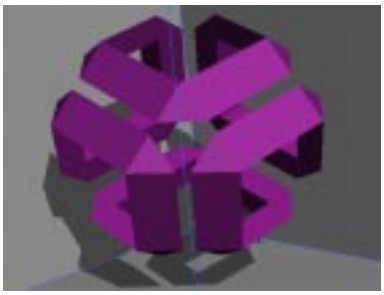
BaBilon



zzuzzu
Karma
Perfect

Implmentation of Normalization

Documetation for Program

이전영 교수님

Babilon

전자계산학과
9425006 김길연
9425033 허윤구
9425034 현대은

1996년 6월 24일

목 차

제 1 절 Introduction

1 Normalization의 필요성

Normalization은 Relational DataBase design에서 중요한 역할을 한다. 일반적인 DataBase design의 목적은 아래와 같다.

- 불필요한 redundancy가 없는 information을 저장한다.
- information을 쉽게 retrieve 할 수 있다.

이를 만족하는 Relational schemes의 집합을 만들어 내는 것이 궁극적인 목적인 것이다. 위의 조건을 만족시키기 위한 한 가지 방법은 적당한 normal form으로 된 scheme들을 design하는 것이다. Relational scheme들이 normal form의 하나인지 아닌지 알기 위해서 정보가 필요하게 되는데 그 정보는 우리가 modeling하고 있는 real-world에 대한 것으로, data dependency라 불리는 constraint의 모임이다.
이와 같이 Normalization 의 과정은 Datba Schema를 Design하는 과정에서 빼놓을 수 없는 중요한 과정이다. 즉, Conceptual Schema과정에서 Nomalized된 Table은 전체 Database의 효율성과 관련이 커서 Realtional Database에서 중요한 과정을 차지한다.
따라서 대부분의 Databse Design Tool에서 이 Normalization에 관해서 자동화된 장치들을 제공하고 있으며, Normalization에 대한 개념의 확실한 이해와 이런 자동화장치를 simulation하기 위해 이 프로그램을 짜게 되었다.

2 Program의 목표

이 프로그램의 개발과정에서 목표로 했던 것과 rule들은 다음과 같다.

- 가장 일반적이고 전체적인 Database system에 적합한 Model을 적용한다.
- 모든 경우를 cover할 수 있는 이론에 기반을 둔 장치를 개발한다.
- C++와 RCS를 사용, 분담작업시 일어날 수 있는 문제점을 해결하고 효율을 높인다.
- Library와 Object component에 의존, program의 확장성과 재사용성을 높인다.

3 Program 개발의 일정과 작업계획

이에 따라 전체적인 program을 진행시켜 나갔으며, 그 일정과 계획은 다음과 같았다.

- 첫째 주
  전체적인 구조와 필요한 Library와 도구들을 결정한다,
- 둘째 주
  Algorithm을 결정하고, DataStructure를 잡는다.
- 셋째 주
  주어진 구조를 직접 implementation하고 상속관계와 내부 function을 결정한다.
- 넷째 주
  Member fuction을 define하고, 필요한 함수들을 coding한다.
- 다섯째 주
  MOTIF를 사용, Interface를 제작하고 짜여진 program에 Binding한다.
- 여섯째 주
  TESTING(BETA) DATA를 시용하여 Debugging을 하고, 수정 보완한다.

4 작업의 분담과 역할

우선 첫째주와 둘째주에 행하게 되는 전체적인 Normalization의 process와 Program의 구조를 잡는 일은 3명이 토론을 통하여, 결정을 하였다. 전체적인 구조와 Datastructure를 잡은 다음. 필요한 Class를 define하는 과정에서는 상호작업 후 서로간의 차이를 조정하는 형식을 취하였다. 기본적인 DataStructure를 잡은 후 필요한 Member function들과 Normaliza function에 대해서는 아래와 같이 분담을 하였다.

- **Attribut와 AttSet Class의 Implementation** : 허 윤구
- **FD와 FDSet Class의 Imeplmentation** : 현 대은
- **RelationSchema Class의 Implementation** : 김 길연
- **2NF 형성 Function** : 허 윤구
- **3NF 형성 Function** : 현 대은
- **BCNF 형성 Function** : 김 길연

여기서 유의할 사실은 2NF, 3NF, BCNF Function이 RelationSchema Class에서 define되고, 이런 Function들을 만들기 위해서는 FDSet과 AttSet의 level에서 처리해 주는 Function이 존재하여야 한다는 것이다. 그래서, 각기 Class를 Define한 후(기본적인 Member만) Normal form을 형성하는 사람이 Class를 확장시켜나가는 방법을 사용하였다. 그리고, RelationSchema에서 처리해 주는

Normal form 형성 Member function은 복잡한 Algorothm의 Implementation을 요구하였기 때문에, 3명이 나누어 Implementaion하였다. 이 과정에서 RCS를 사용하여 공통작업을 수행하였다.

