碩 士 學 位 論 文
指導敎授 金 英 雄

NCPI-MDS:수정된 DTD 간소화 절차를 통한 새로운 Constraints-Preserving Inlining 기법

NCPI-MDS:New Constraints-Preserving Inlining method with Modified DTD Simplification

２００６年

漢城大學校 大學院
컴 퓨 터 工 學 科
컴 퓨 터 工 學 專 攻
安　　成　　哲

碩 士 學 位 論 文
指導敎授 金 英 雄

NCPI-MDS:수정된 DTD 간소화 절차를 통한 새로운 Constraints-Preserving Inlining 기법

NCPI-MDS:New Constraints-Preserving Inlining method with Modified DTD Simplification

위 論文을 컴퓨터工學 碩士學位論文으로 提出함

2 0 0 5 年 12 月

漢城大學校 大學院
컴 퓨 터 工 學 科
컴 퓨 터 工 學 專 攻
安 成 哲

安 成 哲의 컴퓨터工學 碩士學位 論文을 認定함

２ ０ ０ ５ 年 12 月

심사 위원장_____(인)

심 사 위 원_____(인)

심 사 위 원_____(인)

국 문 초 록

XML(eXtensible Markup Language)은 웹 상의 데이터를 표현하고, 교환하기 위한 표준 언어로써, XML로 표현된 문서를 관계형 데이터베이스 관리 시스템(RDBMS:Relational Database Management System)에 저장하고 관리하는 기법에 대한 연구가 활발히 진행되어 왔다. 이러한 연구들은 DTD(Document Type Definition)를 입력받아 해당 DTD에서 관계형 스키마를 추론하는 기법을 사용하고 있다. 하지만, 기존의 연구들은 DTD 간소화(simplification) 절차를 적용하기 때문에 DTD 내에서 추론할 수 있는 의미적인 부분들이 스키마 생성 시에 보존이 되지 못한다. 또한, 기존의 연구들은 XML 데이터의 내용(content)와 구조(structure) 정보만을 저장하는데 초점이 맞춰져 있기 때문에, XML 문서 저장 시 데이터의 무결성을 보장하기 위해 저장 프로시져나 트리거를 사용해야 하는 번거로움이 생긴다.

본 논문에서는 [1]의 연구에서 제시한 Inlining 기법을 기반으로 기존의 Inlining 기법의 문제점인 DTD에서 추론할 수 있는 의미적인 부분의 손실을 관계형 스키마로 보존하는 방법과 효율적인 릴레이션 생성을 위해 개선된 Inlining 기법을 제시한다.

목 차

표 목차

그림 목차

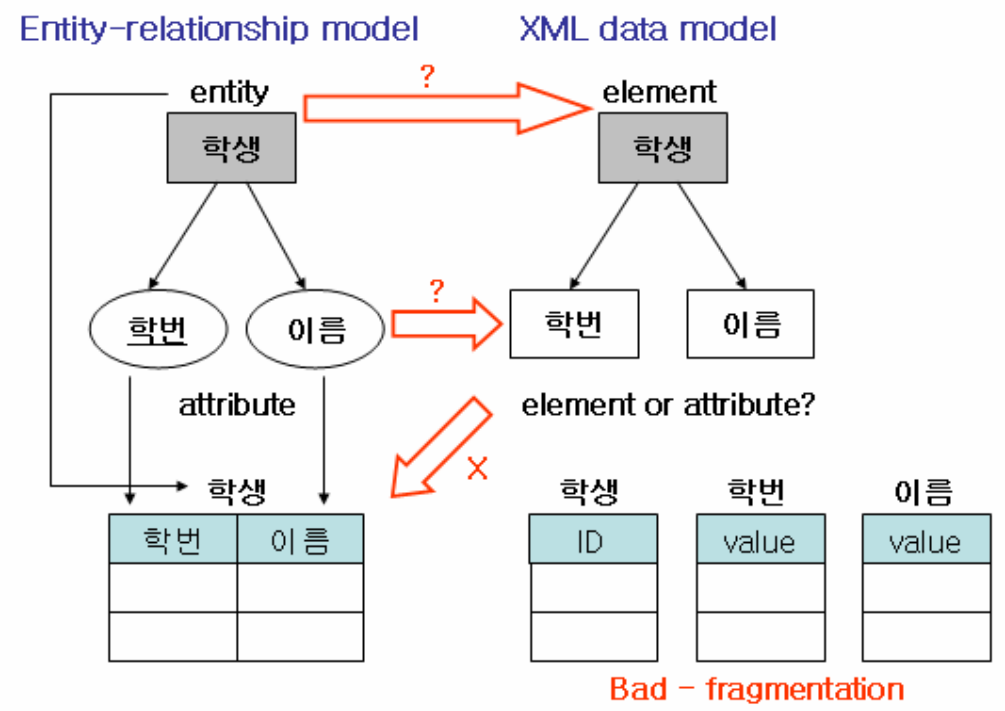
제 1장 서 론

1.1 연구배경

1998년 W3C에서 기존의 인터넷에서 사용되던 HTML(Hypertext Markup Language)의 한계를 극복하고 SGML(Standard Generalized Markup Language)의 복잡함을 해결하는 방안으로 XML을 발표한 이후, 인터넷 상에서 XML로 표현된 데이터들이 많아짐에 따라 이를 효율적으로 저장하고 관리하는 기법들에 대한 연구가 활발히 진행되었다.

이러한 연구들은 주로 관계형 데이터베이스 관리시스템을 이용하여 XML을 저장하는 기법[7][11], 파일 시스템이나 객체지향 DBMS(OODBMS:Object-Oriented Database Management System)로 저장하는 기법[12], 그리고 반구조적인 시스템(Semi-structured System)에 저장하는 기법[6]등으로 연구되어 왔다.

RDBMS가 OODBMS나 반구조적인 시스템보다는 많은 성숙된 기술들을 제공해주기 때문에 RDBMS에 XML 문서를 저장하는 기법이 많이 연구되어 왔으며, 대표적인 기법으로는 연구 [7], [11]에서 제시된 Inlining 기법이 존재한다. 이러한 Inlining 기법은 [그림 1.1]에서 볼 수 있듯이 기존의 RDBMS에서의 데이터 모델과 XML에서의 데이터 모델의 차이에서 오는 불합리한 스키마 생성을 극복하기 위해 제안되었다.



[그림 1.1] RDBMS, XML 데이터 모델

[그림 1.1]에서 볼 수 있듯이 RDBMS의 데이터 모델인 E-R 모델의 entity, attribute가 XML 데이터 모델의 element, attribute로 일대일로의 매핑이 이루어 지지 않기 때문에 최악의 경우 각각의 엘리먼트 별로 릴레이션이 생성이 될 수 있고, 많은 조인 비용을 발생시키게 된다. 그래서 XML 데이터 모델로부터 효율적으로 릴레이션을 추론하기 위한 특별한 기법이 필요하게 되었고, 이러한 연구로써 제안된 기법이 Inlining 기법이다.

Inlining 기법에서는 Input으로 XML의 데이터 모델을 함축하고 있는 DTD를 받아서 해당 DTD로부터 관계형 스키마를 추론해 내는 알고리즘을 사용한다. 이 때, DTD로부터 관계형 스키마에 반영되어야 하는 정보로는 문서의 내용(XML 문서의 실제 데이터), 문서의 구조(XML 무서의 엘리먼트들의 부모-자식 관계 정보), 문서의 의미(XML 문서의 키 제약조건 정보)가 보

존되어야 한다[15].

그러나, 기존의 Inlining 기법에서는 XML 데이터의 내용이나 구조를 저장하는 기법에 초점이 맞추어져 있기 때문에 DTD에서 추론해 낼 수 있는 의미적인(semantic) 부분들을 관계형 스키마에 반영하지 못한다. 또한, 정확한 XML 문서의 의미를 반영하지 못하는 이유 중에 하나는 기존의 Inlining 기법에서 Inlining 알고리즘을 적용하기 전에 DTD 간소화 절차를 거치는 과정에서 DTD의 지나친 일반화로 인해 DTD에서 추론할 수 있는 의미적인 부분들이 손실된다. 반면에 [2], [15], [16]에서 제시한 저장 기법은 XML 데이터의 의미적인 부분을 관계형 스키마로 보존할 수 있는 방법을 보여주지만, 명시적으로 DTD 간소화 절차를 보여주지 못하고 있다. DTD 문서는 같은 의미의 XML 문서 구조를 정의하더라도 작성자의 성향에 따라 다양한 종류의 DTD 문서가 생성될 수 있다. 이러한 특성 때문에 각 논문에서 제시된 기법들이 생성될 수 있는 모든 DTD의 다양한 형태를 고려한다는 것은 불가능하다. 이에 따라 Inlining 기법을 제시하는데 있어서 다양한 DTD 문서에 대한 공통적이고 일반화된 절차를 제시하지 않는다면 해당 기법은 다양한 DTD에 대해 제대로 동작하지 않을 가능성이 커지게 된다.

1.2 연구방향

본 논문은 웹 상의 데이터를 표현한 XML 문서를 RDBMS로 저장하기 위해서 자바 기반의 미들웨어인 XML-DBMS[1]를 수정하여 [11]의 연구에서 제시하는 Hybrid Inlining 기법을 구현하고, 앞에서 기술한 기존 연구의 문제점들을 해결하기 위한 개선된 Inlining 기법을 제안한다.

기존의 제시된 Hybrid Inlining 기법들의 문제점으로는 다음과 같은 요소

들이 있다.

① DTD 간소화 절차를 거치면서 DTD에서 추론할 수 있는 NULL, NOT NULL, Cardinality Constraints와 같은 DTD의 의미적인 부분들이 손실이 발생할 수 있다.

