目　　录

第一章　概述 …… 1－1
1.1　什么是数据库系统 …… 1－1
1.2　数据库系统的特点 …… 1－2
1.3　数据库系统的典型结构 …… 1－6
1.4　数据语言 …… 1－7
1.4.1 数据描述语言 …… 1－7
1.4.2 数据操作语言 …… 1－8
1.5　数据库管理系统 …… 1－9
1.5.1 数据字典（*Data dictionary*）…… 1－10
1.5.2 数据库管理员（*DBA*）…… 1－11
1.5.3 用户访问数据库的过程 …… 1－12
1.6　实体——联系方法 …… 1－14
1.7　数据模型 …… 1－17

第二章　存贮结构 …… 2－1

2.1　引言 …… 2－1
2.1.1 文件的基本概念 …… 2－1
2.1.2 数据库操作速度的估计 …… 2－4
2.1.3 指示器（*Pointer*）…… 2－5
2.1.4 关键字（*Keys*）…… 2－5
2.1.5 钉定（*Pinned*）和未钉定的记录 …… 2－6
2.1.6 文件结构概述 …… 2－7
2.2　顺序文件 …… 2－10
2.2.1 如何确定关键字值的顺序 …… 2－10
2.2.2 顺序文件的存贮组织 …… 2－10
2.2.3 顺序文件的查找 …… 2－11

2.3 随机结构之一 —— 散列方法 …… 2—12
2.3.1 散列方法的简要回顾 …… 2—12
2.3.2 散列文件的设计 …… 2—15
2.3.3 可扩充的散列 …… 2—16
2.4 随机结构之二 —— 索引结构 …… 2—21
2.4.1 索引顺序文件 …… 2—21
2.4.2 索引无序文件 …… 2—21
2.4.3 索引的组织 …… 2—22
2.4.4 索引文件的查找 …… 2—25
2.5 B—树 …… 2—25
2.5.1 二叉树 …… 2—25
2.5.2 B—树 …… 2—27
2.5.3 B+树 …… 2—29
2.5.4 一个B+树实例 …… 2—31
2.6 变长记录文件 …… 2—33
2.7 倒排文件 …… 2—35
第三章 关系方法 …… 3—1
3.1 关系及基本术语 …… 3—1
3.2 关系运算 …… 3—3
3.2.1 关系代数 …… 3—3
3.2.2 元组关系演算 …… 3—9
3.2.3 域关系演算 …… 3—12
3.3 关于数据库的数据操作语言 …… 3—14
3.3.1 基于关系代数的语言 *ISBL* …… 3—15
3.3.2 介于关系代数与演算之间的语言 *SEQUEL* …… 3—19
3.3.3 基于元组演算的语言 *QUEL* …… 3—27
3.3.4 基于域演算的语言 *QBE* …… 3—32
3.4 关系数据库的模式和子模式 …… 3—37
3.4.1 源模式、目标模式及其物理映射 …… 3—37
3.4.2 子模式、目标子模式及其映射 …… 3—41

3.5 询问的优化 …… 3－45
3.5.1 优化的一般策略 …… 3－46
3.5.2 关系代数表达式的等价代换规则 …… 3－47
3.5.3 关系代数表达式的优化算法 …… 3－48

第四章 层次方法 …… 4－1

4.1 一般概念 …… 4－1
4.1.1 树 …… 4－1
4.1.2 层次系统的数据模型 …… 4－3
4.1.3 层次顺序与层次路径 …… 4－5
4.1.4 层次系统的模式与子模式 …… 4－7
4.2 *IMS* 系统的逻辑结构 …… 4－8
4.2.1 *IMS* 的逻辑结构 …… 4－8
4.2.2 *IMS* 的 *DBD* …… 4－9
4.2.3 *IMS* 的 *PSB* …… 4－12
4.3 *IMS* 的存储结构 …… 4－14
4.3.1 *HSAM* …… 4－14
4.3.2 *HISAM* …… 4－15
4.3.3 *HIDAM* 和 *HDAM* …… 4－19
4.4 *IMS* 的数据子语言 …… 4－25
4.4.1 子语言 *DL/1* …… 4－25
4.4.2 *IMS* 的应用程序 …… 4－30
4.4.3 应用程序的运行 …… 4－35
4.5 *IMS* 存储结构补充 …… 4－36
4.5.1 辅数据集组 …… 4－36
4.5.2 *IMS* 辅助索引 …… 4－38
4.5.3 *IMS* 的逻辑数据库 …… 4－40