제 2 절 Normalization Process의 구조

Normalization을 행하기 위해서는 Relation자체의 Attribute들과 그에 따른 Functional Dependency 관계를 가지고 있는 Schema들이 존재하여야만 한다. 따라서, 이들 각각의 Attribute들과 FD(Functional Dependency, 이하 줄여 말하겠음)에 대하여 Set이 존재하여야 하며 이들이 모여 Relation과 Relation Schema를 이루고 이들이 모여 전체 DataBase Schema를 이룬다.

1 Attribute

Relation의 Table에서 각각의 column을 이루고 있는 속성들로 이 들간의 FD관계가 Normalization의 대상이다. Attribute 자체는 Relation과 직접적인 관계는 없고 Attribute들을 Define함으로써 Relation의 원소들을 결정할 수 있다.

2 Attribute Set

Relation에 속하게 되는 Attribute들을 모아둔 집합으로 한 Relation을 규정한다.

3 FD

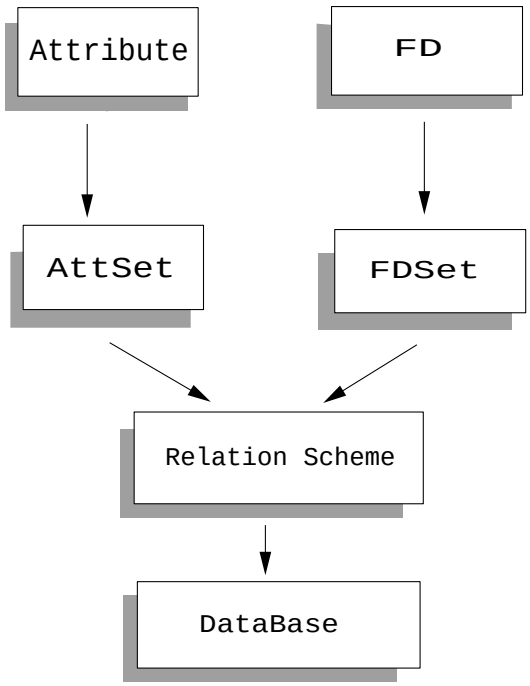
어떤 Attribute들 사이의 Functional Dependency를 규정한 것이다. LHS와 RHS로 이루어져 있는데 LHS $\rightarrow$ RHS는 RHS가 LHS에 Functionally Dependent하다는 이야기다. LHS, RHS는 Attribute의 Set 으로 이루어져 Multiple primary key와의 상관관계등을 표현할 수 있다.

4 FD Set

한 Relation을 구성하고 있는 Attribute들의 관계, 즉 FD들을 모아둔 모임이다. 즉, 한 Relation안에 존재하는 Attribute들로 규정된 FD들의 모임인데, Relation 자체에는 드러나 보이지 않지만 내부적으로 규정된 rule로 Normalization과정이 이런 FD Set을 보고 Relation을 decomposition하는 과정이라 보면 된다.

5 Relation Schema

어떤 Relation(Attribute의 definition)과 그들 간의 FD rule인 FD Set이 모여서 이루어진 것으로 한 Relation의 Detailed information이라 할 수 있다. 이들 Re-



lation Schema를 보고 혹은 이를 실제로 decomposition하여 새로운 Relation을 가지는 scheme들로 나누는 것이다.

6 DataBase

위에서 규정된 Relation Schema들이 모여서 이루어진 것으로, Redundancy가 존재하지 않는 것으로 이들 각각의 Relation Schema들이 Normalization의 대상으로, Normalization 과정에서는 그다지 중요성을 가지지 못하고 그저 전체를 새로 Define하고 Update하는 범위로서의 의미만 가진다.

위의 각각의 Component를 가지고 이들의 계층도를 그려 보겠다.

제 3 절 Normalization의 Algorithm

여기서 언급하게 될 Normalization의 Algorithm은 Atzeni의 Reference 1을 참조한 Algorithm을 기반으로 한 것이다. 2NF는 simple하기 때문에 생략하고 3NF와 BCNF에 대해서 설명하도록 하겠다.이 Algo rithm은 위의 Relation Schema를 가지고 개개의 Relation의 FDSet을 보고 Normalize를 시키는 과정으로 Analysis와 Synthesis로 나누어 진다.물론 이들 두가지 Algorithm은 주어진 FDSet과 Relation에 대해서 같은 결과를 가진 Relation Schema를 생성한다. 우리가 취한 Algorithm은 Synthesis로 보통 우리가 취해온 Normalization의 decomposition과는 다른 것이다. 일반적으로 1NF $\rightarrow$ 2NF $\rightarrow$ 3NF의 과정은 Analysis algorithm을 사용한 것으로 계속 나누어 지는 Relation Schema들에 decomposition을 가하는 것이다.