문제점 - 충분한 의미들을 전달하지 못함으로써 스키마에 반영을 할 수 없기 때문에 실제 XML 데이터를 테이블로 저장할 때 데이터의 무결성을 위해서 저장 프로시져나 트리거를 사용해야 하는 오버헤드가 생긴다.

해결 방안 - 연구 [11]에서 제시된 DTD 간소화 절차를 수정, 정리하여 스키마 생성에 필요한 정보를 손실하지 않으면서 복잡한 DTD 문서에 대해 일반화 작업을 적용한다.

② 연구 [7], [11]에서는 DTD의 의미적인 부분을 관계형 스키마로 매핑시킬 수 있는 방법을 제시하지 않고 있다.

문제점 - ①의 절차에 의해 충분한 정보를 갖는 DTD가 있다 하더라도 각 정보들이 실제로 관계형 스키마의 어떤 부분으로 매핑이 될 수 있는지에 대한 언급이 없다.

해결 방안 - 연구 [2]에서 제시되고 있는 방법을 이용하여 DTD에서 추론된 정보를 SQL의 UNIQUE, NULL, NOT NULL, 외래키 등과 같은 제약조건을 이용하여 관계형 스키마로 의미적인 부분을 보존할 수 있다.

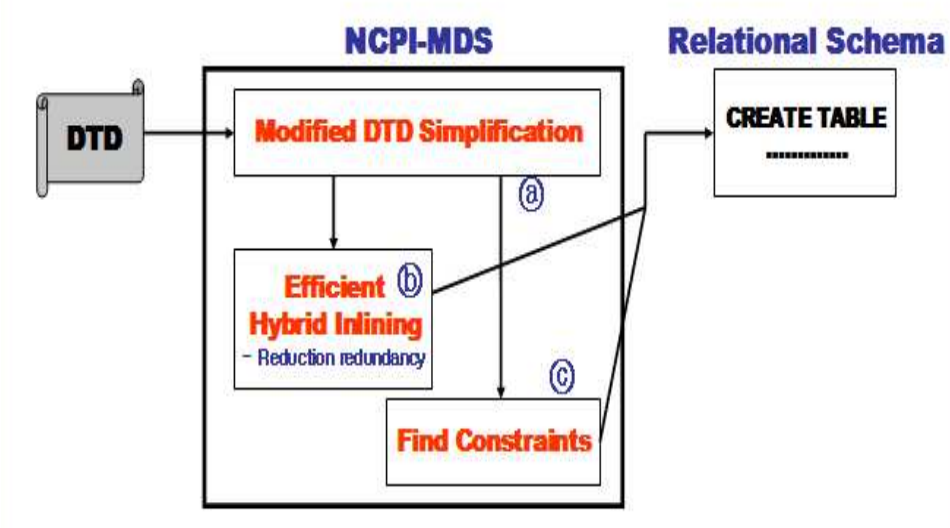
③ 연구 [2]에서는 DTD로부터 추론할 수 있는 의미적인 부분을 보존을 할 수 있지만, 효율적인 릴레이션 생성에 관한 고려가 없기 때문에 연구 [11]에서 제시한 기법보다 더 많은 릴레이션이 생성이 되고, 데이터의 중복이 많아진다.

문제점 - 연구 [2]은 기본적으로 DTD의 의미적인 부분을 보존하는데 초점이 맞추어져 있기 때문에 기본적인 Inlining 알고리즘은 연구 [7]의 기법을 따른다. 연구 [7]에서 제시하는 Inlining 기법은 많은 릴레이션을 생성할 뿐만 아니라, 데이터의 중복도 많이 발생한다.

해결 방안 - 연구 [11]에서 제시된 기법에서는 생성되는 테이블 중 기본키만 가지는 테이블을 하나의 테이블로 통합한다거나, DTD 대응수 규칙 중 “*” 연산자를 다루기 위해 하나의 추가적인 테이블을 추가함으로써 효율적인 릴레이션 생성을 할 수 있다. 여기서, DTD 대응수 규칙은 자식 엘리먼트가 얼마나 나타날 수 있는가에 관한 규칙을 의미하고, “*” 연산자는 자식 엘리먼트가 나타나지 않던가 원하는 수만큼 나타날 수 있다는 것을 의미한다.

본 논문에서는 위의 문제점을 해결하기 위해 Inlining 알고리즘 전에 적용되는 DTD 간소화 절차를 수정하여 DTD로부터 추론할 수 있는 의미적인 부분들을 보존한 채로 Inlining 알고리즘을 적용하고, 보존된 의미적인 부분들을 스키마로 매핑할 수 있는 절차를 제시하며, 연구 [2]에서 고려하지 못한 효율적 릴레이션 생성 문제에 관한 대안을 제시한다. [그림 1.2]는 본 논문에서 DTD로부터 관계형 스키마를 추론하기 위해서 제시하는 NCPI-MDS(New Constraints-Preserving Inlining with Modified DTD

Simplification)의 접근방법이다.



[그림 1.2] NCPI-MDS 기법

NCPI-MDS 기법에서는 기존의 Inlining 기법과 동일하게 Input으로 DTD를 받고, DTD에서 추론할 수 있는 의미적인 부분을 보존하기 위해 수정된 DTD 간소화 절차를 거치게 된다(ⓐ). 다음으로, 의미적인 부분들이 보존된 DTD를 통해 관계형 스키마를 추론하기 위해 기존의 Hybrid Inlining 기법에서 데이터의 중복을 줄이고, 효율적 릴레이션 생성을 하기 위해 수정된 EHI(Efficient Hybrid Inlining) 알고리즘을 적용하고(ⓑ), 이와 동시에 DTD 내의 의미적인 부분들을 추론해 냄으로써 이를 결과 관계형 스키마에 반영하게 되는 과정을 거친다(ⓒ).

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 살펴보고, 3장에서는 Hybrid Inlining 기법에 대해 기술하고, 4장에서는 NCPI-MDS 기법을 제안하고, 5장에서는 기존 기법과 제안 기법에 의해 생성되는 릴레이션에 대한 비교분석을 한다. 마지막으로 6장에서는 본 논문의 결론을 맺는다.

제 2장 관련 연구

본 장에서는 XML, DTD의 소개와 XML 문서를 RDBMS를 이용하여 저장하는 기존의 기법들에 대해 살펴본다.

2.1 XML(eXtensible Markup Language)

XML의 역사는 1996년으로 거슬러 올라간다. XML은 W3C(World Wide Web Consortium)가 제안한 것으로 웹 상에서 구조화된 문서를 전송 가능하도록 설계된 표준화된 텍스트 형식이다.

■ HTML (HyperText Markup Language)

HTML은 원래 하나의 SGML 애플리케이션으로서 WWW(World Wide Web) 상에서 어떤 문서가 표시되는 방식을 규정한 것이다. HTML은 SGML 규칙들, 즉 어떤 태그가 어떤 의미를 가지는지에 대한 약속들의 집합일 뿐이며, 그러한 규칙은 공식 DTD 문서에 담겨 있다. HTML의 경우 DTD는 웹 브라우저 자체에 내장되는 경우가 일반적이며, HTML의 규칙은 SGML에 비해 매우 간단하고 크기도 작기 때문에 현재는 일반적인 최종 사용자들까지도 HTML 문서를 쉽게 만들어서 사용하고 있다.

HTML은 1991년 Tim Berners-Lee라는 학자가 만든 것인데, 자신의 기술 논문들을 쉽게 작성하고, 또 다른 기종을 사용하는 학계의 사람들이 볼 때에도 원래의 문서 모양을 그대로 유지할 수 있도록 하는 것이 목적이었다. 즉, 문서의 구성이나 양식을 설명할 수 있는 몇 가지 태그들을 만들고 그것들을

이용해서 문서를 작성하면 다른 컴퓨터에 문서를 전송해도 별다른 어려움 없이 문서가 원래 담고 있는 내용과 양식을 그대로 표현할 수 있을 것이라는 생각이었다.

HTML은 문서를 인터넷 상에서 전송할 때 HTTP(HyperText Transfer Protocol)라는 프로토콜을 사용한다. HTTP는 인터넷에서 사용되는 여러 프로토콜들, 인터넷 프로토콜 수트(Internet Protocol Suit) 또는 흔히 TCP/IP라고 하는 일련의 프로토콜들 중 하나이다. HTTP가 인기를 끈 이유는 한 문서에서 다른 문서로 연결(Link)하기가 무척 간단한다는 점이었다. 거기에 HTTP를 사용하는 HTML의 강력함과 단순성이 결합됨으로써, HTML과 HTTP는 인터넷을 대중화한 일등공신이 되었다.

물론 HTML이 장점만을 가지고 있는 것은 아니다. HTML은 다음과 같은 단점을 가진다.

- HTML은 태그가 한정되어 있다. 사용자가 스스로 만든 태그로 문서를 꾸며서 다른 사용자들에게 보여줄 수가 없다.

- HTML은 표현을 위한 기술이다. 태그에 포함된 내용의 의미를 전달하기에는 부적합하다.

- HTML은 "평면적(flat)"이다. 태그들의 중요도를 직접 지정할 수 없으므로 데이터의 계층 구조를 표현할 수 없다.

- 브라우저가 애플리케이션 플랫폼으로 사용되고 있는 상황임에도 불구하고 HTML 자체는 현재 개발자들이 추구하는 수준에서의 진보적인 웹

애플리케이션을 만드는데 필요한 기능들을 제공하지 못한다.

● 네트워크 체증 문제, 애플리케이션의 구성 요소로 쓰이고 있는 현재의 HTML 문서들은 클라이언트/서버 간 통신의 체증을 가중시키고 있다.

