1, exp2)
```

（这里符号 *exp1*和*exp2*必须是相同的类型）和下面的表达式等价

```
IF IS_UNK (exp1) THEN exp2 ELSE exp1 END IF
```

4. UNK不是*unk*

理解UNK（“未知型”的空值）和 *unk*（unknown真值）不是同一回事[⊖]很重要。其实，

⊖ 然而SQL3认为它们是同一回事（见附录 B）。

这种情况是基于这样的事实： *unk*是一个值（精确地说是一个真值），而UNK根本不是一个值。现让我们进一步解释清楚一点。假设 *X*是一个BOOLEAN类型的变量。于是 *X*一定有 *true*、*false*或*unk*三者中的一个值。这样，语句“ *X*是*unk* ”精确的意思是 *X*的值被知道为是 *unk*。相比而言，语句“ *X*是UNK ”的意思是 *X*的值不知道。

5. 一个域可以包含一个UNK吗

UNK不能出现在域里（域是值的集合）也是基于 UNK不是值这个事实。其实，如果一个域可以包含一个 UNK，那么这样一个域的类型约束检查就不会失败！然而，既然域事实上不能包含 UNK，那么一个包含 UNK —— 不管它是什么 —— 的“关系”其实根本不是一个关系，不管根据在第 5章给出的定义还是 Codd在[5.1]中给出的原始定义。稍后，我们会在18.6节里讲述这个重要的观点。

6. 关系表达式

现在把注意力转向 UNK对关系代数操作符的影响。为了简单起见，我们局限于乘积、选择、投影、并和差（ UNK对其他操作符的影响可以依据 UNK在这五个操作符上的影响来确定）。

首先，乘积操作不受影响。

第二，重新定义（稍微改动）选择操作，以返回只包含在选择条件上为 *true*的元组的关系，即选择条件对这些元组的值不是 *false*，也不是 *unk*。注意：前面我们已在“布尔表达式”小节的 MAYBE例子里隐含地假定了这个重定义。

接下来是投影。投影当然包括对重复元组的删除。在传统的两值逻辑（ 2VL）中，两个元组完全相同，当且仅当两者有相同的属性且相应的属性有相同的值。然而，在 3VL中某些属性的值可能是 UNK，而UNK（我们已经知道）和任何东西都不相等，甚至和它自己。那么我们是否要作出结论，一个包含 UNK的元组永远不与任何元组相等，甚至它自己本身吗？

按照Codd的说法，这个问题的答案是否定的：两个 UNK，甚至它们互不相等，也被认为其中一个是另一个的副本，这样做的目的是为了删除重复元组 [13.6] ⊖。下面是为这样明显的矛盾进行的辩护：

> 删除重复元组所进行的等同性判定，与检索条件中进行的等同性判定相比，在内容上考虑得更细致一些。因此，可以采用不同的规则。

我们把这个基本结论是否有理留给大家去判断。不过现在让我们先接受它，于是有下面的定义：

- 两个元组是相同的当且仅当（ a）它们有相同的属性；且（ b）对于每一个属性，两个元组在这个属性上要么有相同的值要么都是 UNK。

有了这扩展的“相同元组”定义，原先的投影定义可不加修改便可适用。

并同样要删除多余的重复元组，相同元组定义同样适用于这里。于是，我们定义关系 *r1*和*r2*（具有相同的类型）的并是这样的关系，这个关系包含所有这样的元组：即 *r1*的某个元组的副本或 *r2*的某个元组的副本或是两者中某个元组的共同副本。

⊖ [13.6] 是Codd的第一篇讨论信息空缺问题的论文 (虽然这个问题不是这篇论文的主要论点 —— 见第13章)。在其他方面，这篇论文提出了 Θ-join、Θ-select(即选择)和除操作符(见练习18.4)等的“ maybe ”方案，以及并、交、差、 Θ-join、和自然连接操作符 (见18.5节)等的“ outer ”方案。

最后，差也类似地重新加以定义，尽管它不包括任何副本删除；即一个元组 *t*出现在 *r1* MINUS *r2*里，当且仅当它是 *r1*中某些元组的副本而不是 *r2*中元组的副本（至于交，也做同样的重新定义；即一个元组出现在 *r1* INTERSECT *r2*里，当且仅当它同时是 *r1*的某个元组和*r2*的某个元组的副本）。

7. 更新操作

对此有两个一般性的问题值得注意：

1) 如果关系 *R*的属性 *A*允许 UNK，并且如果一个在属性 *A*上没提供值的元组插入 *R*中，则系统会自动在这个元组的属性 *A*位置上放置 UNK。如果关系 *R*的属性 *A*不允许 UNK，则试图通过 INSERT或UPDATE把在 *A*位置上是 UNK的元组插入 *R*里是一个错误（当然，在这两种情况里都没有定义 *A*的默认值是非 UNK）。

2) 和通常一样，试图通过 INSERT或UPDATE在*R*里创建一个相同元组是错误的。这里“相同元组”的定义和上面是一样的。

8. 完整性约束

正像第 8章所解释的，一个完整性约束本质上是一个结果不等于 *false*的布尔表达式。因此，如果一个约束的值等于 *unk*，则不认为它是违反约束的（其实，在本节先前部分关于类型约束已经隐含地论述了许多这方面的内容）。当然，从技术上讲，我们应当说不论约束是否成立，都不是未知的，为此，就像在 WHERE子句中把 *unk*当作*false*一样，为了完整性约束而把它认为是 *true*（比较不严格的说法）。

18.3 上述方案所造成的某些结果

上一节所论述的 3VL方法会产生一些逻辑结果，但并不是所有的结果都很明显。我们在这一节里讨论其中一些结果以及它们的重要性。

1. 表达式变换

首先，我们注意到，在 2VL中的值是 *true*的一些表达式在 3VL中就不一定总是 *true*了。这里举几个例子，并加以说明。请注意，所举的例子是不可能穷尽所有情况的。

- 比较式 *x*=*x*不会必然得出 *true*

  在2VL里，任何值 *x*总是和它自己相等，但在 3VL里，*x*如果是 UNK，那么它就不等于它自己。

- 布尔表达式 *p* OR NOT(*p*)不会必然得出 *true*

  这里*p*也是布尔表达式。在 2VL里，*p* OR NOT(*p*)必然会等于 *true*，不管 *p*为何值。但在3VL里，如果 *p*的值是*unk*，则整个表达式的值是 *unk* OR NOT(*unk*)，即*unk* OR *unk*，简化为 *unk*，而不是 *true*。

  这个特殊的例子说明了 3VL有违反直觉的特性，这个特性举例说明如下：如果我们发出“求出在伦敦的所有供应商”的查询，接着发出“求出不在伦敦的所有供应商”的查询，然后对这两个结果做并，则我们不会必然获得所有的供应商。这其中可能漏掉了“所有可能在伦敦供应商”的结果 ㊀。

  关于这个例子的要点，当然是“城市是伦敦”和“城市不是伦敦”这两种情况在现

㊀ 按照这个论点，一个在 3VL里的值始终是 *true*的表达式 —— 即类似于 2VL表达式 *p* OR NOT(*p*)的3VL表达式 —— 是*p* OR NOT(*p*) OR MAYBE(*p*)。

实世界中是互斥的、并穷尽现实世界的所有可能，而数据库并不包含现实世界——其实，它仅仅包含它对现实世界的认知。关于对现实世界的认知存在三种情况，而不是两种；在这个例子里这三种情况是“被知道是伦敦的城市”、“被知道不是伦敦的城市”和“不被知道的城市”。当然（如[18.6]所说的），我们甚至显然不能问系统关于现实世界的问题，而只能问系统它所知道的用数据库中数据表达出来的关于现实世界的问题。这样，对范畴的混淆产生了这个例子的反直觉特性：用户是根据现实世界这个范畴来思考的，而系统是根据它对现实世界的认知来操作的。

注意：但对本文作者来讲，这样的范畴混淆是一个非常容易掉进去的陷阱。注意，在这本书前面章节中的每一个单一查询是按照“现实世界”表达的，而不是按照“对现实世界的认知”——当然这本书在这方面也不会特殊。

- 表达式 *r* JOIN *r* 不会必然给出 *r*

  在2VL里，一个关系 *r*和它自己进行连接总能得出原先的关系 *r*（即连接是幂等的）。然而在3VL里，一个任何一个位置是 UNK的元组不能和它自己做连接，因为不像联合，连接是基于“检索类型”（retrieval-type）的相等性判定，而不是“重复类型”的相等性判定。

- 交不再是连接的特殊情况

  这个事实同样来自于连接是基于检索类型的相等性判定而交是基于重复类型的相等性判定所造成的结果。

- A=B和B=C两个式子不能推出 A=C

  关于这个论点的进一步说明在下面的“部门和雇员的例子”小节里给出。

概括地说，许多在 2VL中有效的恒等式在 3VL中不再有效。这种情况造成一个如下的严重后果。通常，简单的恒等式如 *r* JOIN *r*≡*r*是各种各样变换规则的基础，而这些规则用来把查询转化成某些更有效的形式，如第 17章所介绍的。甚至不仅系统使用这些规则（当做优化的时候），用户也使用这些规则（当试图决定“更好”地描述某个查询的时候）。如果恒等式不再有效，那么规则也会失效。如果规则失效，那么变换也不再有效。如果变换无效，那么我们将会从系统那里得到错误的答案。

2. 部门和雇员例子

为了阐明不正确变换问题，我们来稍微详细地讨论一个特殊的例子（这个例子来自[18.9]；这里使用了关系演算而不是关系代数，但这并不是很重要）。假设给定一个简单的数据库，包含部门和雇员，见图 18-1。考虑下面的表达式