第五章 *DBTG* 建议的网状模型的数据库系统 …… 5－1

5.1 *DBTG* 系统的结构 …… 5－1

5.2 DBTG的数据模型 …… 5—2
5.2.1 记录类型 …… 5—2
5.2.2 络类型(Set type) …… 5—3
5.2.3 络事件(Set occurrence) …… 5—5
5.2.4 事物联系的DBTG表示法 …… 5—7
5.3 记录类型描述及其存储映射 …… 5—10
5.3.1 DBTG句法使用的符号 …… 5—10
5.3.2 记录类型的描述 …… 5—11
5.3.3 记录类型的存贮映射 …… 5—13
5.3.4 记录类型举例 …… 5—16
5.4 络类型描述及其存贮映射 …… 5—17
5.4.1 络类型(Set mode) …… 5—17
5.4.2 络次序(Set order) …… 5—18
5.4.3 从记录类型性质的描述 …… 5—23
5.4.4 络选择(Set selection) …… 5—24
5.4.5 络类型举例 …… 5—25
5.5 模式数据描述 …… 5—29
5.6 子模式数据描述 …… 5—29
5.6.1 子模式与模式的区别 …… 5—30
6.6.2 子模式举例 …… 5—30
5.7 数据操作语言(DML) …… 5—32
5.7.1 程序的运行 …… 5—32
5.7.2 DML语句概述 …… 5—33
5.7.3 应用程序举例 …… 5—38

第六章 关系数据库的规范化理论 …… 6—1

6.1 关系模式的规范化概述 …… 6—1
6.2 关系数据库的设计理论 …… 6—5
6.2.1 函数依赖(Functional Dependency) …6—6
6.2.2 计算闭包 …… 6—10

6.2.3 依赖集的复盖 …… 6 — 12
6.3 关系模式的分解 …… 6 — 13
6.3.1 分解的定义 …… 6 — 14
6.3.2 联接的不丢失性（*Lossless join*） …… 6 — 14
6.3.3 联接不丢失性的检验 …… 6 — 14
6.3.4 分解对依赖的保持 …… 6 — 16
6.4 关系模式的规范化 …… 6 — 17
6.5 结果为*BCNF*的联接不丢失性分解 …… 6 — 19
6.6 多值依赖及第四范式 …… 6 — 22

第六章　关系数据库的规范化理论

6·1 关系模式的规范化概述

我们以前提到的关系模式都是建立在已分解好的基础上。现在的问题是如何划分关系数据模型的关系是比较好的，不是随便地把一个大的关系分解成若干小的关系就行的。我们应明确关系模型的优劣，来改进较差的模型。

所谓规范化是指用更单纯、结构更规则的关系代替原有关系的过程。规范化一般有两种办法做，一种是将关系告诉系统，由系统自动分解。另一种是用户将关系规范化后交给系统。

（1） 规范化的目的

规范化是一个可逆过程，它将一组给定的关系转换为另一组关系，这一过程的可逆性保证了能够恢复到原先的那组关系，即变换的过程没有丢失任何信息。规范化的目的可以概括为以下六点：

1. 为使用户使用方便，关系中的每一个数据项应是一个简单的数或符号串，而不应再是一组数或一个重复组；

2. 为了使得对关系的检索操作更为简化；

3. 为了消除对关系中的数据进行插入、修改和删除时的相互牵扯；

4. 为了当对数据库引入新型数据时，减少对原有关系结构的修改。这样就可以使得应用程序也不需要常常改变；

5. 为了对用户来说，关系模式更灵活，更易于使用非过程化的高级查询语言进行查询；

6. 为了更易于进行各种查询统计工作。

（2） 向第一范式的变换

在关系中的每个值，也就是在元组中的每个属性值是原子的（不可再分）则此关系称为规范化的关系。也称第一范式。

例如图 6—1说明这种范式的实例。关系 *BEFORE* 定义在S^*和PQ上；而PQ的元素（因PQ本身又是一个关系）定义在域P^*和QTY上，因此 *BEFORE* 是来规范的关系。而 *AFTER* 是与 *BEFORE* 的一个等价规范关系。