시간이 지날수록 사용자들은 좀 더 창의적이고 특수한 정보를 자신만의 방식으로 웹에 표현하고 싶어 하게 되었고, HTML 자체의 한계를 극복하려 하기 보다는 여러 가지 스크립트 언어나 DHTML(Dynamic HTML), 채널 같은 우회적인 방법을 통해서 해결책을 찾으려고 하게 되었으며, 결과적으로는 '브라우저 간의 호환성 부재'라는 심각한 문제가 발생하는 상황을 초래하였다.

■ SGML (Standard Generalized Markup Language, 표준화된 범용적 마크업 언어)

1986년 국제 표준(ISO 8879)로 제정된 마크업 언어이다. SGML의 개념은 1960년대 후반부터 존재하였으며 언어의 각 요소(element)와 속성(attribute)에 대한 정규적 정의를 작성하는 방식을 통해서 자신만의 태그를 규정할 수 있도록 하는 데 필요한 메타 언어, 즉 '언어의 언어'라고 할 수 있다. SGML에 근거해 만들어진 HTML의 획기적인 성공(Web의 확장)에 힘입어 현재는 SGML 이외의 메타 언어는 거의 언급되지 않고 있다.

SGML의 목적은 텍스트, 이미지, 오디오 및 비디오 등을 포함하는 멀티미디어 전자 문서들을 이기종 시스템들 간에 정보의 손실 없이 효율적으로 전송, 저장 및 자동 처리하는 것이다. SGML은 문서의 논리 구조와 내용을 기술하기 위한 언어로 CALS, EC 등 웹의 공개된 표준 체계로 정착되어 많이

사용되고 있다. 그리고 시스템이나 플랫폼에 독립적으로 동작하고 문서의 구조를 저장할 수 있기 때문에 문서 구조를 기반으로 한 검색 저장 등의 다양한 응용에 사용할 수 있다.

SGML은 매우 강력하긴 하지만 그만큼 복잡하며 여러 기능들 중 많은 것들은 거의 쓰이지 않는다. 또한 하나의 SGML 문서는 그 자신만으로는 해석될 수 없으며 마크업 언어의 정의가 담겨있는 DTD(Document Type Definition : 문서 원형 정의)라는 개별적인 문서가 붙어 다녀야 한다. DTD는 SGML에 들어 있는 언어에 대한 모든 규칙들을 담은 것이다. SGML을 해석하는 프로그램은 SGML 문서와 함께 제공된, 또는 SGML 문서 안에 포함된 DTD를 이용해서 SGML 문서의 사용자 정의 태그들을 해석하게 된다.

■ XML(확장성 마크업 언어)

브라우저의 주요 개발사들이 SGML 자체를 완전하게 지원하지는 않을 것이라는 점은 거의 명백하다. 그리고 사실 지원한다고 해도 SGML이라는 언어는 소수 전문가들 이외의 사람들까지 널리 사용할 수 있을 만한 언어가 아니다. 그래서 SGML처럼 문서의 내용에 맞게 태그들을 정의할 수 있으면서도 SGML 보다는 훨씬 더 단순한 웹 전용 마크업 언어가 필요하다는 요구들이 자연스럽게 일어났다. W3C(World Wide Consortium)는 그러한 요구들을 받아들여서 그에 관련한 프로젝트를 지원하기로 결정했으며, 썬 마이크로시스템의 Jon Bosak이 이끄는 SGML 전문가 그룹이 웹 공동체가 받아들일 만한 SGML의 부분 집합을 만드는 작업에 착수했다. XML의 사양은 훨씬 간략하다. SGML 사양문서는 무려 500페이지 이상이지만, XML의 경우에는 단 26페이지이다.

HTML은 태그의 종류가 한정되어 있는 반면 XML은 문서의 내용에 관련된 태그를 사용자가 직접 정의할 수 있으며 그 태그를 다른 사람들이 사용하도록 할 수 있다. XML은 본질적으로 다른 언어를 기술하기 위한 언어, 즉 메타언어인 것이다.

다음은 하나의 XML문서가 제대로 표시되기 위해 필요한 필수적인 구성요소 또는 절차들이다.

● DTD (Document Type Definitio, 문서 원형 정의)
- 다른 사용자가 한 사용자가 작성한 태그들의 의미를 파악할 수 있도록, 또는 XML 문서가 태그정의를 참조할 수 있도록 하는 선언 파일이다.

● Style Sheet
- XML에는 없는 XML이 표시되는 방식에 대한 규정을 담고 있는 확장성 양식시트 언어(eXtensible Stylesheet Language, XSL) 또는 계단식 스타일 시트(Cascading Stylesheet, CSS) 메커니즘을 사용한다.

● 확장성 연결 언어 (eXtensible Linking Language)
- HTML의 링크 메커니즘은 상당히 제한적이다. XML은 HTML 식의 기본적인 링크방식을 바꾸지는 않는다. 다만 여기에 좀 더 획기적인 링크 연결 메커니즘을 추가하는데, XML의 링크 연결 사양은 XLinks와 Xpointers라는 두 가지 부분으로 구성되어 있다. Xlinks는 문서들 사이의 일 대 다, 다 대 일 관계를 만들게 할 수 있으며, Xpointers는 문서들의 특정 부분만을 서로 연결하게 할 수 있다.

● 파서 (Parser, 해석기)

- XML을 다루기 위한 시스템은 XML 처리기와 응용 프로그램으로 구성된다. 첫번째 부분인 XML 처리기(proccessor)는 우선 XML 파일이 사양(Specification)을 지키는지 검사한다. 그런 다음 컴퓨터가 XML 파일을 해석하는데 필요한 문서 트리라는 것을 생성해 낸다. 이 문서 트리를 이용해서 컴퓨터는 처리 지시들을 차례로 뽑아낸다. 파서는 XML 처리기의 역할을 담당하며, 응용 프로그램이란 파서가 뽑아낸 처리 지시에 따라 트리의 데이터를 처리(표시, 조작 등등)하는 부분을 말한다.

이외에도 이름공간(Namespace)이나 XML 데이터(XML-Data)나 문서 내용 정의(Document Content Definition) 같은 좀 더 진보적인 기술들이 존재한다. XML의 장점을 정리하면 다음과 같다.

● 정보 제공자는 자기 마음대로 새로운 태그 세트와 속성을 정의할 수 있다. 즉 사용자가 자신의 편의에 따라 혹은 자신의 데이터를 구분하고자 새로운 태그 세트를 임의로 만들 수 있다.

● 문서의 구조는 연속적인 중첩을 허용한다. 즉, XML은 HTML이 지원하지 않는 객체 지향적 구조 혹은 데이터베이스 스키마의 구성을 위해 필요한 여러 번의 중첩을 허용하고 있다.

● 문서 구조의 검증이 필요한 어플리케이션을 위하여 문법적인 구별을 문서 안에서 제공할 수 있다. 즉, 어플리케이션이 어떠한 문서를 받아들일 때 그 문서의 오류를 쉽게 판단할 수 있게 된다.

● 구조 검색 및 전문 검색이 가능하다

● DTD를 이용하여 문서의 논리적 구조를 다양한 형식으로 표현이 가능하다. 또한 하나의 문서로 각각의 목적에 맞게 스타일 시트를 적용시켜서 정보를 재가공할 수 있다.

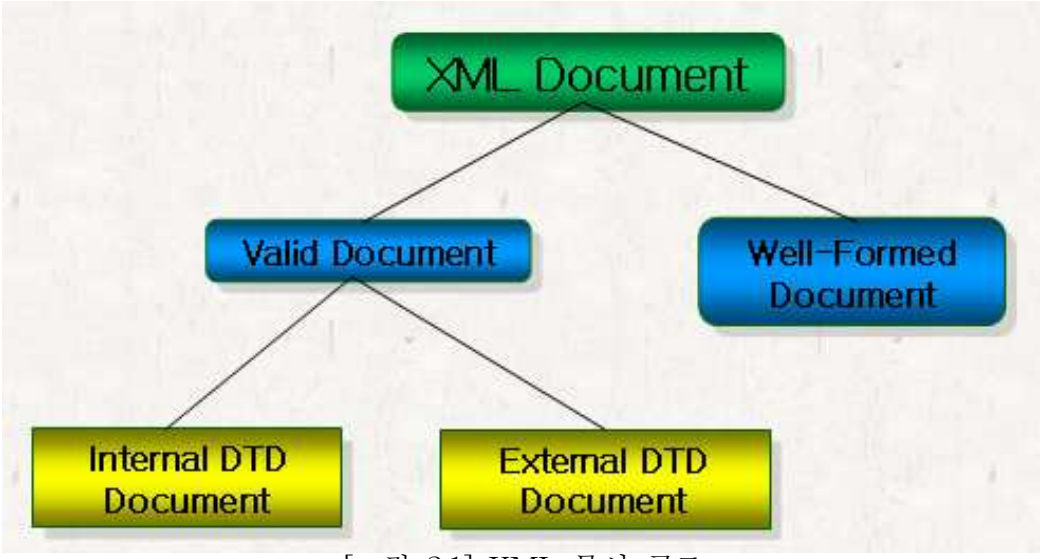
● 양방향 링크, 다방향 링크의 지원이 가능하다.

HTML, SGML, XML을 각 비교항목에 따라 나열하면 [표 2.1]과 같다.