1 Analysis Decompositio

Analysis Algorithm은 원하는 Nornmal form에 모자라는 원인을 제거함으로써 Normalized된 scheme을 형성하는 방법이다. 어떤 dependency 관계 때문에 원하는 Normal form을 형성하지 못한다 하면 이 dependency의 관계에 의해 2,3개의 Relation Scheme오 나누어 진다. 이렇게 decompose된 각각의 Relation Scheme들은 이전 Scheme이 가지고 있었던 제약조건들을 모두 만족시킨다. 이런 작업은 Normal form이 만족될 때까지 계속 반복된다. 그래서 전체 Schem은 각각의 단계마다 보다 많은 Relation을 가진 새로운 Scheme로 바뀌는 것이다. 하지만, 이 Analysis Algorithm에는 두 가지의 큰 약점이 있다.

- 문제는 목적으로 하는 Normal form을 만족하는가 마는가를 확인하는 문제이다. 각각의 Normalize 과정은 이전 단계의 Normal form을 만족하는가를 필요로 한다. 이 과정은 특히 3NF의 과정에서 엄청난 co st를 만든다.
- 대부분의 Normal form이 각각의 Relation Scheme를 참조하고 그에 관계된 constraints 또한 참조 하기 때문에 decomposition step은 새로운 decomposition 과정에서 나온 새로운 relation 과정의 constra int를 모두 계산해야 하는 번거로움이 생긴다. 이 문제는 주어진 dependency에 관한 Closure를 모두 구해 야만 할때 더욱 많은 문제점을 일으킨다.

이 두가지 문제점 때문에 Synthesis Decomposition을 사용하게 된다.

2 Synthesis Decomposition

Synthesis 과정은 주어진 Dependency를 고려하여 standard cover로 Reduce하여 일정한 기준에 따라 구분해 나감으로써 시작된다. 그 다음 이런 구분된 단위들이 Relation Scheme와 관련이 되어 element에 들어 있는 모든 dependency를 담게 된다. partition들은 원하는 Normal form을 관련되는 Relation Sc heme들이 만족하도록 나누어 진다. 그래서, 중간 과정의 schem들이 생성되는 것이 필요 없이 final Scheme들이 생성되는 것이다.

2.1 3NF의 Algorithm

3NF는 우선 L-R Minimum cover를 구하여 dependency에 따라 구분을 할 수 있게 하여 놓고 3NF의 과정의 구분 기준이 되는 transitive depemcency에 따라 나누게 된다. 이런 dependency에 따라 3NF 형태를 만족하는 형태들이 Relation Scheme에 속해지는 것이다.
다음의 Pseudo-Algorithm에서 1,2 과정이 이런 Synthesis Algorithm의 중심부를 나타내고, 3번 과정은 lossless decomposition임을 보증하는 장치이다.

Input : Database Scheme R = [R(U), F]
Output : DataBase Scheme S
Begin
1. Find G, circular LR Minimum cover of F
2. For each equivalence class y of LHS's in G,
let S contain one relation scheme $[R_i\ (U_i,F_i)]$ with
U_i = U (x∈y,x→y ∈ G) XY and
F_i = V→A | V→W ∈ G, A∈W, VA ⊆ U_i
3. If there is no relation scheme, $R_i(U_i)$ ∈ S such that U_i → U ∈ G^+
Then add $[R_k$(K),] to S, where K → U ∈ G^+ **End**

2.2 BCNF의 Algorithm

BCNF는 Minimum cover를 구해서 determinant의 요건, 즉, Key를 찾음으로써 partition의 조건을 구해 나간다. 이것을 구한 다음, Relation Scheme를 만들어 나가는 것이다. 1,2,3의 과정이 이런 Synthe sis과정의 핵심이며 4의 부분은 BCNF를 유지하며 Relation Scheme를 reduce 해 나가는 과정이다.