BEFORE

	PQ	
	P#	QTY
S_1	P_1	300
	P_2	200
	P_3	400
	P_4	200
	P_5	100
	P_6	100
S_2	P_1	300
	P_2	400
S_3	P_3	200
S_4	P_2	200
	P_4	300
	P_5	400

AFTER

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_3	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

满足第一范式

图 6—1 规范化的范例

但 *AFTER* 仍有若干不合要求的结构和性质，为了说明这点，让我们考虑供应者和库存信息，不是分成两个独立的关系（*S* 和 *SP*），而是合成一个关系 *FIRST*（S^*，*STATUS*，*CITY*，P^*，*QTY*），而这些属性含义如常。然而，为了举例起见，我们引入一个额外的限制，即 *STATUS* 决定于 *CITY*（整个伦敦的供应者必须具有状态字 20）为了简便起见我们略去属性 *SNAME*。*FIRST* 的关键字是（S^*，P^*）的组合，如图 6—2所示。

FIRST

S#	STATUS	CITY	P#	QTY
S_1	20	London	P_1	300
S_1	20	London	P_2	200
S_1	20	London	P_3	400
S_1	20	London	P_4	200
S_1	20	London	P_5	100
S_1	20	London	P_6	100
S_2	10	paris	P_1	300
S_2	10	paris	P_2	400
S_3	10	paris	P_3	200
S_4	20	London	P_2	200
S_4	20	London	P_4	300
S_4	20	London	P_5	400

图 6—2 *FIRST* 范例表

从图中看出（a）$STATUS$和$CITY$并不完全决定于（S^*，P^*），（b）$STATUS$和$CITY$也并不互相独立。

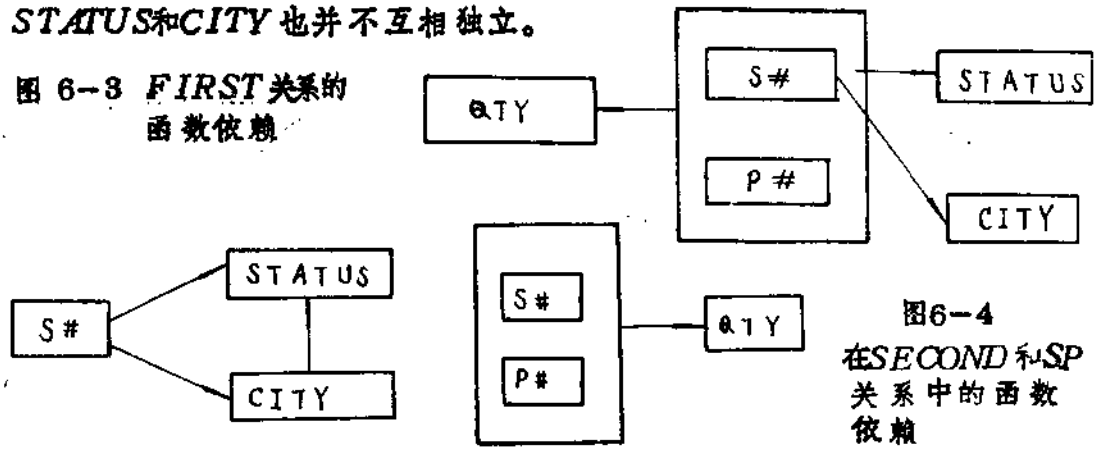

图 6-3 $FIRST$关系的函数依赖

图6-4 在$SECOND$和SP关系中的函数依赖

$FIRST$关系在存贮操作方面遇到了困难。为了明确起见，我们集中在供应者和城市之间的联系上，也就是，$CITY$函数依赖于$S\#$。对于三种基本存贮，每一种都会产生问题。

插入。直到那个供应者至少供应一个零件之前，我们就不可能记入该供应者到一个特定城市。其理由是：$FIRST$关系Key是由供应者号码和零件号码组成。故S_5直到供应某些零件之前，我们没有适当的Key值。

删除。如果我们仅仅删除$FIRST$中的特定供应者元组，不仅仅破坏了某些零件的库存量，而且还破坏了供应者位于特定城市的信息。例如，若我们要删除具有$S\#=S_3$，同时$P\#=P_3$ $FIRST$的元组，这样一来就损失了S_3在巴黎的信息。