비교항목	HTML	SGML	XML
태그사용	사용자 정의가 불가능하고 제한적.	사용자 정의가 무제한적으로 가능.	사용자 정의가 SGML보다 제한적으로 가능.
문서의 재사용	불가능.	가능.	가능.
응용분야	단순문서 내용이 너무 길지 않은 문서.	방대한 내용과 구조를 요하는 기술 문서.	SGML과 동일 웹상의 교환문서
문서작성	문서작성은 간단하나 문서의 논리구조를 작성하는 것은 어려움.	사전과 같이 복잡함.	SGML보다 단순함.
문서검색	비효율적.	정확한 검색이 가능 구조검색이 가능	SGML과 동일
링크	HTML	HyTime	XLL
출력형식언어	CSS	DSSSL	X니

[표 2.1] HTML, SGML, XML 비교분석

XML 문서의 구조는 [그림 2.1]과 같다.



[그림 2.1] XML 문서 구조

[그림 2.1]에서 Well-Formed Document는 해당 XML 문서가 문법상에는 오류가 없지만 문서와 관련된 DTD에 대해서는 전체 논리 구조가 유효하지 않은 문서를 의미한다. 이 개념이 유효한 이유는 DTD를 읽는 데에는 많은 시간과 대역폭이 필요하기 때문이다. 문서가 문법적으로 유효하며 손상되지 않았다는 것을 미리 보장한다면 많은 시간을 절약할 수 있게 된다. Well-Formed Document는 다음의 조건을 갖추어야 한다.

- 전체적으로 문서의 양식과 맞아야 한다.
- 하나 또는 그 이상의 엘리먼트를 포함해야 한다.
- 최상위에는 root 또는 document element라 불리는 하나의 엘리먼트만 표시되어야 한다.
- XML 스펙에 서술된 모든 규약을 만족해야 한다.

● 문서 내에서 직간접적으로 참조되는 엔티티 파싱이 잘 구성되어야한다.

이에 반해 [그림 2.1]에서 Valid Document는 반드시 DTD를 구성해야 할 뿐만 아니라 각각의 엘리먼트들은 DTD에 정의된 규칙을 따라야 하는 등 모든 규칙을 반드시 준수해야 한다.

2.2 DTD (Document Type Definition)

XML 문서에서 사용되는 마크업을 선언한 문서가 DTD이다. DTD는 문서에 대한 논리, 물리 구조를 정의하게 되며, 다음과 같은 사항을 정의하게 된다.

● Content model과 더불어 문서에서 허용되는 엘리먼트의 타입을 정의한다.

● 각 엘리먼트에 할당되어 있는 속성을 정의한다.

● 문서에 허용되는 entity를 정의한다.

● 외부 entity와 함께 사용되는 표기법을 정의한다.

XML 개발자들이 DTD에 따라 응용 프로그램을 작성한다는 것은 애플리케이션 개발 관점에서 보면 데이터베이스 개발자들이 DB 스키마에 따라 응용 프로그램을 작성하는 것과 같다. 즉 DTD는 XML 문서 안에 결합되거나 외부에 독립적으로 존재할 수 있는 문서이고, 해당 DTD에는 XML 문서가 어떻게 구조화되고 어떤 요소를 포함해야 하는지, 어떤 종류의 데이터가 포

함되어야 하는지, 기본값이 무엇인지에 대한 규칙을 정의하고 있다. XML 문서는 이러한 DTD 문서를 하나만 가진다는 제한을 가진다.

2.2.1 DTD의 기본 예약어

예약어	설명
ELEMENT	XML 엘리먼트 타입의 이름과 그 요소의 허용 가능한 하위 요소를 선언한다.
ATTLIST	XML 요소 속성 이름과 그 속성에 허용되는 기본 속성 값을 선언한다.
ENTITY	특별한 문자 레퍼런스, 텍스트 매크로 및 외부 소스로부터 가져온 반복되는 컨텐츠들을 선언한다.
NOTATION	외부 비 XML 컨텐츠와 이 컨텐츠를 다루는 외부 애플리케이션을 선언한다.

[표 2.2] DTD 예약어

2.2.2 엘리먼트 타입 선언(element type declarations)

DTD 내에서의 엘리먼트 타입을 정의할 때의 형식은 [그림 2.2]와 같다.

<!ELEMENT element_name (content_model)>

[그림 2.2] 타입 선언 형식

엘리먼트 선언 시 정의할 수 있는 content_model의 종류는 [표 2.3]과 같이 정리할 수 있다.

content_model	설명
|	엘리먼트의 내용을 선택적으로 나타낼 수 있다. 즉, 엘

	리먼트의 선택적인 사용이 가능하다. 예) <!ELEMENT desert (ice_cream | coffee)> XML 문서 : <desert> <ice_cream>vanilla</ice_cream> </desert>
,	엘리먼트의 내용이 정의되어진 순서대로 나타나야 한다. 예) <!ELEMENT order (appetizer, desert)> XML 문서 : <order> <appetizer>Cream</appetizer> <desert>ice_cream</desert> </order>
?	엘리먼트가 생략될 수 있거나 기껏해야 한번만 나타날 수 있다. 예) <!ELEMENT order (appetizer?, desert)> XML 문서 : <order> <desert>ice_cream</desert> </order>
*	엘리먼트가 생략될 수 있거나 여러 번 나타날 수 있다. 예) <!ELEMENT order (appetizer*, desert)> XML 문서 : <order> <appetizer>Cream</appetizer> <appetizer>Soup</appetizer> <desert>ice_cream</desert> </order>
+	엘리먼트가 여러 번 나타날 수 있고, 최소 한번은 나타나야 한다. 예) <!ELEMENT order (appetizer+ , desert+)> XML 문서 : <order> <appetizer>Cream</appetizer> <appetizer>Soup</appetizer> <desert>ice_cream</desert> </order>

()	엘리먼트의 내용을 그룹화 시킨다. 예) <!ELEMENT ice_cream (vanilla | chocolate)*> XML 문서 : <ice_cream> </ice_cream>
#PCDATA	TEXT DATA 만이 올 수 있다. 정의된 엘리먼트는 글자 데이터만을 포함할 수 있다. PCDATA도 CDATA처럼 문자열을 포함하지만 PCDATA는 파서가 파싱을 한다. 예) <!ELEMENT name (#PCDATA)> XML 문서 : <name>Frank</name>

[표 2.3] content model

2.2.3 애트리뷰트 선언

애트리뷰트는 엘리먼트를 설명하는 속성이다. 파서에게 XML 문성에 있는 애트리뷰트가 특별한 의미를 갖고 있다는 것을 보이기 위해서 DTD에 필요한 애트리뷰트를 선언하게 된다. 일반적으로 애트리뷰트 리스트의 선언은 4가지 측면에서 기술하게 된다.

● 애트리뷰트와 관련된 엘리먼트

● 애트리뷰트의 이름

● 애트리뷰트의 타입

● 애트리뷰트의 값이 없을 경우 파서의 역할

이러한 애트리뷰트를 정의할 때 허용 가능한 타입 및 형식은 [표 2.4]와

같다.

타입	형식
문자열	형식 - <!ATTLIST element_name attribute_name CDATA default> 정의 - <!ATTLIST book ISBN CDATA "89"> 예제 - <book ISBN="89">The Hobbit</book>
열거형	형식 - <!ATTLIST element_name attribute_name (value | value1 | value2 | ...)> 정의 - <!ATTLIST book type (a1|a2)> 예제 - <book type="a1">The Hobbit</book> <book type="a2">The Hobbit</book>
엔티티	형식 - <!ATTLIST element_name attribute_name ENTITY default> 정의 - <!ENTITY im1 SYSTEM "im1.gif" NDATA gif> <!ATTLIST book img ENTITY default> 예제 - <book img="im1">The Hobbit</book>
ID, IDREF,IDREFS	형식 - <!ATTLIST element_name attribute_name ID default> <!ATTLIST element_name attribute_name IDREF default>

	<!ATTLIST element_name attribute_name IDREFS default> 정의 - <!ATTLIST book cdnum ID default> <!ATTLIST part ref IDREF default> <!ATTLIST part_one IDREFS default> 예제 - <book cdnum="a106">The Hobbit</book> <part ref="a106">part one</part> <part_one ref="a106 b204">a</part_one>
NMTOKEN, NMTOKENS	형식 - <!ATTLIST element_name attribute_name NMTOKEN default> <!ATTLIST element_name attribute_name NMTOKENS default> 정의 - <!ATTLIST car license NMTOKEN default> 예제 - <car license="HDF321-00">Ben</car>

[표 2.4] 애트리뷰트 타입 및 형식

[표 2.4]에 나열된 타입의 의미는 다음과 같다.

● 문자열 - 프로그래밍 언어에서의 문자열과 같다. 다만 XML에서는 문자열의 길이는 정의하지 않는다.

● 열거형 - 애트리뷰트의 값이 나열된 값 중에서 하나만 선택이 가능하다는 것을 의미한다.

● Entity, Entities - Entity 키워드는 파서에게 파싱되지 않는 객체를 찾게 되고, 이를 애트리뷰트의 값으로 취급하게 한다. 반면, Entities는 파서에게 특정 애트리뷰트가 하나 이상의 객체를 가진다는 것을 알려주게 된다.

● ID, IDREF, IDREFS - ID 애트리뷰트는 엘리먼트의 유일한 식별자이다. 애트리뷰트가 ID로 정의가 되어 있다면 해당 애트리뷰트의 값은 XML 문서에서 모든 엘리먼트에 대해서 유일한 식별력을 가져야 한다. IDREF는 문서 내부에서 선언된 ID 값을 참조하게 된다. 반면, IDREFS는 여러 개의 ID 값을 참조한다는 것을 제외하고는 IDREF와 동일하다.