```
DEPT.DEPT# = EMP.DEPT# AND EMP.DEPT# = DEPT# ('D1')
```

(当然这可能是一个查询的一部分)；这里 DEPT和EMP暗指范围变量。对于数据库里单一的元组，这个表达式等于 *unk* AND *unk*，即*unk*。然而，一个“好的”优化器会注意到这个表达式形式是 *a*=*b* AND *b*=*c*，于是它会推出 *a*=*c*，并会在原先的表达式后边加上限制条件 *a*=*c*（如在第 17章第 17.4节里讨论的），这样得到

```
DEPT.DEPT# = EMP.DEPT# AND EMP.DEPT# = DEPT# ('D1')
                       AND DEPT.DEPT# = DEPT# ('D1')
```

这个修改过的表达式等于 *unk* AND *unk* AND *false*（对于数据库仅有的两个元组来说）。于是，下面的查询（举例）

```
EMP.EMP# WHERE EXISTS DEPT( NOT
      ( DEPT.DEPT# = EMP.DEPT# AND EMP.DEPT# = DEPT# ('D1')))
```

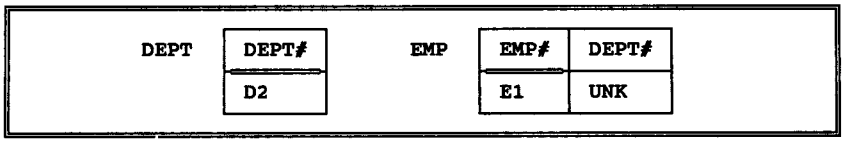


图18-1 部门和雇员数据库

如果按上述的理解进行“优化”，将返回雇员 *E1*。换句话说，这样的“优化”其实是不正确的。这样我们就知道了某些在 2VL中是正确且有用的优化在 3VL中不再正确。

注意那些为了把 2VL系统扩展到支持 3VL系统而造成的潜在影响。最好的情况是这样的扩展很可能需要对现有的系统进行重设计；最坏的情况是这样会导致错误。更普遍的是，要注意那些把支持 *n*值逻辑的系统扩展到支持（ *n*+1）值逻辑的系统的潜在影响，这里*n*是任何一个大于 1的数；对于每一个离散值 *n*都会产生类似难题。

3. 解释方法

现在，更进一步地来研究部门和雇员这个例子。既然雇员 E1在现实世界里没有某个对应的部门，那么让 UNK代表某个真实值，比如是 *d*。现在 *d*或是D1或不是。如果它是，那么原先的表达式