修改。一般说来，对于一个给定供应者的城市在$FIRST$关系中出现多次。这种更新便产生了冗余的问题。例如，若S_1从$London$改到$Amsterdam$，这可能产生不一致的结果。因为S_1的城市有的地方是$London$，而另外的地方是$Amsterdam$。

（3） 向第二范式的变换

解决上述第一范式存贮操作方面的问题方法是：用两个关系$SECOND$（$S\#$，$STATUS$，$CITY$）和SP（$S\#$，$P\#$，QTY）代替一个关系$FIRST$。如图6-4说明了这两个关系依赖图；而图 6-5指出相应图 6-2的范例表。只是供应者S_5并入$SECOND$关系中（没在SP中）。

很明显这个修改结构克服了包括在$S\#-CITY$存贮操作中的全部缺点。

插入。我们将S_5位于$Athens$的信息记入，尽管S_5目前没供应任何

零件，此处仅仅插入到*SECOND*的适当元组中去。

删除。我们可以通过删除*SP*中适当元组，我们就可以删除连接S_3和P_3的库存量。

修改。在修改结构中，对于每个供应者所在的城市只出现一次。消除了非一致性。

从图 6—4和6—3可以看出，修改结构的作用是消除不完全决定于主关键字的结构。说明一个关系是第一范式而不是第二范式，可以通过适当的投影方法得到新的关系来代替原始的关系。原始关系又可以用这些投影关系的适当连接来恢复。

SECOND

S#	STATUS	CITY
S_1	20	London
S_2	10	Paris
S_3	10	Paris
S_4	20	London
S_5	30	Atnens

SP

S#	P#	QTY
S_1	P_1	300
S_1	P_2	200
S_1	P_3	400
S_1	P_4	200
S_1	P_5	100
S_1	P_6	100
S_2	P_1	300
S_2	P_2	400
S_3	P_3	200
S_4	P_2	200
S_4	P_4	300
S_4	P_5	400

图 6—5 满足第二式的*SECOND*和*SP*的范例表

然而，*SECOND*、*SP*结构仍然存在一些问题。关系*SP*令人满意的，现在我们不管它。可是关系*SECOND*的非*Key*属性中缺乏相互独立性。*SECOND*的依赖图仍然是比较复杂的。具体的说，*STATUS*依赖于*S#*，尽管这种依赖关系是函数的依赖，但却又是传递依赖（通过*CITY*）：每个*S#*确定了一个*CITY*值，而*CITY*转而又确定了*STATUS*的值。这种传递性再次导致存贮操作的困难。现在集中在城市和状态值之间联系上，也就是集中在*STATUS*和*CITY*的函数依赖上。

插入。直到某一供应者位于一个城市之前，我们不可能把该城市的特定状态值记入。例如，我们不能说，在*Rome*的任何供应者必须有状态字50。

原因还是直到这样一个供应者存在之前，我们没有适当的关键字值。

删除。若我们删除仅仅是*SECOND*中特定城市的元组，不仅破坏了所涉及供应者的信息，同时又破坏了该城市的特定状态值。例如，若我们要删除*SECOND*中S_5的元组，我们损失了*Athens*的状态字是30。

修改。对于一个已知城市的状态字在*SECOND*中出现多次（关系仍然包含某些冗余性）。在修改时可能引起不一致性。

（4） 向第三范式的变换

解决上述问题办法仍然采用投影的方法，原始关系*SECOND*用两个投影，这两个投影是*SC*(*S#*,*CITY*)和*CS*(*CITY*、*STATUS*)。图 6-6 是相应依赖图。图 6-7是相对原始关系*SECOND*的数据值。由于过程可逆，*SECOND*是*SC*和*CS*在*CITY*上的连接。

图6-6 在*SC*和*CS*中的函数依赖

很明显，新结构克服了关于*CITY*和*STATUS*联系在存贮操作方面存在的问题。比较图 6-4和6-6可知，进一步重构的效果是消除了*STATUS*传递依赖于*S#*。具体见图 6-7。

SC

S#	CITY
S_1	London
S_2	Paris
S_3	Paris
S_4	London
S_5	Athens

CS

CITY	STATUS
Athens	30
London	20
Paris	10

图 6-7 满足第三范式的*SC*和*CS*范例表

6·2 关系数据库的设计理论

我们采取将关系进行分解的规范化方法，来解决可能存在的插入、删除和修改时潜在的问题。但也带来另外的问题，例如在第三范式的*SP*、*SC*、*CS*三个关系上找出供应P_1的供应者的供应者号，住址和状态，就要多次联接运算才能达到目的，这要花费系统的很大开销。而从原始关系进