Input : Attribute Set U와 FD Set F
Output : DataBase Scheme S
Begin
1. Find G, Minimum cover of F
2. Let S be the Database Scheme with one relation scheme
$R_i(X_i\ Y_i)$, $X_i \rightarrow Y_i$ for each $X_i \rightarrow Y_i$ ∈ G;
3. If there is no relation scheme $R_i(U_i)$ ∈ S such that U_i →U ∈ G^+
Then add $[R_k$(K),] to S, where K → U ∈ G_+
4. While S contains a pair of distinct relation schemes
$R^{'}(Z^{'})$, $F^{'}$
and $[R^{''}(Z^{''})$,$F^{''}]$ with $Z^{'} = Z^{''}$
Do drop the two schemes from S and add $[R^{'}(Z^{'}),\ F^{'}F^{''}]$
End

제 4 절 Program의 구성과 주요 DataStructure

위에서 언급한 바 있는 Database Schme를 Impelmentation하기 위해서는 Attribute들과 FD의 Data structure를 먼저 만드는 것이 중요하다. 그 다음 이 class들을 토대로 Attribute Set과 FD Set 을 이끌어 내야 하는데, 이 들 set을 Implementation하기 위해서는 여러가지 방법이 있으나 program의 목적에 맞게 General한 Architecture를

위해 가장 효율적이라고 알려진 Set Library를 사용하였다. 이 Library에서 define된 각종 Set operation을 Attribute Set과 FD Set에서 Virtual 상속해서 사용하였다. 이 Library는 General한 tool로 우리가 익히 아는 operation을 행함으로, 여기서 자세한 설명은 피하겠 다. 참고로 이 Library는 LEDA-R-3.3.1로써 공학 2동 Work Station room의 /home/2/94/zzuzzu/ftp/Cpp-lib/LEDA-R-3.3.1에 깔려 있으며, 그 외 public ftp site에서 받아올 수 있다.
여기서는 이런 Set에 기반을 둔 주요 Data Structure와 그 기능에 대해서 이야기하겠다.

1 Attribute Class

Attribute Class는 그 자신의 이름과 어디에 속하는 지를 알려주기만 하면 된다.

```
class   Attribute {
         string  name;
         string  domain;
```

이런 Data Member 외에 Attribute사이간의 비교와 연산을 위해 Constructor, destructor, operator등이 필요하다. 뒤에 나오는 모든 Class Object들이 다 그렇지만 operator loading을 통해 주어진 결과를 빨리 확인할 수 있도록 하였다. 중요하진 않지만 가장 기본적인 Class object이다.

2 FD Class

FD는 Functional Dependency를 결정하는 LHS와 RHS part의 Attribute Set이 존재하면 된다.

```
class   FD {
         AttSet  LHS;    // Left Hand Side Attribute Set.
         AttSet  RHS;    // Right Hand Side Attribute Set.
```

이 들 Data Member들을 가지고 FD의 Set내에서 operation을 해야하고 이들의 값을 할당 또는 출력하는 기본적인 일들이 필요하다.즉, 어느 FD Set에 속하였는가 또는 Redundant한가와 LHS와 RHS 값을 할당받고 출력 비교하는 일이 필요한 것이다.이들을 행하는 Member function은 다음과 같다.

```
        //--------
        bool    isMemberOf (const FDSet& fd_set);       // 말그대로
. F |= X->Y?
        bool    isRedundantIn (const FDSet& fd_set);    // 말그대로
.
        //--------
        AttSet  GetAttSet() const;
```

```
AttSet  GetLHS() const { return LHS; }
AttSet  GetRHS() const { return RHS; }
//--------
void    SetLHS(const AttSet& lhs) { LHS.clear(); LHS = lhs; }
void    SetRHS(const AttSet& rhs) { RHS.clear(); RHS = rhs; }
void    SetLHS(const Attribute& start, ...);
void    SetRHS(const Attribute& start, ...);
//--------
void    AddLHS(const AttSet& lhs) { LHS += lhs; }
void    AddRHS(const AttSet& rhs) { RHS += rhs; }
void    AddLHS(const Attribute& start, ...);
void    AddRHS(const Attribute& start, ...);
//--------
FDSet   one_to_one();
```

마지막의 function은 3NF에서 쓰이는 Function으로 Algorithm 과정 2에 해당한다.