```
DEPT.DEPT# = EMP.DEPT# AND EMP.DEPT# = DEPT# ('D1')
```

等于（对于上述特定的值来说） *false*，因为第一项等于 false。另一方面，如果 *d*不是D1，则这个表达式还是等于（对于上述特定的值来说） *false*，因为第二项等于 false。换句话说，原先的表达式在现实世界里总是等于 *false*，不管UNK代表什么真实值。这样，在3VL里是正确的结果和在现实世界是正确的结果不是同样的事情！换句话说，三值逻辑的方法和现实世界是不一致的；即， 3VL看起来没有一种符合现实世界本身规律的解释方法。

注意：这个解释问题远远不是由空值和 3VL所造成的仅有的一个问题（关于其他问题的进一步讨论请见 [18.1~18.11]）。它甚至不是最基本的（见下面讨论）。然而，它也许是其中最具有实际意义的一个；其实，以作者观点来看，这只是一幕精彩的表演。

4. 再论谓词

假定关系 EMP仅仅含有两个元组，（E2，D2）和（E1，UNK）。第一个对应于这样的叙述“有一个标识为 E2、且在标识为 D2的部门里的雇员”。第二个对应于这样的叙述“有一个标识为 E1的雇员”（回忆一下，说一个元组“含有一个 UNK”其实真正地是说这个元组在这个可应用的位置上根本没有任何东西；这样，“元组”（E1，UNK） —— 如果它是元组的话，本质上这是一个可能有问题的的概念 —— 应当被认为仅仅是（ E1）的形式）。换句话说，这两个元组是两个不同谓词的事例，且这个“关系”根本不是一个关系，而是（不精确地说）两个有不同标题的不同关系的并。

也许有人会提出上述情况可以通过一个谓词来解决，这个谓词含有一个 OR：

有一个标识为E#、且在标识为D#的部门里的雇员 OR 有一个标识为E#的雇员。

然而，幸亏封闭世界假设（ the Closed World Assumption），使得这个关系不得不为所有的雇员 Ei都包含一个（ Ei，UNK）形式的元组！如果使这种拯救意图普遍化到几个“属

性”都“含有 UNK”的“关系”上，那几乎太可怕了（在任何情况下，有这样结果的“关系”将仍然不是一个关系——见下一段落）。

用另一种方法来考虑上述问题：如果一个给定关系的一个给定元组的一个给定属性的值“是UNK”，于是（重复一下）这个属性其实根本没有包含任何东西……于是可推出这个“属性”不是一个属性，这个“元组”不是一个元组，这个“关系”不是一个关系，且我们正在做的（不论它是其他什么东西）基础不再是数学上的关系理论。换句话说，UNK和3VL破坏了关系模型的整个基础。

18.4 空值和码

注意：我们现在放弃术语UNK（大部分），并由于历史原因回到更传统的术语“空值（null）”上。

不管前一节的内容如何，事实是现在大部分产品都支持空值和 3VL。这样的支持尤其对于码来说有重要的潜在影响。因此在这一节里，将简要地研究一下这些潜在影响的内容。

1. 主码

正如在第8.8节里所介绍的，关系模型（至少对于基变量是如此的）要求选出一个候选码来作为关系的主码。剩下的候选码，如果有的话，被称为可选码（alternate key）。于是，随同主码概念一起，模型在历史上曾包括下面的“元约束（metaconstraint）”或规则（实体完整性规则）：

- 实体完整性：基变量的主码的任何部分都不能为空值。

这个规则的理论基础来自于这样的解释：（a）基本关系中的元组表示现实世界的实体；（b）现实世界中的实体是用定义标识的；（c）因此它们在数据库中的副本也必须被标识；（d）在数据库中主码的值是被用来作这些标识的；（e）因此主码值一定不能“空缺”。

这样就产生了下述要点：

1) 首先，经常会认为“主码值必须是唯一的”，但其实并不这样（当然主码值必须唯一是正确的，但这个要求本身蕴涵在主码的基本定义里）。
2) 其次，注意这条规则仅适用于主码；可选码显然允许有空值。但如果 *AK*(可选码)是一个允许有空值的可选码，由于实体完整性规则，则不能选择 *AK*做为主码，这样一来，首先在哪种意义上 *AK*正好是一个“候选”码？另一种方法是，如果我们不得不说可选码也不能有空值，则完整性规则适用于所有的候选码，而不仅仅是主码。对这两种方法，所述规则看起来都存在一些错误。
3) 最后，注意实体完整性规则仅适用于基本关系变量：其他关系变量显然可以有允许是空值的主码。作为直接和明显的例子，考虑关系变量 *R*在任一允许有空值的属性上做投影。显然这规则违背了交换性原则（关于基本关系变量和导出关系变量）。我们认为，这是拒绝这个规则的根本理由（即使它不包含空值，我们也要拒绝这个概念）。

注意：假设同意丢弃空值的整个思想，取而代之用特殊值来表示空缺的信息（其实正

像在现实世界里所做的——稍后见第18.6节）。于是我们可能会想保留一个被修改过的实体完整性规则——“任何基变量的主码的任何部分都不准接受这样的特殊值”。——作为指导，而不是作为一个不可违背的法则（如许多更标准化的思想都作为指导，而不是作为不可违背的法则）。图18-2给了一个关于基本关系变量叫 SURVEY的例子（取自[5.7]）；它表示一个工资调查的结果，用来显示某一人群样本按照出生年份的平均、最大和最小工资（BIRTHYEAR是主码）。BIRTHYEAR值为“????”的元组表示拒绝回答“你什么时候出生？”这一问题的人。

SURVEY

BIRTHYEAR	AVGSAL	MAXSAL	MINSAL
1960	85K	130K	33K
1961	82K	125K	32K
1962	77K	99K	32K
1963	78K	97K	35K
....	...	...	...
1970	29K	35K	12K
????	56K	117K	20K

图18-2 基本关系变量SURVEY（样本值）

2. 外码

再一次考虑图18-1的部门和雇员数据库。也许大家没有注意到，图中关系变量 EMP的属性DEPT#是一个外码。因此很显然参照完整性规则需要一些改进，因为现在外码很明显必须要接受空值，而空值外码明显然违反了原先在第 8章所述的规则：

- 参照完整性（原先形式）：数据库不能包含任何不匹配的外码值。

其实，只要适当地扩展术语“不匹配的外码值”的定义，就可以保持所述的规则。为了明确性，我们定义在某些参照关系变量中一个不匹配的外码值是一个非空的外码值，而相关的被参照的关系变量中不存在这个外码值的相关候选码的匹配值。

由此产生了下述要点：

1) 将不得不把是否允许任一给定外码接受空值详细说明为数据库定义的一部分（当然，其实通常这也适用于属性，不管它们是否是某个外码的一部分）。

2) 外码可以接受空值的可能性会导致另一个参照动作的可能性，这个动作是 SET NULL，它或许会在一个外码的 DELETE或UPDATE规则中详细说明。比如：

```
VAR SP BASE RELATION{...}...
FOREIGN KEY{S#} REFERENCES S
                    ON DELETE SET NULL
                    ON UPDATE SET NULL
```

有了这样的说明，一个在供应商关系变量上的 DELETE操作会把所有相匹配的供货中的外码设置为空值，然后删除相应的供应商；同样，一个在供应商关系变量的属性S#上的UPDATE操作会把所有相匹配的供货中的外码设置为空值，然后删除相应的供应商。注意：当然仅仅对首先能接受空值的外码， SET NULL才可以被说明。

3) 最后，注意到适当的数据库设计可以避免在外码中有空值 [18.20]。比如，再一次来考虑部门和雇员。如果真的有可能不知道某些雇员的部门号，那么（如上节接近结尾所述的）更好的方法是根本不要在 EMP关系变量里包括 DEPT#，而要一个单独的关系变量 ED（比方说），这个关系变量的属性是 EMP#和DEPT#，表示一个指定

的雇员在一个指定的部门这样的事实。于是，某个雇员有一个不知道的部门的事实可以用从关系变量 ED中删除这个雇员元组来表示。

18.5 外连接

在这一节里，我们将简要讨论一下被称为外连接的操作（见 [18.3~18.4]，[18.7]，和 [18.14~18.16]）。外连接是常规连接或内连接操作的一种扩展形式。它不同于内连接的是，若一个关系中的元组在另一个关系中没有相匹配的元组，则这些元组会在结果中出现，并在另一个关系的其他属性位置放上空值，而不是像通常那样被忽略。它不是一个基本操作；比如，下述表达式被用来构造供应商和供货在供应商号码上的外连接（为了这个例子，假设“ NULL ”是一个合法的标量表达式）：