行限制筛选和投影，可解决问题且系统开销要小。在设计关系数据库时，需考虑在什么情况下进行分解，以及如何进行分解才是最有利。下面讨论关系模型的一些性质及一些算法，使之获得性能较好的关系模式集合。

6·2·1 函数依赖(*Functional Dependency*)

我们用一些关系来描述客观世界，也可以表示客观对象之间的联系。而实体内部的属性之间也存在着某种联系，即属性之间的函数依赖。函数依赖问题是关系模型的分解与关系数据的规范化理论的基础，我们应给予足够的重视。

1. 函数依赖的定义

定义1：

设有一个关系模式$R(A_1, A_2, \cdots, A_n)$，X和Y均为$\{A_1, A_2, \cdots, A_n\}$的子集。对于$R$的值$r$来说，当其中任意两个元组$u$，$v$中的对应于$X$的那些属性分量的值均相等时，则有$u$，$v$中对应于$Y$的那些属性分量的值也相等，我们就称$X$函数决定$Y$，或$Y$函数依赖于$X$，记为$X \rightarrow Y$。

我们可以举一些例子来说明这种函数依赖。例如，若X是组成R的关键字的那些属性的集合，则对这一关系的所有属性的子集Y，都有$X \rightarrow Y$。这是因为关键字唯一地确定一个元组。当两个元组的关键字相等时，这两个元组必定相等，即它们所有属性的值都应该相等。

关于函数依赖，我们强调三点：

1）当我们确定一个关系R中的函数依赖时，R的所有的值都必须满足函数依赖；反过来，R中只要有一个值不满足函数依赖，我们就可以认为R中不存在这个函数依赖；

2）当确定一个关系中的函数依赖时，我们只能从属性的含义上去加以说明，是对某些具体关系依赖是成立的。依赖是实际世界的真实说明，这是不能在数学上进行证明的；

3）只有数据库的设计者，才能决定是否存在函数依赖。这使得数据库管理系统可以根据设计者的意图来维护数据库的完整性。例如，一旦规定"姓名"函数决定"电话号码"，即

$$NAME \rightarrow PHONE$$

则就是排除了在这个数据库中一个人可以有两部电话的可能性。

2. 依赖的逻辑蕴涵

例如：假定R是一个关系模式，A、B、C为其属性，同时假定在R中

有函数依赖：$A\to B$和$B\to C$。则我们说R中必有函数依赖$A\to C$。

我们可以用反证法来证明这一结果。假定r是满足$A\to B$和$B\to C$的一组关系值，但不满足$A\to C$。其中两个元组u和v的相应于属性A的分量值相等，而它们相应于属性C的分量值不相等。我们要问：u和v的相应于属性B的分量值是否相等？如果不相等，则违反条件$A\to B$；如果相等，则违反$B\to C$。可见，r必定满足$A\to C$。

在这个例中，我们说函数依赖$A\to B$和$B\to C$逻辑蕴涵了函数依赖$A\to C$，或称从函数值赖$A\to B$和$B\to C$可以推出函数依赖$A\to C$。

一般地，逻辑蕴涵的定义如下：

定义2：设F为对应于关系模式R的一个函数依赖的集合。如果$X\to Y$为R中的一个函数依赖，而相应于R的每一个满足F中的函数依赖的关系r都满足$X\to Y$，则称F逻辑蕴涵$X\to Y$，或称$X\to Y$可以从F推出。

所有被F逻辑蕴涵的那些函数依赖的集合称为F的闭包（$Closure$），记为F^+。一般情况下，$F\subseteq F^+$。当$F=F^+$时，称F为函数依赖的一个满族（$Full\ Family$）。F^+有时也被称为F的传递闭包（$transitive\ Closure$）。