● NMTOKEN, NMTOKENS - 공백글자를 허용하지 않는다는 점을 제외하고는 문자열과 동일하다. 예를 들어 자동차번호, 주민등록번호, 신용카드번호와 같이 공백 없는 데이터의 입력에 유용하다. NMTOKEN이 하나의 NMTOKEN 값을 취할 수 있는 반면 NMTOKENS는 여러 개의 NMTOKEN 값을 취할 수 있다.

앞서 언급한대로 애트리뷰트의 타입을 정의하는 것 이외에 애트리뷭의 디폴트 값을 정의할 수 있는 방법은 [표 2.5]와 같다.

정의	설명
#REQUIRED	애트리뷰트가 값을 반드시 가져야 한다는 것을 의미한다.
#IMPLIED	애트리뷰트가 값을 반드시 가져야 하는지를 결정하지 못한 경우에 사용한다. 애트리뷰트 값의 유무에 상관없이 에러가 발생하지 않는다.,
#FIXED	애트리뷰트를 정의한 후에 XML 문서 내에서 애트리뷰트

	값을 갖는 어떤 다른 값으로 변경할 수 없도록 한다. 만일 XML 문서 내부에서 애트리뷰트의 값을 공백으로 두면 디폴트 값을 사용한다. 예) <!ATTLIST person mood CDATA #FIXED "Happy"> <person mood="Happy">man</person> 또는 <person>man</person>
Supplied	Supplied 애트리뷰트를 사용하게 될 경우 XML 문서 내에서 값을 제공하지 않으면 DTD가 값을 제공하게 된다. 예) <!ATTLIST hat color (red | green | blue) "red"> <hat color="blue"></hat> 또는 </ hat>

[표 2.5] 애트리뷰트의 디폴트 값

2.3 Inlining 기법

[3]의 연구에서는 Basic, Shared, Hybrid Inlining의 세 가지 기법을 제안하고 있다. 각 Inlining 기법을 요약하면 다음과 같다.

2.3.1 Basic Inlining

● DTD 그래프에서 모든 노드들에 대해 자식 노드들을 inline 시킨 후, 각 노드들에 대해서 릴레이션을 생성한다.

● “*” edge(부모 엘리먼트와 자식 엘리먼트가 “*” 연산자를 사용해서 정의되어 있는 경우)로 연결되는 자식 노드는 새로운 릴레이션으로 생성한다.

● 두 개의 노드 사이에 recursive가 존재하면 둘 중 하나의 엘리먼트를 새로운 릴레이션으로 생성한다.

2.3.2 Shared Inlining

● in-degreee edge(DTD 그래프에서 노드로 들어오는 간선)의 수가 1보다 큰 노드는 새로운 릴레이션으로 생성한다.

● in-degree edge가 0인 노드는 새로운 릴레이션으로 생성한다.(root 엘리먼트)

● “*” edge로 연결된 자식 노드는 새로운 릴레이션으로 생성한다.

● 두 개의 노드 사이에 recursive가 존재하면 둘 중 하나의 엘리먼트를 새로운 릴레이션으로 생성한다.

2.3.3 Hybrid Inlining

● "*" edge나 recursive가 존재하지 않고, in-degree edge가 1보다 큰 노드들을 부모 노드로 inline 시킨다는 것을 제외하고는 Shared Inlining 기법과 동일하다.

2.4 확장된 Inlining 기법

[7]의 연구에서 제시하는 Inlining 기법은 특정 DTD로부터 XML 문서의

내용과 구조를 추론하는데 초점을 맞춘 기법이다. 이에 반해, 연구 [2][15]에서 제시하는 기법은 연구 [7]에서 제시한 Inlining 기법을 기반으로 DTD 문서에서 의미적인 제약조건들을 생성되는 릴레이션으로 보존하는 기법을 보여주고 있으며, 요약하면 다음과 같다.

2.4.1 X2R Inlining

● Inlining 기법은 연구 [7]에서 제시한 Hybrid Inlining 기법을 사용한다.

● XML key와 keyref 정보를 보존하는 constraint relations라는 개념을 도입하여 키 정보를 보존한다.

● XML 문서의 엘리먼트들 사이의 부모-자식 관계 정보를 보존하기 위해 각 자식 릴레이션에 parent-id 속성을 추가하여 보존한다.

2.4.2 CPI(Constraints-Preserving Inlining)

● Inlining 기법은 연구 [7]에서 제시한 Shared Inlining 기법을 사용한다.

● DTD로부터 Domain, Cardinality Constraints와 Inclusion, Equality-Generating, Tuple-Generating Dependencies와 같은 의미적인 제약조건들을 추론해 낸다.

● 추론된 제약조건들을 기존 Inlining 기법을 통해 추론된 릴레이션에

rewrite한다.

X2R Inlining과 CPI는 DTD로부터 의미적인 제약조건을 찾아내는데 초점이 맞추어져 있기 때문에 데이터의 중복이나 효율적인 릴레이션 생성에 대해서는 고려하지 못하고 있다. 또한, DTD의 간소화 절차가 생략되어 있는데, 이는 각 기법들이 DTD의 복잡성에 대해서는 다루지 못함을 알 수 있다.

제 3장 Hybrid Inlining 기법

본 장에서는 XML 문서의 저장 절차와 기본적인 Hybrid Inlining 기법을 기술한다.

3.1 XML 문서 저장 절차

Hybrid Inlining 기법에서는 기본적으로 XML 문서가 특정 DTD를 준수한다고 가정한다. 즉, 유효한 DTD가 존재한다고 가정하고 해당 DTD로부터 스키마를 추출한다. XML 문서를 RDBMS에 저장하기 위한 절차(스키마를 생성하기 위한 절차 또는 XML 문서의 실제 데이터를 로드하는 절차)는 다음과 같다.

① XML 문서를 저장하기 위해 특정 DTD를 파싱하여 DTD 그래프를 생성한다.

② DTD 그래프를 깊이우선탐색(DFS:Depth First Search)을 통해 Hybrid

Inlining 기법의 inline 규칙에 따라 릴레이션을 생성한다.

③ 해당 DTD를 따르는 XML 문서를 파싱해서 각 데이터를 생성된 테이블로 로드한다.

④ 생성된 테이블에 대해 XQuery와 같은 semi-structured 질의 요청이 있는 경우 해당 질의에 대응되는 SQL 질의문으로 변환한다.

⑤ 선택된 결과 레코드를 XML 문서의 형태로 반환한다.

위의 절차 중 ④, ⑤는 본 논문의 범위를 벗어나므로 더 이상 언급하지 않고, 본 논문에서는 ③번까지의 과정을 기존의 Inlining 기법과 NCPI-MDS 기법으로 구현하며, 각 기법에 의해 생성된 릴레이션을 비교분석하는 것으로 한정한다.

3.2 Hybrid Inlining 알고리즘

기존의 Hybrid Inlining 기법의 세부적인 알고리즘은 다음과 같다.

① DTD 문서를 파싱해서 얻어진 DTD 그래프를 기반으로 릴레이션으로 생성해야 할 필요가 있는 노드를 시점으로 DFS를 실행한다.

② DFS 동안에 맨 처음 방문한 노드는 “VISITED”로 표시하고, 처음 시작한 노드의 모든 자식 노드들을 방문했다면 언마크한다.

③ 만약 마킹되지 않은 노드가 DFS 동안에 방문되었다면 같은 이름으로 엘리먼트 그래프에 새로운 노드로 생성한다. 엘리먼트 그래프에 노드를 추가한 후, 방문 전 노드로부터 방문 중인 노드까지의 간선(edge)을 만든다.

④ 만약, 이미 마킹이 된 노드를 다시 방문하게 된다면, 이전 노드와 방문 중인 노드 사이에는 recursive가 존재하게 되므로 방문 중인 노드에서 이전 노드로의 역방향의 간선을 추가한다.

⑤ 위의 과정을 반복한 후 만들어지는 엘리먼트 그래프를 기반으로 스키마를 생성한다.

Hybrid Inlining 기법에 따라 생성된 엘리먼트 그래프를 기반으로 릴레이션을 생성하는 규칙은 다음과 같다.

① 엘리먼트 그래프의 root 엘리먼트에 대해서 릴레이션을 생성한다. 즉, 0의 in-degree를 가지는 노드.
- root 엘리먼트의 모든 자식 노드들은 릴레이션으로 inline된다.
- “*” 나 recursive가 존재하지 않는 경우에 대해서는 in-degree가 1 이상이어도 inline 시킨다.

② in-degree가 1인 노드는 다른 노드의 속성으로 inline 된다.

③ “*” 아래에 있는 노드는 새로운 릴레이션으로 생성한다.

④ recursive가 존재한다면 관련되 두 노드 중에 하나의 노드를 릴레이션

으로 생성한다.

3.3 정리

본 장에서는 XML 문서를 RDBMS로 저장하기 위해서 연구 [7]에서 제시한 기본적인 Hybrid Inlining 기법을 보여준다. 기본적인 Hybrid Inlining 기법은 DTD에서 추론할 수 있는 많은 의미적인 부분들을 보존하지 못하고, 중복 데이터의 발생, 비효율적인 릴레이션 생성과 관련된 문제를 가진다.