```
( S JOIN SP)
  UNION
( EXTEND ( ( S{S#} MINUS SP{S#} ) JOIN S )
                              ADD NULL AS P#,NULL AS QTY )
```

这个表达式的结果包括没有供应零件的供应商元组，并扩展地在 P#和QTY位置上放置空值。

更仔细一点来研究这个例子。参考图 18-3。在这个图中，上面部分表示一些样本值，中间部分表示相应的内连接，而下面部分表示相应的外连接。如这个图所显示的，内连接在没有供应零件的供应商上（这个例子里是供应商 S5）“ 空缺了信息 ”（不精确地讲），然而外连接“ 保留 ”了这样的信息。其实，这个区别就是外连接的要点所在。

S

S#	SNAME	STATUS	CITY
S2	Jones	10	Paris
S5	Adams	30	Athens

SP

S#	P#	QTY
S2	P1	300
S2	P2	400

规则(内)连接

S#	SNAME	STATUS	CITY	P#	QTY
S2	Jones	10	Paris	P1	300
S2	Jones	10	Paris	P2	400

“ 空缺 ”了供应商 S5 的信息

外连接

S#	SNAME	STATUS	CITY	P#	QTY
S2	Jones	10	Paris	P1	300
S2	Jones	10	Paris	P2	400
S5	Adams	30	Athens	UNK	UNK

“ 保留 ”了供应商 S5 的信息

图18-3 内连接和外连接（例子）

外连接要解决的问题 —— 即内连接有时会“ 空缺信息 ”的事实 —— 当然是一个很重要的问题。于是某些作者提出系统应当提供直接的、清楚的外连接支持，而不是要求用户通过非常复杂的陈述来获得想要的结果。尤其 Codd认为外连接是关系模型中的固有部分 [5.2]。然而，我们自己不认可这种观点，因为有下述的一些原因：

- 首先，这个操作包含空值，而我们有许多好理由来反对空值。
- 第二，外连接有好多种形式 —— 左、右和完全外 Θ连接，以及左、右和完全外自然连接（“ 左 ”连接保留了来自第一个操作数的信息，“ 右 ”连接保留了来自第二个操作数的信息，而“ 完全 ”连接则两者兼有；图 18-3的例子是一个左连接 —— 精确地

说是左外自然连接）。进一步讲，这里不存在直接的方法从外 ⊖连接导出外自然连接[18.7]。因此，难以确定究竟哪一种外连接需要明确的支持。

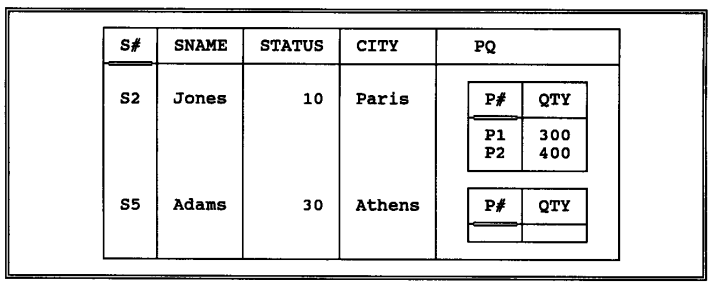
- 其次，图18-3的例子远远不能把外连接问题叙述完整。其实正如 [18.7]所述的，外连接具有许多有害的特点，这些特点加起来使在现有语言 —— 尤其是SQL —— 加上外连接是困难的。一些 DBMS试图去解决这个问题，但最终以失败告终（即它们被这些有害的特点所困扰）。关于这个论点更详细的论述请参见 [18.7]。
- 最后，可以说关系值属性（ relation-valued attribute）提供了解决这个问题的一个方法 —— 一个不包含空值也不包含外连接的方法，并且是（作者的观点）一个更完美的解决方法。比如给定如图 18-3上面部分的样本数据值，那么下面的表达式

```
WITH ( S RENAME S# AS X ) AS Y
EXTEND Y ADD ( SP WHERE S#=X ) AS PQ
```

会产生如图18-4所示的结果。

在图18-4中，尤其要注意供应商 S5所提供的零件空集是用一个空集表示出来的，而不是（像图18-3）使用一些古怪的“ null ”。用一个空集表示一个空集不失为一个好主意。其实，如果适当地支持关系值属性，那么就根本没有必要使用外连接。

再来讨论一下这个论点：怎样来解释出现在外连接结果中的空值？比如它们在图 18-3的例子中意味着什么？它们当然既不意味着“ 值不知道 ”，也不意味着“ 值不可应用 ”；其实，能给出任何有逻辑性的唯一精确的解释是“ 值是空集 ”。有关进一步论述请参见[18.7]。



S#	SNAME	STATUS	CITY	PQ
S2	Jones	10	Paris	P# QTY: P1 300; P2 400
S5	Adams	30	Athens	P# QTY:

图18-4 保留供应商S5的信息（一种更好的方法）

在结束本节内容之前，我们要指出的是，关系代数的其他操作也可以定义类似“ 外 ”连接的形式 —— 尤其是并、交和差操作 [13.6] —— 并且Codd认为至少其中一个即外并应是关系模型的一部分 [5.2]。这样的操作允许并（等）在两个甚至是不同类型的关系上进行操作；它们使每个操作数都包括对它来说是古怪的属性（这样，这两个操作数现在就成为相同的类型了），并在这些增加的属性上放置空值，然后变成普通的并，交或差 ⊖。然而，由于下述的原因我们不打算对这些操作进行详细的论述：

- 外交操作肯定会返回一个空的关系，除非原来的关系具有相同的类型，若是这样，就退化为普通的交了。
- 外差操作肯定回返回它的第一个操作数，除非原来的关系具有相同的类型，若是这

⊖ 这个说明引用了原先定义的操作 [13.6]。[5.2]在某些地方改变了这个定义；关于细节请参考 [5.2]。

样，就退化为普通的差了。

- 外并操作的主要的问题是关于解释的问题（它们比外连接所造成的问题更糟糕）。关于进一步论述请参见 [18.2]。

18.6 特殊值

我们已经看到，空值破坏了关系模型。但值得一提的是，没有空值的关系模型已经好好使用了十年！——模型在1969年第一次被定义 [5.1]，空值直到1979年才被加进来 [13.1]。

因此，假设——像18.4节所提议的——我们赞成丢弃空值的整个思想，取而代之使用特殊值来表示空缺的信息。注意，使用特殊值正是我们在现实世界中所做的。比如，在现实世界里，如果因某些原因不知道某个雇员的工作小时 ⊖，可能会用“？”来表示这个值。这样，当没有实际值可用的时候，通常的方法是可以简单地使用一个与这个属性的所有实际值不同的特殊值。注意，特殊值必须是一个来自可应用的域；因此在“工作小时”这个例子里，属性 HOURS_WORKED的类型不仅仅是整型，而应当是整型加上特殊值。

这里首先得承认上述方案也不是完美的，但它的最大优点在于它不会破坏关系模型的逻辑基础。因此，在本书的其余部分，我们将完全忽略对空值的支持（除非在有关 SQL的讨论中，会涉及到空值的问题）。关于特殊值方案的详细论述请参见 [18.12]。

18.7 SQL的支持

SQL对空值和3VL的支持遵循上面 18.1~18.5节里所述方法的主要内容。这样，比如，当SQL在某个表 *T*上应用 WHERE子句时，它会删除所有对于这个 WHERE子句中的表达式求出的值是 *false*和*unk*（即非 *true*）的元组。同样地，当 SQL在某个“分组表（grouped table）”*G*（见附录 A）上应用一个 HAVING子句时，它会删除所有对于这个 HAVING子句中的表达式求出的值是 false和unk（即非 *true*）的*G*的分组。因此，在接下来的部分中，我们将仅限于讨论对 SQL本身来说是特殊的某些 3VL特点，而不是像先前所述的 3VL方法的本质部分。

注意：SQL关于空值支持的整个蕴涵和延伸是很复杂的。有关更多的信息，请参见正式标准文档 [4.22]或[4.19]中详细的辅导教程。

1. 数据定义

如第5章第5.5节所述的，基表中的列通常有一个相应的默认值，而且这个默认值经常被显式或隐式地定义为是空值。甚至，基表中的列经常允许取空值，除非存在一个完整性约束——可能仅是 NOT NULL——这个约束明白地禁止这些列不允许为空值。空值的表示和实现有关；但是，系统必须做到能够区分空值和非空值（即使比较式“ null ≠ *x*”也不应给出 *true*！）。

2. 表表达式

回忆一下第 7章第7.7节，在SQL/92标准里给 SQL增加了显式连接支持。如果在关键字 JOIN的前面加上 LEFT、RIGHT或FULL（每种情况后面可有可无 OUTER词），那么所述的

⊖ 注意到我们没有做的一件事是使用一个空值。在现实世界里没有一件事能作为空值。

连接就是一个外连接。下面有几个例子：