例6.1若$R=ABC$而$F=\{A\to B, B\to C\}$，则F^+由所有下列这些逻辑蕴涵组成

$$
F^+=\left|\begin{array}{lllll}
A\to\phi, & AB\to\phi, & AC\to\phi, & ABC\to\phi, & B\to\phi, \quad C\to\phi \\
A\to A, & AB\to A, & AC\to A, & ABC\to A, & B\to B, \quad C\to C \\
A\to B, & AB\to B, & AC\to B, & ABC\to B, & B\to C, \\
A\to C, & AB\to C, & AC\to C, & ABC\to C, & B\to BC, \\
A\to AB, & AB\to AB, & AC\to AB, & ABC\to AB, & BC\to\phi, \\
A\to AC, & AB\to AC, & AC\to AC, & ABC\to AC, & BC\to B, \\
A\to BC, & AB\to BC, & AC\to BC, & ABC\to BC, & BC\to C, \\
A\to ABC, & AB\to ABC, & AC\to ABC, & ABC\to ABC, & BC\to BC,
\end{array}\right|
$$

由此可见，一个小的F集合，常有一个大的闭包F^+，计算F^+通常很困难。在F^+中，含有许多平庸的（$trivial$）函数值赖，例如：$A\to\phi$，$AB\to A$，$ABC\to ABC$。所谓$X\to Y$是平庸的，是指$Y\subseteq X$。

3. 关键字

下面，我们利用函数依赖，给关键字下一个定义：

定义3：设$R(A_1, A_2, \cdots, A_n)$为一个关系模式，F为其函数依赖的一个

集合，X为$\{A_1, A_2, \cdots, A_n\}$的一个子集。如果

1）$X \to \{A_1, A_2, \cdots, A_n\} \in F^+$，并且

2）不存在$Y \subset X$，使得$Y \to \{A_1, A_2, \cdots, A_n\} \in F^+$，

则称X为R的一个候选关键字（*Candidate Key*）。

这里，条件1）是说X函数决定R的所有属性；条件2）是说X是满足条件1）的最小的集合。即，唯一地确定一个实体的最小属性集合，称为关键字。

对任一关系来说，可能存在不止一个候选关键字，通常选择其中一个作为主关键字（*Primary Key*），这种选择可以是任意的，即任一候选关键字都可以被选作主关键字。

例 6.2 R和F按例6.1所示，则A为唯一关键字，因$A \to ABC$在F^+中，A决定所有属性且是最小的属性集合（单属性）。

4. 函数依赖公理

为了确定候选关键字，我们需要从F计算它的闭包F^+，或者至少对于一个给定的F和$X \to Y$，确定$X \to Y$是否属于F^+。这里提供了一套完整的推理规则，它能从F推出F^+中所有的依赖。而且，这套规则的正确性还体现在：它从F决推不出任何不属于F^+的依赖。

这套规则称为阿姆斯特朗公理（*Armstrong's axioms*）

假定一个已知的关系模式R，其全部属性的集合为U，F是只涉及U中属性的一个函数依赖集合。推理规则如下：

A1:（反射性，*reflexivity*）

如果$Y \subseteq X \subseteq U$，则$X \to Y$一定被$F$逻辑蕴涵。此规则给出了平庸依赖，即$X \to Y$的右边含在左边中。

可见这个规则的使用与F无关。

A2:（外延性，*augmentation*）

如果$X \to Y$成立，而$Z \subseteq U$，则$XZ \to YZ$成立。

注意：为了简单起见，此后我们用XZ代表X与Z之并，即$X \cup Z$。可从给定依赖集利用此公理推导出蕴涵依赖。

A3:（传递性，*transitivity*）

如果$X \to Y$和$Y \to Z$成立，则$X \to Z$成立。

例 6.3 设有关系模式$R(CITY, ST, ZIP)$，其中ST表示城市$CITY$

中的某个街道，而ZIP表示这一地址的邮政编码。

假定其中有函数依赖如下：

$CITY, ST \rightarrow ZIP$

$ZIP \rightarrow CITY$

我们运用上面的公理：

1） $ZIP \rightarrow CITY$（给定条件）

2） $ST, ZIP \rightarrow CITY, ST$（外延性）

3） $CITY, ST \rightarrow ZIP$（给定条件）

4） $CITY, ST \rightarrow CITY, ST, ZIP$（外延性）

5） $ST, ZIP \rightarrow CITY, ST, ZIP$（传递性）

可知ST, ZIP是候选关键字。

引理1：阿姆斯特朗公理是正确的（$Sound$）。即如果$X \rightarrow Y$是根据公理由F推出的，则对于任何使F为真的关系，都有$X \rightarrow Y$为真。