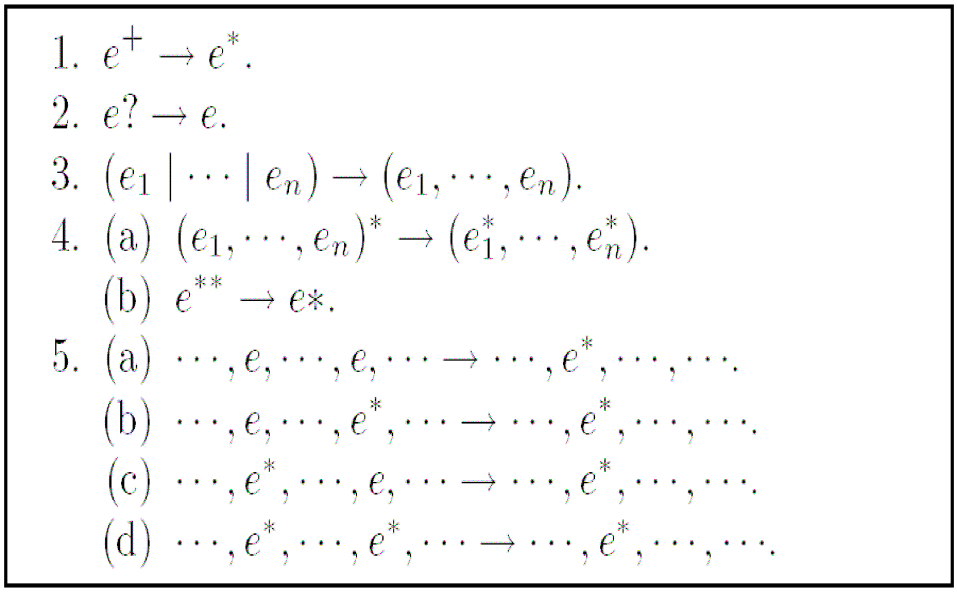
제 4장 NCPI-MDS 기법

본 장에서는 DTD의 의미적인 부분을 관계형 스키마로 보존하기 위해서 DTD 간소화 절차의 개선사항을 제시하고, 기존의 Hybrid Inlining 기법에서의 문제점이었던 데이터의 중복을 제거하면서 효율적인 릴레이션을 생성하기 위한 기법을 제시한다. 또한, DTD 내에서 추론할 수 있는 의미적인 Constraints를 언급하고, 이를 관계형 스키마로 보존할 수 있는 기법을 제시한다.

4.1 DTD 간소화(Simplification)

DTD는 XML 문서의 구조 정보를 기술하기 위해 제시된 방법 중에 하나이다. [11]의 논문은 DTD로부터 관계형 스키마를 추론하기 전에 DTD의 복잡성에 대한 문제를 다루기 위해 DTD 간소화 규칙에 따라 Input DTD를 간

소화시킨다. 그러나, DTD 간소화 규칙을 거치면서 릴레이션 스키마로 보존되어야 하는 정보들이 손실되고, 이로 인해 효율적이지 못한 스키마를 생성하게 된다. [그림 4.1]은 연구 [11]에서 제시되고 있는 DTD 간소화 절차이다.



[그림 4.1] DTD 간소화(Simplification)

[그림 4.1]에는 5가지의 DTD 간소화 규칙이 제시되어 있다. DTD에서 추론할 수 있는 정보임에도 불구하고 [그림 4.1]의 간소화 규칙을 거치면서 정보의 손실을 초래하는 규칙은 1, 2, 3 규칙이다. 우선, 1, 2 규칙은 각각 “+” 연산자를 “*” 연산자로 “?” 연산자를 “default” 연산자로 변환함으로써 대응수 제약조건에 관한 의미가 손실된다. 그리고, 3 규칙에서 “|” 연산자가 여러 엘리먼트 중에서 하나가 자식 엘리먼트로 온다는 의미를 가짐에도 불구하고, “,” 연산자로 변환됨으로써 스키마 생성 시에 비효율적인 릴레이션 생성을 초래한다. 예를 들어, <!ELEMENT A (B | C | D | E)>와 같은 엘리먼트 정의는 엘리먼트 A가 자식 엘리먼트로써 B, C, D, E 중 하나를 가진다라는 의미를 가지는데, 이러한 의미는 관계형 스키마의 NULL, NOT NULL의 개

넘으로 대응시킬 수 있다. 그러나, [그림 4.1]의 3 규칙에 의해 이전 엘리먼트 정의는 <!ELEMENT A (B, C, D, E)>로 변환이 되고, A 엘리먼트가 정의된 순서대로 B, C, D, E의 자식 엘리먼트를 가진다라는 의미의 “,” 연산자의 의미로 변질되어 버린다. 따라서, 변환된 엘리먼트 정의를 가지고 Inlining 기법으로 스키마를 생성하면, [그림 4.2]와 같이 된다.

```
CREATE TABLE A
(
        B VARCHAR(20) NOT NULL
,       C VARCHAR(20) NOT NULL
,       D VARCHAR(20) NOT NULL
,       E VARCHAR(20) NOT NULL
)
```

[그림 4.2] 의미가 상실된 추론 릴레이션

[그림 4.2]의 릴레이션은 원래 DTD의 의미적인 부분을 완전히 보존하고 있다고 볼 수 없다. “|” 연산자의 원래의 의미를 상실하지 않으면서 “|” 연산자를 제거함으로써 DTD의 복잡성을 줄일 수 있는 방법은 두 가지로 생각해 볼 수 있다. 우선, 첫 번째는 <!ELEMENT A (B | C | D | E)>의 정의를 <!ELEMENT A (B?, C?, D?, E?)>로 변환을 하는 것이다. 여기서, “?” 연산자는 A 엘리먼트가 0개 또는 1개의 자식 엘리먼트를 가질 수 있다라는 의미이기 때문에 관계형 스키마 생성 시에 NULL 조건으로 “|” 연산자의 의미를 보존할 수 있다. 이 방법의 단점은 생성된 릴레이션에 많은 NULL 값이 존재한다는 점이다. 다음으로 두 번째 방법은 앞선 방법의 단점인 많은 NULL 값의 발생을 줄일 수 있도록 한 방법이다. “|” 연산자가 자식 엘리먼트 중에서 기껏해야 하나의 자식 엘리먼트를 가질 수 있다는 의미를 가지기 때문에 스키마 생성 시에 이러한 의미를 이용해서 자식 엘리먼트의 데이터를 저장하기 위한 하나의 칼럼을 생성하고 실제로 저장된 데이터가 어떤 엘리먼트에 대한 튜플인지를 구분하기 위한 여분의 칼럼을 추가한다. 즉, [그림 4.3]과 같

은 릴레이션을 생성하게 된다.

```
CREATE TABLE A
(
        BCDE_Column VARCHAR(20) NOT NULL
        nodeType    VARCHAR(20)
,
)
```

[그림 4.3] 의미가 보존된 추론 릴레이션

두 번째 방법으로 "|" 연산자를 다룰 경우에는 DTD 간소화 절차가 아닌 Inlining 알고리즘 자체에서 "|" 연산자를 다루어야 하기 때문에 Inlining 의 구현이 다소 복잡해지는 경향이 있으나, 불필요한 칼럼의 생성을 방지할 수 있고, 그에 따라 NULL 값을 갖는 칼럼의 수를 줄일 수 있다.

그 외에 [그림 4.1]에서 제시되지 않았지만 DTD의 복잡성을 줄이기 위해 추가되어야 하는 규칙들을 포함하여 정리한 DTD 간소화 절차는 [그림 4.4]와 같다.

1. $e^+ \rightarrow e^*$: Cardinality Constraints를 위해 생략
2. $e? \rightarrow e$: Cardinality Constraints를 위해 생략
3. $(e_1 | \ldots. | e_n) \rightarrow (e_1, \ldots.., e_n)$: 효율적 릴레이션 생성을 위해 생략
4. (a) $(e_1, \ldots, e_n)^* \rightarrow (e_1^*, \ldots., e_n^*)$
   (b) $(e_1, \ldots, e_n)^+ \rightarrow (e_1^+, \ldots., e_n^+)$
5. $(e_1 | \ldots | e_n) | (e'_1 | \ldots | e'_n) \rightarrow (e_1 | \ldots. | e_n | e'_1 | \ldots. | e'_n)$
6. (a) $e^{**} \rightarrow e^*$, (b) $e^{*?} \rightarrow e^*$, (c) $e?^* \rightarrow e^*$, (d) $e?? \rightarrow e?$
7. (a) $\ldots, e^*, \ldots, e^*, \ldots \rightarrow \ldots, e^*, \ldots$

[그림 4.4] 수정된 DTD 간소화 절차

[그림 4.4]의 7 규칙은 여러 경우의 수가 존재하게 되는데, 각 경우를 정리하면 [그림 4.5]와 같다.

	*	+	?	default
*	*	*	*	*
+	*	+	*	+
?	*	*	*	*
default	*	+	*	+

[그림 4.5] 규칙 7에 적용 가능한 경우들

[그림 4.4]의 수정된 DTD 간소화 절차에서 남겨진 규칙들 또는 추가된 규칙들은 자명한 규칙들만 남게 된다. 예를 들어, 규칙 7에서 볼 수 있듯이 {0. ...}의 발생이 가능한 엘리먼트와 {0, ...}의 발생이 가능한 엘리먼트를 더하면 당연히 해당 엘리먼트는 {0, ...}의 발생이 가능하다는 하나의 엘리먼트 정의로 간소화될 수 있다. 또한, {0, ...}의 발생이 가능하다는 하나의 엘리먼트 정의로 간소화됐음에도 원래의 의미적인 정보인 {0, ...}의 발생이 가능하다는 것이 손실되지 않는다.

4.2 NCPI-MDS 알고리즘

기존의 Hybrid Inlining 알고리즘을 기반으로 하면서 효율적인 릴레이션 생성을 위해 수정된 NCPI-MDS 알고리즘은 다음과 같다.

① 수정된 DTD 간소화 절차를 거친 DTD를 Input으로 내부적으로 DTD 그래프를 생성한다. 그래프의 각 노드들은 DTD의 엘리먼트, 애트리뷰트를 나타낸다. 각 엘리먼트는 그래프에서 정확하게 한 번만 나타나고, 애트리뷰트와 연산자는 DTD에서 정의되어 있는 만큼 DTD 그래프에서 나타난다.