```
S LEFT JOIN SP ON S.S#=SP.S#
S LEFT JOIN SP USING(S#)
S LEFT NATURAL JOIN SP
```

这三个表达式实际上都是相等的，除了第一个得出一个有两个相同列（都是 S#）的表，第二个和第三个得出一个只有一列的表。

SQL也支持一个外并的近似方法，称为并连接。关于它的详细论述已超出本书范围。

3. 条件表达式

如第8章所述的，条件表达式就是一种布尔表达式（在附录 A里有它们的详细论述）。显然，这些表达式是受空值和 3VL影响最厉害的 SQL部分。我们在这里作几个适当的说明。

- 关于空值的计算： SQL提供两个特殊的操作， IS NUL和IS NOT NULL，来计算空值的存在或不存在。语法如下：

```
<row constructor> IS [NOT] NULL
```

（关于 <row constructor>的详细内容见附录 A）。这里有一个陷阱：两个表达式 *r* IS NOT NULL和NOT(*r* IS NULL)是不相等的。关于解释请参见 [4.19]。

- 关于*true*、*false*和*unknown*的计算：如果 *p*是条件表达式，于是下述也是条件表达式：

```
p IS [NOT] TRUE
p IS [NOT] FALSE
p IS [NOT] UNKNOWN
```

这些表达式的含义可用下述真值表来表述：

p	*true*	*false*	*unk*
p IS TRUE	*true*	*false*	*false*
p IS NOT TRUE	*false*	*true*	*true*
p IS FALSE	*false*	*true*	*false*
p IS NOT FALSE	*true*	*false*	*true*
p IS UNKNOWN	*false*	*false*	*true*
p IS NOT UNKNOWN	*true*	*true*	*false*

表达式*p* IS NOT TRUE和NOT *p*是不相等的。注意：表达式 *p* IS UNKNOWN和MAYBE（*p*）含义相同。

- MATCH条件：<match condition>的语法 —— 见附录 A —— 包括一个 PARTIAL与FULL选项（这并没有出现在附录 A里，也未加讨论）；若出现 null，这些选项会影响结果：

```
<row constructor> MATCH [UNIQUE]
                  [PARTIAL | FULL] (<table expression>)
```

这里存在 6种情况，取决于（ a）UNIQUE选项是否特别说明；且（ b）PARTIAL或FULL是否有说明（若有，则是哪一个）。详细情况会是是相当复杂的，且已超出了本书的范围。关于进一步的讨论参见 [4.19]。

- EXISTS条件：参见 [18.6]的讨论。

4. 标量表达式

- “文字（literal）”：关键字 NULL可以被用作空值的文字表示（如在 INSERT语句里）。然而注意，这个关键字并不能在所有文字可以出现的环境里出现；如标准所述的，

“不存在空值的 <literal>，虽然在某些地方使用关键字 NULL来表示这里需要一个空值”[4.22]。这样，就不能清楚地指明 NULL作为一个简单比较式的操作数——例如，“WHERE X=NULL”是非法的（正确的形式是“ WHERE X IS NULL”）。

- COALESCE（合并）：SQL中的COALESCE类似于前面提到的 IF_UNK操作符（见18.2节）。
- 聚集操作符：SQL聚集操作符SUM，AVG等的计算和18.2节所述的标量操作符的规则并不一致，它们在计算中忽略了参数中任何的 空值（除了COUNT(*)以外，它把空值看成是一般值）。同样，如果这样的操作符的参数碰巧是空集，则 COUNT会返回零；其他操作符都返回空值（如第7章所述的，后者在逻辑上是错误的，但 SQL就是如此定义的）。
- “标量子查询（ Scalar subquery）”：如果一个标量表达式是一个在括号里的表表达式——比如，（SELECT S. CITY FROM S WHERE S.S#='S1'）——一般地讲，这个表表达式会求得一个单列单行的表。于是这个标量表达式的值其实精确地来说是一个只包含一个标量值的表。但是，如果这个表表达式求得的值是一个根本没有行的单列表，则SQL把这个标量表达式的值定义为空值。

5. 码

SQL中空值和码的相互关系概括如下：

- 候选码：假设 *C*是某个基本表的某个候选码 *K*的一个组成列。如果 *K*是主码，则 SQL不会允许*C*包含任何空值（换句话说，它应当遵守实体完整性规则）。然而，若 *K*不是主码，则SQL会允许*C*包含任何数目的空值（当然也允许包含任何数目的非空值）。
- 外码：在空值情况下如何确定外码与相应主码值的关系，规则是相当复杂的；它们基本上和MATCH条件的细节一样（见先前部分），有关细节这里不再详述。

  空值对于参照行为——CASCADE、SET NULL，等——也有潜在影响，在 ON DELETE和ON UPDATE子句中有详细说明（SET DEFAULT也以明显的解释得到支持）。同样，这些细节也是相当复杂的，并且超出了本书范围：关于细节请参见 [4.19]。

6. 嵌入式SQL

- 指示变量：考虑下面的一个嵌入式 SQL“SELECT”（第4章的重复例子）；