证明：$A1$是明显正确的。因为对于任一关系，其两个元组在X的分量上相等，则在任一X的子集的分量上也自然相等。

为了证明$A2$(外延性)，我们假定一个关系r满足$X \rightarrow Y$，同时它有两个元组u和v在$X \cup Z$的分量上相等，而在$Y \cup Z$的分量上不相等。

由于它们在Z的任何分量上必然是相等的，因此必定是在Y的分量上不相等。但这违反假定$X \rightarrow Y$，可见必有$XZ \rightarrow YZ$。

至于$A3$（传递性）的正确性，只是6-6页下面例的扩展，这里就不证了。

从阿氏公理出发，我们还可以得出几个推理规则。

引理2：a）并规则（$The\ Union\ Rule$）

若$X \rightarrow Y$与$X \rightarrow Z$成立，则$X \rightarrow YZ$成立；

证明：$X \rightarrow Y \Rightarrow XX \rightarrow XY$ （$A2$）

$X \rightarrow Z \Rightarrow XY \rightarrow YZ$ （$A2$）

∴有 $X \rightarrow YZ$ （$A3$）

b）伪传递规则（$The\ Pseudotransitivity\ Rule$）

若$X \rightarrow Y$与$WY \rightarrow Z$成立，则$XW \rightarrow Z$成立；

证明：$X \rightarrow Y \Rightarrow WX \rightarrow WY$ （$A2$）

$WY \rightarrow Z$（给定条件）

∴有 $XW \rightarrow Z$ （$A3$）

c) 分解规则(*The Decomposition Rule*)

若$X \to Y$成立，且$Z \subseteq Y$，则$X \to Z$成立。

证明：$Z \subseteq Y \Rightarrow Y \to Z$　　(A1)

∴ 有　　$X \to Z$　　(A3)

由并规则和分解规则可以得出一个重要的结论，即：如果A_1，A_2，…，A_n是属性，则$X \to \{A_1, A_2, \cdots, A_n\}$成立的充分必要条件是对于每个$i$，$X \to A_i$均成立。

定理：*Armstrong*公理是正确的(*Sound*)，同时也是完全的(*Complete*)。给定依赖集F能推出所有的依赖F^+，称此规则是完全的。而依给定依赖集F由此规则推导不出不属于F^+的依赖，称规则是正确的。证明从略。

6.2.2 计算闭包

(1) 以前我们讲过一个函数依赖的集合F的闭包F^+的概念。现在我们来定义一下一个属性集合的闭包的定义，此属性集合有关于一个函数依赖的集合。即要知道依赖$X \to Y$是否成立，可根据*Armsteng*公理由F推导迅速得知。

定义4：设F是对属性集合U的一个函数依赖集合，X是U的一个子集，则X关于F的闭包X^+是属性A的集合，其中$X \to A$可以根据阿氏公理由F推出。即

$$X^+ = \{A \mid A \in U, X \to A \in F^+\}$$

引理3：$X \to Y$能由阿氏公理推出的充分必要条件是$Y \subseteq X^+$。

证明：

设$Y = A_1 A_2 \cdots A_n$(A_1，A_2，…，A_n为属性)并假定$Y \subseteq X^+$。

根据X^+的定义，对于所有的i，$X \to A_i$可由阿氏公理推出。根据并规则，有$X \to Y$。

反之，假定$X \to Y$能由阿氏公理推出。根据分解规则，对于所有的i，$X \to A_i$成立。

所以有$Y \subseteq X^+$。证完。

(2) 闭包的计算

显然，对于依赖的一个集合F计算闭包F^+是一件相当麻烦的事。这是因为有时尽管F很小，而F^+却可能很大。

考虑集合$F=\{A\to B_1, A\to B_2, \cdots, A\to B_n\}$，则$F^+$包括所有的依赖$A\to Y$，其中$Y$是$\{B_1, B_2, \cdots, B_n\}$的一个子集。由于一共有$2^n$个这样的$Y$，因此就是对于并不太大的$n$，也很难列出全部$F^+$来。

而另一方面，对于一个属性的集合X，计算其闭包X^+却并不太难。这个工作量正比于F中所有依赖的总数。而由引理3要判断$X\to Y$是否在F^+中（是否是F^+的一个成员），只要计算X^+即可，一个计算X^+的简单方法如下：