3 Attribute Set Class

AttSet Class는 Library에서 inherit한 Obejct로 Set에서 Attribute로 Generic한 것이다. 따라서, Set의 기본적인 operation을 Attribute element에 맞춰 redefine했고 virtual inheritance를 이용하였다. 즉, 첨가된 Data Member는 없이 LHS와 RHS를 이름으로써 필요한 Closure들(XPLUS = X^+)을 구하는 Member Function 들이 추가되었다.

```
class   AttSet : public set<Attribute> {
public:
//--------
 string          GetAttNameConcat();
         Attribute       choose (const char* _name);
         AttSet          ComputeClosure(const FDSet& fd_set);
          AttSet          GetSubSet(int i, int order);
```

4 FD Set Class

FDSet도 마찬가지로 Library에서 inherit한 Obejct로 Set에서 FD로 Generic한 것이다. 거의 대부분의 Normalize과정을 담고 있기 때문에 Standard Cover를 구하는 Member function을 이룬다. virtual로 Set operation이 redefine된 것은 마찬가지다.

```
class   FDSet : public set<FD> {
```

```
public:
 //----- Find Various Cover. "This" object does not change.
        FDSet   LHSNonRedundant();
        FDSet   NonRedundantCover ();
        FDSet   MinimumCover1 ();               // [Ref1]의 알고리
즘 이용.
        FDSet   MinimumCover2 ();               // [Ref2]의 알고리
즘 이용.
        FDSet   LeftReducedCover ();    // FDSet의 모든 X->Y에서 X를
최소로.
        bool    RightReduceTest (const Attribute&, const FD&, FDSet);
        FDSet   RightReducedCover ();   // FDSet의 모든 X->Y에서 Y를
최소로.
        FDSet   ReducedCover ();                // L+R 즉 LRReduce.
        FDSet   CanonicalCover() const; // FDSet의 모든 FD는 X->A의
형태.
        FDSet   LRMinimumCover ();              // LR-circular Minimum구
함.
```

5 RelationSchma Class

RelationSchema는 Attribute Set과 FD Set으로 이루어져 있으며 실제의 Normalization 과정이 일어나는 곳이다. 따라서, 중요한 두 부분의 Data Member를 담고 있으며, Normal form을 형성하는 다른 Member function들이 들어 있다.

```
class   RelationSchema {

         string          name;
         Relation        relation;
         FDSet           fd_set;
public:
//---------
        void            SetName (const char* _name) { name = (string) _name; }
        void            SetRelation (const Relation& src)       { relation = src; }
        void            SetFD (const FD& src);
        void            AddFD (const FD& src);
        void            SetFDSet (const FDSet& src);
        void            AddFDSet (const FDSet& src);
        //---------
        AttSet          FindKey ();
```

```
        DataBase        ComputeBCNF2();         // [Ref2]의 알고리
즘 이용.
        DataBase        ComputeBCNF3();         // [Ref3]의 알고리
즘 이용.
        DataBase        Compute3NF() ;
```

6 Interface의 구현

Interface는 Motif 1.2.2를 사용, Relation Schema를 ADD, DELETE하고 그에 따른 FDSet과 AttSet을 사용자가 Interactive하게 setting할 수 있게 하였다. 이렇게 설정된 Relation Schema를 Directed Graph로 보면서 Normalize를 시킬 수 있다. 이때, 적용된 결과 또한 Directed Graph로 볼 수 있으며 중요한 Standard Cover인 L-R Minimum Cover와 Minimum Cover 또한 볼 수 있다. 이 들 FD를 Directed Graph로 나타내기 위해서 FDViewer라는 Class Object 를 생성하였으며, xhdg-1.1이라는 DG generate Lib를 사용하였다. Lib는 /home/2/94/zzuzzu/ftp/Widget/xhdg-1.1에 있으므로 얼마든지 참조하기 바란다.

제 5 절 Project에 대한 Remark

이번 Project는 General한 Design및 System adaptability를 유지하기 위해 노력했기 때문에, Implementaion과정이 생각보다 까다로왔다. 일반적으로 모든 Cover를 구하기 위해서는 FD에 대한 이론적인 지식이 매우 강해야만 하는데, 책의 Algorithm자체만을 보고 이해하고 Implementation 하기는 매우 어려 웠다. 따라서, 먼저 각 Algorithm에 대한 기반 지식(단순한 FD개념의 이상)을 갖춰야 지만 Implementation이 잘 진행되리라 믿느다. 특히 Synthesis Algoprithm은 우리가 익히 보아온 Algorithm과는 거리가 먼것으로 중간에 Test를 하기가 쉽지 않고, 직관적인 coding보다 논리에 기반한 coding이기 때문에 이론에 충실히 몰두해야만 한다.

Group Porject의 편의를 위해 OO-programming과 RCS를 시용한 분담 작업을 추진했지만, 초보단 계인 관계로 익숙치 않아 더 불편해 지고 작업 속도가 느려지기도 했다. 따라서, Group Project인 경우 개인의 독자적인 진척보다는 서로간의 교류로 작업과정과 분담을 조정해 나가는 것이 Project의 추진에 core라고 생각된다.