② DTD 그래프에서 새로운 릴레이션으로 만들어 져야 하는 top nodes를 시작으로 DFS를 실행한다. top nodes[2]의 식별은 다음과 같다.
- DTD 그래프 내에서 어떠한 노드들로부터 도달할 수 없는 노드(최상위 노드)
- 부모 노드에서 자식 노드로 “*” 연산자나 “+” 연산자에 의해 연결된 자식 노드
- recursive가 존재하는 두 노드 중 하나의 노드.

③ top node를 시작으로 DFS를 하는 동안 다른 top node를 만나지 않는 한 모든 도달 가능한 leaf node들을 최초 top node로 inline시킨다. DTD에 정의된 어떤한 엘리먼트도 루트 엘리먼트가 될 수 있으므로 생성된 테이블에 저장된 튜플을 구분하기 위해 nodeType이라는 속성을 스키마에 추가한다.

④ 생성되는 스키마의 속성의 이름은 top node로부터 leaf node까지의 각 노드들의 이름에 “.”를 붙여서 구성한다. 또한, DTD에서 ID 타입의 애트리뷰트가 정의되어 있으면 이를 기본키 속성으로 사용하고, 그렇지 않다면 일련번호를 사용하여 기본키를 생성한다.

⑤ DFS 동안에 “|” 연산자를 사용하는 노드를 만나면, 다음 두 가지 중의 하나의 방법으로 다룬다.

- 정의되어지는 자식 노드가 단말 노드일 경우에는 부모 엘리먼트의 릴레이션에 통합해서 저장하기 위한 하나의 칼럼을 생성하고, 튜플에 실제 저장된 데이터가 어떤 엘리먼트인지를 구분하기 위한 choiceType이라는 칼럼을 추가한다.
- 정의되어지는 자식 노드가 비단말 노드이면서 새로운 릴레이션으로 생성되어지는 경우는 각 자식 노드마다 릴레이션이 생성되는 것을 방지하기 위해 각 자식 노드를 통합해서 저장하는 하나의 릴레이션을 생성하고 실제 저장된 튜플이 어떤 엘리먼트인지를 구분하기 위해 choiceType이라는 칼럼을 추가한다.

⑥ ③ ~ ⑤까지의 절차를 더 이상 추론되는 릴레이션이 없을 때까지 반복한다.

⑦ 추론된 릴레이션 중 T1(ID), T2(ID)의 형태를 갖는 릴레이션이 적어도 두 개 이상 존재한다면, 이들 릴레이션을 TABLE(ID, nodeType)의 형태의 하나의 릴레이션으로 통합시킨다.

⑧ 또한, T1(ID, value1), T2(ID, value2)의 형태를 갖는 릴레이션이 두 개 이상 존재한다면, 이들 릴레이션을 TABLE(ID, nodeType, value)의 형태의 하나의 릴레이션으로 통합시킨다.

⑨ 마지막으로, “*” 연산자에 의해 만들어지는 릴레이션과 부모 릴레이션 사이의 관계정보를 저장하는 asterisk(parentID, childID, parentType, childType)의 릴레이션을 추가한다. 이 릴레이션을 이용해서 DTD내에

서의 “*” 연산자로 정의된 부모-자식 관계정보를 저장한다. 이러한 스키마 생성은 XML 데이터 저장 시에 중복을 제거해 줄 수 있을 뿐만 아니라, 일반적인 기본키와 외래키의 조인에 의한 부모 엘리먼트에서 자식 엘리먼트를 탐색하는 하향식 탐색에 있어서 탐색비용을 단축할 뿐만 아니라 자식 엘리먼트에서 부모 엘리먼트를 탐색하는 상향식 탐색에 있어서도 탐색비용을 줄이는데 도움이 된다.

4.3 DTD 내의 의미적인 제약조건[2]

DTD에서 추론할 수 있는 의미적인 제약조건과 관계형 스키마에서 매핑될 수 있는 제약조건들은 다음과 같다.

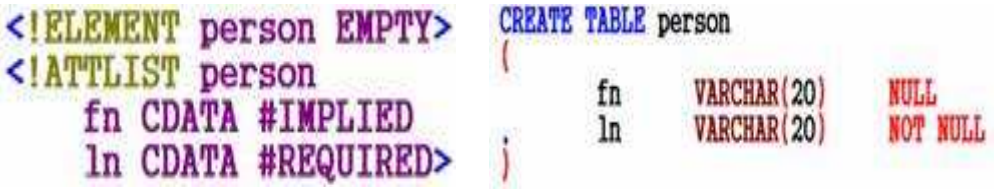
① 도메인 제약조건 - DTD에서 추론할 수 있는 도메인 제약조건은 속성의 값이 특정 집합의 값으로써 한정될 때 확인할 수 있다. <!ATTLIST author gender (male | female) #REQUIRED married (yes | no) #IMPLIED>와 같은 엘리먼트 정의에서 볼 수 있듯이 author 엘리먼트는 속성으로 gender와 married를 가지고, gender라는 속성은 값으로써 “male” 또는 “female” 중에 하나를 갖는다. 또한, married라는 속성은 값으로써 ”yes“ 또는 ”no“ 중에 하나를 가짐을 볼 수 있다. 이러한 특성은 [그림 4.6]에서와 같이 관계형 스키마에서 CHECK 절을 사용하여 보존될 수 있다.

```
CREATE DOMAIN gender VARCHAR(10)
        CHECK (VALUE IN ("male", "female"))

CREATE DOMAIN married VARCHAR(10)
        CHECK (VALUE IN ("yes", "no"))
```

[그림 4.6] SQL CHECK 절

또한, #REQUIRED와 #IMPLIED와 같은 속성은 [그림 4.7]에서와 같이 SQL의 NULL, NOT NULL로 보존이 될 수 있다.



[그림 4.7] #REQUIRED, #IMPLIED 속성의 보존

② 대응수 제약조건 - DTD 내에서 추론이 될 수 있는 대응수는 총 네 가지로써, [표 4.1]과 같다.

타입	대응수	연산자	의미
A	1 대 {0,1}	?	자식 엘리먼트를 0 또는 한 개를 가질 수 있음
B	1 대 {1}	default	자식 엘리먼트를 정확하게 한 개 가질 수 있음
C	1 대 {0, ...}	*	자식 엘리먼트를 0개 이상 가질 수 있음
D	1 대 {1, ...}	+	자식 엘리먼트를 1개 이상 가질 수 있음

[표 4.1] 가능한 Cardinality Relationship

[표 4.1]의 대응수에서 추론할 수 있는 제약조건은 세 가지가 존재한다. 첫 번째는 자식 엘리먼트가 NULL 여부에 대한 제약조건으로써, [표 4.1]에서 A와 C는 NULL Constraints로 보존될 수 있고, B와 D는 NOT NULL Constraints로 보존될 수 있다. 두 번째는 자식 엘리먼트가 한 번 이상 나타날 수 있는지 없는지에 대한 Constraints로써, 이는 자식 엘리먼트가 기껏해야 한 번 나타난다라는 의미를 보존하기 위한

것으로 Equality-Gnerating Dependencies(EGD)로 알려져 있다. [표 4.1]에서 A와 B가 이에 해당하고, 관계형 스키마에서 SQL UNIQUE 절을 사용해서 보존될 수 있다. 마지막으로는 자식 엘리먼트가 한 번은 나타나야만 하는 지에 대한 Constraints로써, [표 4.1]에서 B와 D가 이에 해당하고, Tuple-Generating Dependencies(TGD)로써 알려져 있다. TGD를 관계형 스키마로 보존하는 방법은 자식 엘리먼트가 같은 테이블로 매핑될 경우에는 NOT NULL, Constraints로 보존되고, 다른 테이블로 매핑될 경우에는 외래키 제약조건으로 보존된다.

③ Inclusion Dependencies (IDs) - IDs는 XML 문서 내의 특정 속성의 값이 다른 속성의 값으로 나타나야만 한다라는 의미이다. 관계형 데이터베이스 이론의 참조무결성의 일반화된 개념이라 할 수 있다. DTD에서 IDs를 추론할 수 있는 요소는 ID, IDREF, IDREFS 속성이 있다. [그림 4.8]과 같은 DTD를 고려해보자.

```
<!ELEMENT person  (name, (email | phone)?)>
<!ATTLIST person  id       ID      #REQUIRED>
<!ELEMENT contact          EMPTY>
<!ATTLIST contact aid      IDREF   #REQUIRED>
<!ELEMENT editor  (person*)>
<!ATTLIST editor  eids     IDREFS  #IMPLIED>
```

[그림 4.8] person DTD

[그림 4.8]에서 볼 수 있듯이, person 엘리먼트는 속성의 값이 XML 문서 내에서 유일한 값을 갖는다는 의미의 ID로 정의된 id 속성을 갖는다. 그러므로, 이 속성은 person 엘리먼트가 관계형 스키마로 매핑됐을 때, 기본키로 보존이 될 수 있다, 그러나, [그림 4.8]의 DTD에서 더 중요한 것은 ID로 정의된 id 속성과 IDREF로 정의된 aid, IDREFS로 정

의된 eids의 관계라고 볼 수 있다. IDREF와 IDREFS로 정의된 속성은 XML 문서 내에서 ID로 정의된 속성의 값을 참조한다. 두 정의의 차이는 IDREF가 ID 값을 하나만 가질 수 있는 반면, IDREFS는 여러 ID 값을 갖는다는 것이다. 이러한 의미 또한 관계형 스키마에 보존이 되어야 하는데, 이는 앞서 언급한대로 ID 값이 관계형 스키마의 기본키가 되고, 이를 참조하는 IDREF, IDREFS로 정의된 속성은 외래키 제약조건을 정의함으로써 보존을 할 수 있다.