算法1：

输入：一个有穷的属性集合U，一个U上的函数依赖集合F，以及一个属性集合$X\subseteq U$；

输出：X关于F的闭包X^+；

方法：根据下面的规则计算一系列属性集合$X^{(0)}, X^{(1)}, \cdots$

1) $X^{(0)}$就是X；

2) $X^{(i+1)}$就是$X^{(i)}$加上属性集合A，使得F中有一个依赖$Y\to Z$，而$A\subseteq Z$，同时有

$$Y\subseteq X^{(i)}$$

由于$X=X^{(0)}\subseteq\cdots\subseteq X^{(i)}\subseteq\cdots\subseteq U$，而$U$是有穷的，我们最终一定达到一个$i$，使得$X^{(i)}=X^{(i+1)}$，而从此以后有$X^{(i)}=X^{(i+1)}=X^{(i+2)}=\cdots$，因此，一旦我们发现$X^{(i)}=X^{(i+1)}$，就不需要再计算下去了。可以证明$X^+$就是对于这个$i$值的$X^{(i)}$。

例6·4：设F由下面8个依赖所组成：

$AB\to C$	$D\to EG$
$C\to A$	$BE\to C$
$BG\to D$	$CG\to BD$
$ACD\to B$	$CE\to AG$

并设$X=BD$。

为了使用算法1，令$X^{(0)}=BD$。

先计算$X^{(1)}$。找一个依赖，其左边是B，D或BD。这样的依赖只有一个：$D\to EG$。因此将E和G加到$X^{(0)}$去，则$X^{(1)}=BDEG$。

再计算$X^{(2)}$。找左边包含于$X^{(1)}$的依赖，有两个：$D\to EG$和$BE\to C$。则得$X^{(2)}=BCDEG$。

再计算$X^{(3)}$。找左边包含于$BCDEG$的依赖，除原先的两个外，还有C

$\rightarrow A$，$BG\rightarrow D$，$CG\rightarrow BD$和$CE\rightarrow AG$。则得$X^{(3)}=ABCDEG$。这是所有属性组成的集合。显然$X^{(3)}=X^{(4)}=\cdots$，因此得：

$$(BD)^{+}=ABCDEG$$

6·2·3 依赖集的复盖

上面讲过根据函数依赖集可找到一属性集X的闭包，然后根据引理3可判断依赖$X\rightarrow Y$是否可由依赖集推导出来，从而可利用此办法判断X是否为关键字。下面看一下，如何利用此办法找到一个依赖集的最小复盖问题。

定义5：依赖集的复盖

设F和G为依赖集；如果$F^{+}=G^{+}$，则称F和G是等价的。

如果F和G是等价的，则称F覆盖G（同时有G覆盖F）。检查F和G是否等价是很容易的：只要对F中的每一个依赖$Y\rightarrow Z$，检查它是否属于G，同时使用算法1计算Y^{+}，然后检查是否满足$Z\subseteq Y^{+}$。如果F中的某个依赖$Y\rightarrow Z$不属于G^{+}，则必然$F^{+}\neq G^{+}$。如果F中的每一个依赖都属于G^{+}，则F^{+}中的每一个依赖也属于G^{+}。我们可以得出结论：F和G等价的充分必要条件是：F中的每一个依赖都属于G^{+}，即$F\subseteq G^{+}$；同时G中的每一个依赖都属于F^{+}，即$G\subseteq F^{+}$。

下面的引理是很有用的：

引理4：设G是一个依赖集，且其中所有依赖的右部都只有一个属性，则G覆盖任一左部与G相同的函数依赖集F。证明略。

下面我们定义最小依赖集。

定义6：我们称一个依赖集F是最小的，如果：

1）F中每一个依赖的右部都是一个单属性；

2）对于F中的任一依赖$X\rightarrow A$来说，$F-\{X\rightarrow A\}$都不与F等价；

3）对于F中的任一依赖$X\rightarrow A$来说，$(F-\{X\rightarrow A\})\cup(Z\rightarrow A)$都不与$F$等价，其中$Z$为$X$的任一子集。

在这个定义中，条件1）保证了每个依赖的右部都不会有重复的属性；条件2）保证了F中没有重复的依赖；条件3）保证了每个依赖的左边都不会有重复的属性。

定理：所有的依赖集F都与相应的某个最小的依赖集F'等价。（F'是F的最小复盖）证明略。