```
EXEC SQL SELECT STATUS, CITY
         INTO    :RANK, :CITY
         FROM    S
         WHERE   S#= :GIVEN_S#;
```

假设有可能某些供应商的 STATUS的值是空值。如果被选中的 STATUS是空值，则上面的SELECT语句将会失败（SQLSTATE将会被置为异常值 22002）。通常，如果存在被选取的值也许是空值的可能，则用户应当给目标变量指定一个指示变量，如下面所示：

```
EXEC SQL SELECT STATUS, CITY
         INTO    :RANK INDICATOR :RANK_IND, :CITY
         FROM    S
         WHERE   S#= :GIVEN_S#;
IF RANK_IND=-1 THEN  /*STATUS是null*/  ... ; END IF;
```

如果被选取的值是空值且指示变量已被指定，则这个指示变量被置为值- 1。对于通常的目标变量所造成的影响依执行而定。

- 排序：ORDER BY子句在游标定义（cursor definition）中被用来对表表达式结果里的行进

行排序（当然，在交互式的查询中也会用到）。这样产生了下述问题：如果标量值 *A*或*B* 是空值（或都是），则*A*和*B*的相对次序是什么？SQL关于这个问题的回答如下：

1) 对于排序，所有的空值被认为是互相相等的。
2) 对于排序，所有的空值被认为大于所有的非空值或小于所有的非空值（到底应用那种情况视实现而定）。

18.8 小结

我们已经讨论了关于信息空缺的问题和目前流行的 —— 虽很不好 —— 一种基于空值和三值逻辑(3VL)的关于这个问题的解决方法。我们强调这样的观点，即空值不是一个值，虽然通常好像它是一个值(比方说，某个特殊元组的某个特殊属性值“是空值”)。有一个操作数是空值的任何比较式的值等于“第三个真值” *unknown*(缩写为*unk*)，因此有了三值逻辑。同时提到，至少在概念上，存在许多不同种类的空值，并把 UNK作为“未知值”这个类型的一个简写。

然后我们研究了 UNK和3VL在布尔表达式、量词EXISTS和FORALL、计算表达式、关系操作符乘积、选择、投影、并、交和差以及更新操作符 INSERT和UPDATE上的潜在影响。介绍了操作符 IS_UNK（用来测试 UNK）、IF_UNK（用来把 UNK转化为非 UNK值）和 MAYBE（用来把*unk*转化为*true*）。讨论了关于 UNK的等同问题，并指出了 UNK和*unk*不是同一回事。

接下来，我们研究了上述思想所造成的一些后果。首先，说明了某些恒等式在 3VL中不再有效 —— 即在 2VL中有效的等价式在 3VL中不再有效。因此，用户和优化器可能在表达式变换中产生错误。而且，即使这样的错误不会发生， 3VL也会因它和现实世界不符这个严重问题而带来麻烦 —— 即对于3VL是正确的结果某些时候在现实世界中是不正确的。

于是我们继续讨论了在主码和外码上的空值的潜在影响（特别提到了实体完整性规则和修改过的参照完整性规则）。接下来介绍了外连接。我们自己不提倡直接支持这个操作（至少不像通常理解的那样），因为我们相信有更好的方法来解决外连接要解决的问题 —— 特别地，我们推荐一种使用关系值属性的解决方法。简要地提到了其他“外”操作的可能性，特别是外并。

接下来，我们研究了上述思想的 SQL支持。SQL对空缺信息的处理主要是基于 3VL，但它设法包括了大量附加的复杂东西，大部分都超过了本书的范围。其实， SQL是在设法引入一些附加的缺点，这些缺点是 3VL本身造成的 [18.6，18.1]。更糟的是，这些附加的缺点成了优化的抑制剂（在第 17章结尾处提到过）。

最后小结如下：

- 大家会注意到，我们仅仅涉及到由空值和 3VL所造成的问题的表面。但我们已经尽力用足够多的理由来说明从 3VL方法中得到的“好处”是值得怀疑的。
- 我们也应当使大家明白，即使你不相信关于 3VL本身的问题，但仍建议大家应避开它在SQL上所造成的相关东西，因为有上面所述的“附加的缺点”。
- 我们给 DBMS用户的建议是要完全忽略厂家的 3VL的支持，而使用一个遵守规则的“特殊值”方案（从而牢牢地坚持二值逻辑）。这个方案在 [18.12]里有详细论述。
- 最后，我们重复下述来自 18.6节的基本要点：如果 —— 不精确地讲 —— 一个给定关系的一个给定元组的一个给定属性的值“是空值”，则这个属性的位置其实根本没有包含任何东西……这就推导出这个“属性”不是一个属性，这个“元组”不是一

个元组，这个“关系”不是一个关系，且我们正在做的（不管它会是什么）基础不再是数学上的关系理论。

练习

18.1 若A=6，B=5，C=4，且D是UNK，请说出下列表达式的真值：

a. `A = B OR ( B > C AND A > D )`

b. `A > B AND ( B < C OR IS_UNK ( A - D ) )`

c. `A < C OR B < C OR NOT ( A = C )`

d. `B < D OR B = D OR B > D`

e. `MAYBE ( A > B AND B > C )`

f. `MAYBE ( IS_UNK ( D ) )`

g. `MAYBE ( IS_UNK ( A + B ) )`

h. `IF_UNK ( D, A ) > B AND IF_UNK ( C, D ) < B`

18.2 假设关系*r*正好包含下列元组：

```
(  6,   5,   4  )
( UNK,  5,   4  )
(  6,  UNK,  4  )
( UNK, UNK,  4  )
( UNK, UNK, UNK )
```

像本章内容所述的，假设（a）这三个属性以上面所示的顺序从左到右依次叫作 *A*，*B*和*C*，且（b）每一个属性都是 INTERGER类型。若 *V*是*r*上的范围变量，请说出下列表达式的真值：

a. `EXISTS V ( V.B > 5 )`

b. `EXISTS V ( V.B > 2 AND V.C > 5 )`

c. `EXISTS V ( MAYBE ( V.C > 3 ) )`

d. `EXISTS V ( MAYBE ( IS_UNK ( V.C ) ) )`

e. `FORALL V ( V.A > 1 )`

f. `FORALL V ( V.B > 1 OR IS_UNK ( V.B ) )`

g. `FORALL V ( MAYBE ( V.A > V.B ) )`

18.3 严格地说，IS_UNK操作符是多余的。为什么？

18.4 在[13.6]里Codd提出了一些（不是所有）关系代数操作符的“ maybe”版本。比如，maybe-restrict和通常的 restrict 不同，它返回这样的一个关系，这个关系包含在选择条件上求出的值是 *unk*而不是 *true*的元组。然而这样的操作符严格地讲是多余的。为什么？

18.5 在二值逻辑（2VL）里，正好存在两个真值， *true*和*false*。结果正好有 4个可能的单元逻辑操作符 —— 一个把 *true*和*false*都变换为 *true*，还有一个把两者都变换为 *false*，一个把*true*变换为*false*，并反之亦然（当然这个操作符是 NOT），还有一个是使两者都不会变化。正好存在 16个可能的二元操作符，如下表所示：

A	B														
t	t	t	t	t	t	t	t	t	f	f	f	f	f	f	f
t	f	t	t	t	t	f	f	f	f	t	t	t	f	f	f
t	t	t	t	f	f	t	t	f	f	t	t	f	f	t	f
t	f	t	f	t	f	t	f	t	f	t	f	t	f	t	f

证明，2VL中的所有4个单元操作符和16个二元操作符可以用NOT、AND和OR的适当组合来表示（因此就没有必要明确地支持所有20个操作符）。

18.6 在3VL里有多少逻辑操作符？在4VL里会是多少？更一般地，在*n*VL里会是多少？

18.7 (来自[18.5])图18-5表示常用的供应商和零件数据库里的一些样本值，但稍微有点变化（这个变化是，关系SP包含一个新的供货号属性SHIP#，且这个关系中的属性P#现在“允许为UNK”；关系P和这个练习没有关系，故这里被省略）。考虑下面的关系演算查询

```
S WHERE NOT EXISTS SP ( SP.S# = S.S# AND
                        SP.P# = P# ( 'P2' ) )
```

（这里S和SP是隐含的范围变量）。下面哪一个（若有）是这个查询的正确解释？

a. 取出不供应P2的供应商。

b. 取出不知道供应P2的供应商。

c. 取出已知道不供应P2的供应商。

d. 取出已知道不供应P2或不知道供应P2的供应商。

18.8 为SQL基本表设计一个物理表示方案，这些基本表允许包含空值。

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London

SP

SHIP#	S#	P#	QTY
SHIP1	S1	P1	300
SHIP2	S2	P2	200
SHIP3	S3	UNK	400

图18-5 供应商和零件的一个变种

参考文献和简介

18.1 E. F. Codd and C. J. Date: “Much Ado about Nothing,” in C. J. Date, *Relational Database Writngs 1991-1994*. Reading, Mass.: Addison-Wesley (1995).

Codd也许是为了处理空缺信息而把空值和3VL作为基础的最初提倡者。这篇文章包含了Codd和本书作者在这个问题上所进行的辩论。它包含下列有趣的评论：“如果不存在空缺值，数据库管理将会变得容易”（Codd）。

18.2 Hugh Darwen: “Into the Unknown,” in C. J. Date, *Relational Database Writings 1985-1989*. Reading, Mass.: Addison-Wesley (1990).

提出了一些关于空值和3VL的另外的问题，下述也许是其中最值得研究的问题：如果（如第5章练习5.9答案所述的）TABLE_DEE相应于*true*而TABLE_DUM相应于*false*，且TABLE_DEE和TABLE_DUM是唯一可能的零级（degree zero）关系，那么什么相应于*unk*？

18.3 Hugh Darwen: “ Outer Join with No Nulls and Fewer Tears,” in C. J. Date and Hugh Darwen, *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

提出“外连接”的一个简单变种，这个变种不涉及空值，并解决了许多外连接应该解决的问题。

18.4 C. J. Date: “ The Outer Join,” in *Relational Database: Selected* Writings. Reading, Mass.: Addison-Wesley (1986).

深度分析了外连接问题，并提出了一个建议用来支持如 SQL这样的语言中的外连接操作。

18.5 C. J. Date: “ NOT Is Not “ Not ”! (Notes on Three-Valued Logic and Related Matters),” in *Relational Database Writings* 1985-1989. Reading, Mass: Addison-Wesley (1990).

假设 *X*是一个 BOOLEAN类型的变量。那么 *X*必须是 *true*、*false*或*unk*中的一个。这样，语句“ *X* is not *true* ”意味着 *X*的值是 *unk*或*false*。相反，语句“ *X* is NOT *true* ”意味着 X的值是 false（见 NOT的真值表）。3VL里的 NOT不是自然语言里的 not……这个事实已经使有些人（包括 SQL标准的设计者）困惑，并无疑还会继续下去。

18.6 C. J. Date: “ EXISTS Is Not “ Exists ”! (Some Logical Flaws in SQL),” in *Relational Database Writings 1985-1989*. Reading, Mass: Addison-Wesley (1990).

表明SQL EXISTS操作符和 3VL里的有关存在量词不是同一回事，因为它求出的值总是等于 *true*或*false*，从来不会是 *unk*，甚至当 *unk*在逻辑上是正确的值的时候。

18.7 C. J. Date: “ Watch Out for Outer Join,” in C. J. Date and Hugh Darwen, *Relational Database Writing 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

本章第 18.5节提到这样的事实，外连接有一些“令人讨厌的特点”。这篇论文把这些特点总结如下：

1) **外Θ连接不是笛卡尔积的一个选择。**
2) **外Θ连接上没有选择。**
3) **“** $A \leq B$ **”和“** $A<B$ OR $A=B$ **”不是同一回事（在 3VL里）。**
4) **Θ比较操作符不可传递（在 3VL里）。**
5) **外自然连接不是外等值连接的一个投影。**

这篇论文继续考虑为 SQL SELECT-FROM-WHERE结构增加外连接将会怎么样。它叙述了上述令人讨厌的特点会推导出下列情况：

1) **WHERE子句的扩展操作不能工作。**
2) **外连接的 AND操作和选择操作不能工作。**
3) **不能在 WHERE子句里表达连接条件。**
4) **多于两个关系的外连接若没有嵌套表达式将不能被明确地表达出来。**
5) **SELECT子句（单独）的扩展操作不能工作。**

这篇论文也说明了许多现有产品与这些因素的冲突。

18.8 C. J. Date: “ Composite Foreign Keys and Nulls,” in C. J. Date and Hugh Darwen, *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

复合外码允许整个或部分是空值吗？这篇论文讨论了这个问题。

18.9 C. J. Date: “ Three-Valued Logic and the Real World,” in C. J. Date and Hugh Darwen,

Relational Database Writings 1989-1991. Reading, Mass.: Addison-Wesley (1992).

18.10 C. J. Date: “ Oh No Not Nulls Again,” in C. J. Date and Hugh Darwen, *Relational Database Writings 1989-1991*. Reading, Mass.: Addison-Wesley (1992).

这篇论文提供了很多的大家也许想知道的关于空值的东西。

18.11 C. J. Date: “ A Note on the Logical Operators of SQL,” in *Relational Database Writings 1991-1994*. Reading, Mass.: Addison-Wesley (1995).

由于3VL有三个真值 *true*、*false*和*unk*（这里分别缩写为 *t*、*f*和*u*），所以存在3*3*3=27个可能的单元操作符，因为每个可能的输入 *t*、*f*和*u*可以映像为三个可能输出*t*、*f*和*u*中的一个。因此有 3^9=19 683个可能的二元3VL操作符，如下面的表格所示：

	t	u	f
t	t/u/f	t/u/f	t/u/f
u	t/u/f	t/u/f	t/u/f
f	t/u/f	t/u/f	t/u/f

其实更一般的情况是， *n*值逻辑包括 *n*的*n*次方幂个单元操作符和 *n*的n^2次方幂个二元操作符：

	monadic operators	*dyadic operators*
2VL	4	16
3VL	27	19,683
4VL	256	4,294,967,296
...	.....	.....
nVL	(n)**(n)	(n)**(n^2)

于是，对任何 *n*VL（*n*>2），会产生下述问题：

- 一个合适的原语操作符集合是什么？（比如，集合 {NOT，AND} 或{NOT，OR}是2VL的一个适合的原语集合。）
- 一个有用的操作符集合是什么？（比如，集合 {NOT，AND，OR}是2VL的一个有用集合。）

这篇论文说明了SQL标准（在一个很宽松的解释方式下）至少直接或不直接地支持所有19 710个操作符。然而应当强调，在逻辑上操作符是对谓词和命题进行操作的，而在SQL里（甚至在上述宽松的解释方式下）操作符仅仅是对命题进行操作。

18.12 C. J. Date: “ Faults and Defaults” (in five parts), in C. J. Date, Hugh Darwen, and David McGoveran, *Relational Database Writings 1994-1997*. Reading, Mass.: Addison-Wesley (1998).

叙述一个解决空缺信息问题的系统方法，这个方法是基于特殊值和 2VL而不是空值和3VL。这篇论文有力地辩明特殊值是我们在现实世界所使用的东西 —— “ 在现实世界里不存在空值这样的事物 ” —— 于是数据库系统在这个方面应当像现实世界处理问题一样来处理问题。

18.13 Debabrata Dey and Sumit Sarkar: “ A Probabilistic Relational Model and Algebra,” *ACM TODS 21*, No. 3 (September 1996).

提出一个基于概率理论而不是空值和 3VL的、用来解决“ 数据值不确定性 ” 问题的方法。“ 或然性关系模型 ” 是传统关系模型的一个可兼容的扩展。

18.14 César A. Galindo-Legaria: “ Outerjoins as Disjunctions,” Proc. 1994 ACM SIGMOD Int. Conf.

on Management of Data, Minneapolis, Minn. (May 1994).

一般来说，外连接不是一个关联的操作符 [18.4]。这篇论文准确地描述了外连接的特点，这些外连接有的是关联的，有的不是关联的，并为每种情况提出了实现方案。

18.15 César Galindo-Legaria and Arnon Rosenthal: " Outerjoin Simplification and Reordering for Query Optimization," *ACM TODS 22*, No. 1 (March 1997).

介绍了包含外连接的表达式的 " 一个完整的变换规则集合 "。

18.16 Piyush Goel and Bala Iyer: " SQL Query Optimization: Reordering for a General Class of Queries," Proc. 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada (June 1996).

像[18.15]一样，这篇论文讨论了包含外连接的表达式的转换："[我们]提出了一个方法，用来重新组织 [一个]包含外连接、和……聚集的 SQL查询…… [我们]标识了一个强有力的[在这样的重组织中起辅助作用]的原语，[我们]把这个原语称为通用化查询。

18.17 I. J. Heath: IBM internal memo (April 1971).

这篇论文介绍了 " 外连接 " 术语（和概念）。

18.18 Ken-Chih Liu and Rajshekhar Sunderraman: " Indefinite and Maybe Information in Relational Databases," *ACM TODS 15*, No. 1 (March 1990).

包含一组正式的建议，这些建议被用来扩展关系模型以便处理 maybe信息（比如，" 零件 P7可能是黑色 "）和不确定性的或离散的信息（比如，" 零件 P8或零件 P9是红色的 ")。介绍了 I表，用来表示正常（确定的）信息， maybe信息和不确定性信息。扩展了选择、投影、乘积、并、交和差操作符，以便对 I表进行操作。

18.19 David Maier: *The Theory of Relational Databases*. Rockville, Md.: Computer Science Press (1983).

18.20 David McGoveran: " Nothing from Nothing" (in four parts), in C. J. Date, Hugh Darwen, and David McGoveran, *Relational Database Writings 1994-1997*. Reading, Mass.: Addison-Wesley (1998).

这篇论文由四部分构成。第一部分叙述了在数据库中逻辑的至关重要的角色。第二部分明确地表述了为什么必须是二值逻辑，为什么试图使用三值逻辑（ 3VL）是被误导的。第三部分研究了三值逻辑（ 3VL）应当要 " 解决 " 的问题。最后，第四部分介绍了一组对这些问题注重实效的解决方案，这些方案不包含 3VL。

18.21 Nicholas Rescher: *Many-Valued Logic*. New York, N.Y.: McGraw-Hill (1969).

标准文本。

部分练习答案

18.1 a.*unk*. b.*true*. c.*true*. d.*unk*（注意这个答案的违反直觉性）； e.*false*; f.*false*（注意 IS_UNK永远不会返回 *unk*）；g. *false*; h.*true*。

18.2 a. *unk*. b. *unk*. c. *true*. d. *false*. e. *unk*. f. *true*. g. *false*.

18.3 因为下面的恒等式：

```
IS_UNK ( x )  =  MAYBE ( x = x )
```

18.4 因为（比如）“ MAYBE_RESTRICT *r* WHERE *p* ”和“ *r* WHERE MAYBE(*p*) ”是一样的。

18.5 这四个单元操作符可以定义如下（ *A*是单一操作数）：

```
A
NOT(A)
A OR NOT(A)
A AND NOT(A)
```

这16个二元操作符可以定义如下（ *A*和*B*是两个操作数）：

```
A OR NOT(A) OR B OR NOT(B)
A AND NOT(A) AND B AND NOT(B)
A
NOT(A)
B
NOT(B)
A OR B
A AND B
A OR NOT(B)
A AND NOT(B)
NOT(A) OR B
NOT(A) AND B
NOT(A) OR NOT(B)
NOT(A) AND NOT(B)
(NOT(A) OR B) AND (NOT(B) OR A)
(NOT(A) AND B) OR (NOT(B) AND A)
```

顺便提及，想了解为什么不同时需要 AND和OR，请观察下面的例子：

```
A OR B  =  NOT ( NOT ( A ) AND NOT ( B ) )
```

18.6 见[18.11]中的注释。

18.7 c. 更详细的讨论请见参考 [18.5]。补充练习：给出解释 b的一个关系演算式子。

18.8 我们简要介绍在 DB2中的表示方法。在 DB2里，一个可以接受空值的列在数据库里物理上用两列来表示，数据列自己和一个 1字节宽的标志列，这个标志列作为真实数据列的前缀。一个值是 1的标志表示相应的数据列值被忽略（即作为空值）；一个值是 0的标志表示相应的数据列值是真实的值。但列的标志（当然）对用户来说是隐藏的。