4.4 정리

본 장에서는 XML 문서를 RDBMS로 저장하기 위해서 NCPI-MDS 기법을 제시했다. 본 장에서 제시한 NCPI-MDS 기법은 우선 Inlining 알고리즘을 적용하기 전에 DTD 간소화 절차를 수정함으로써, DTD에서 추론할 수 있는 많은 의미적인 부분들을 보존한다. 다음으로 NCPI-MDS 기법은 기존의 Inlining 기법들에서 야기되는 많은 중복 데이터의 발생, 비효율적인 릴레이션 생성과 관련하여 수정된 Inlining 알고리즘을 보여주고 있다. 마지막으로 NCPI-MDS 기법은 간소화 절차를 거친 DTD로부터 추론할 수 있는 의미적인 부분들이 무엇이 있으며, 이러한 요소들이 관계형 스키마의 어떤 요소로 매핑이 될 수 있는지를 보여주었다.

제 5장 비교 분석

본 장에서는 같은 DTD에 대해 NCPI-MDS 기법을 이용해서 생성되는 릴레이션과 기존의 Hybrid Inlining 기법으로 생성되는 릴레이션을 비교 분석

한다.

5.1 restaurants DTD 예제

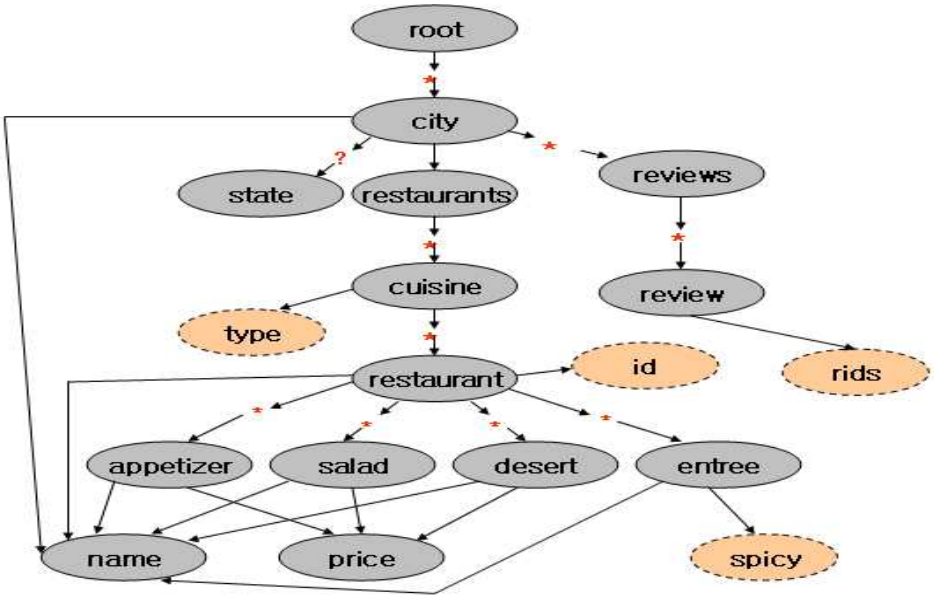
[그림 5.1]과 같은 DTD가 존재한다고 가정한다.

```
<!ELEMENT root          (city*)>
<!ELEMENT city          (state?, restaurants, reviews*, name)>
<!ELEMENT state         (#PCDATA)>
<!ELEMENT reviews       (review*)>
<!ELEMENT review        (#PCDATA)>
<!ATTLIST review        rids    IDREF   #IMPLIED>
<!ELEMENT restaurants   (cuisine*)>
<!ELEMENT cuisine       (restaurant*)>
<!ATTLIST cuisine type  (French | Chinese | American | Korean)
                                #REQUIRED>
<!ELEMENT restaurant    ((appetizer | salad | desert | entree)*, name)>
<!ATTLIST restaurant    id      ID      #REQUIRED>
<!ELEMENT appetizer     (name, price)>
<!ELEMENT salad         (name, price)>
<!ELEMENT desert        (name, price)>
<!ELEMENT entree        (name)>
<!ATTLIST entree        spicy   CDATA   #IMPLIED>
<!ELEMENT name          (#PCDATA)>
<!ELEMENT price         (#PCDATA)>
```

[그림 5.1] restaurants DTD 문서

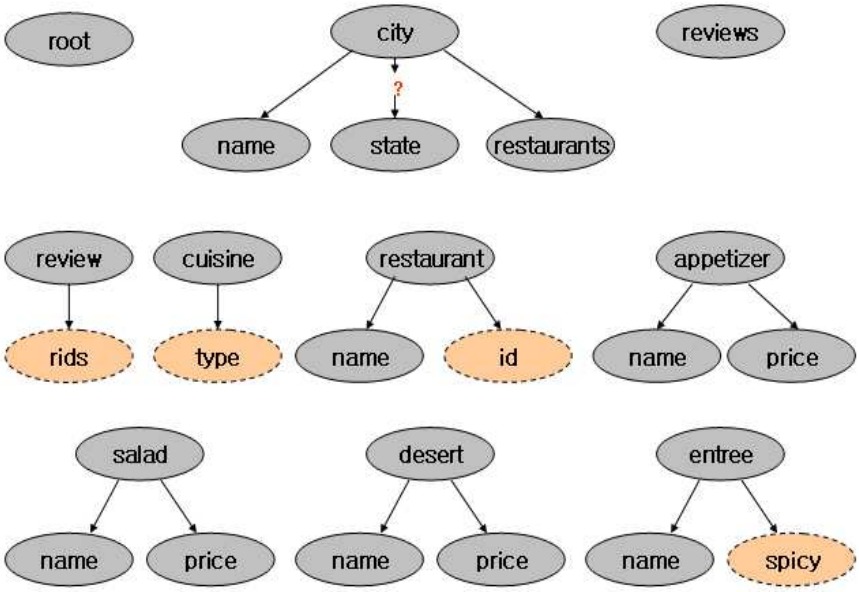
5.1.1 기존 Inlining 기법에 의한 릴레이션 생성

[그림 5.1]의 DTD를 따르는 XML 문서를 저장하기 위해 기존의 Hybrid Inlining에서 DTD 문서를 파싱하면 [그림 5.2]와 같은 DTD 그래프를 생성할 수 있다.

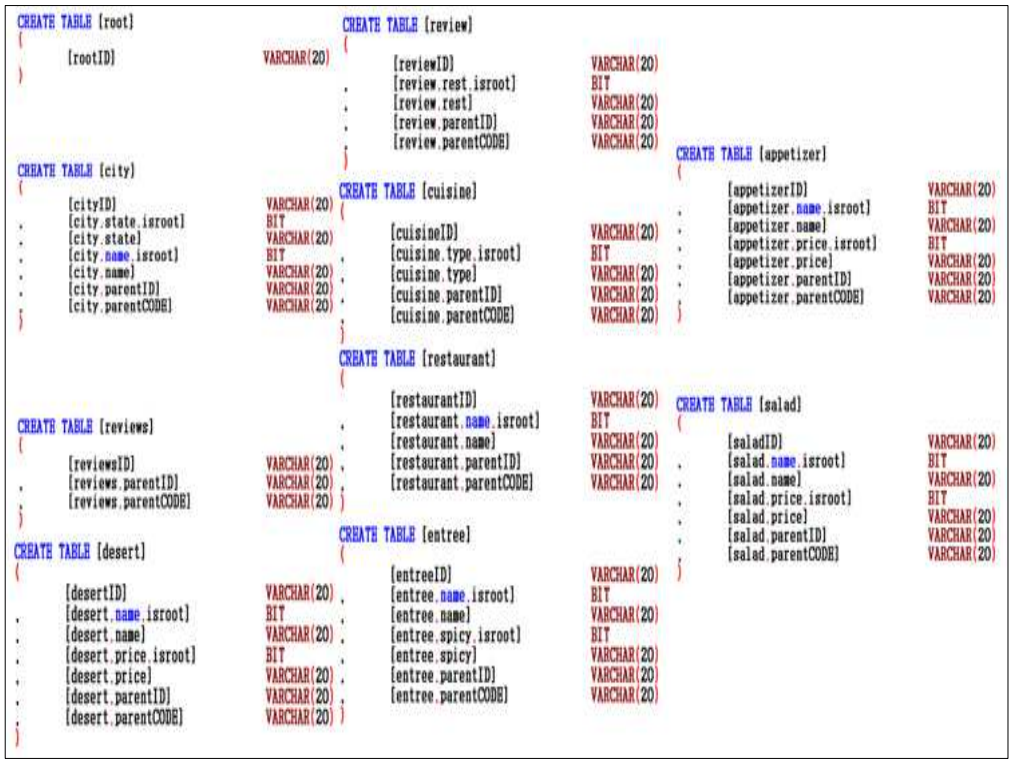


[그림 5.2] restaurants DTD 그래프

[그림 5.2]에서 점선 노드는 DTD에서 애트리뷰트를 의미하고, 노드와 노드 사이의 edge는 DTD에서의 “*”, “+”, “?” 연산자를 표시하고, 연산자가 나타나지 않는 edge는 DTD에서 연산자 없이 정의된 엘리먼트, 즉 반드시 한번은 나타나야만 하는 엘리먼트를 연결하는 edge이다. 다음으로, [그림 5.2]의 DTD 그래프를 기반으로 DFS 탐색을 통해서 생성 규칙에 의해 [그림 5.3]과 같은 엘리먼트 그래프를 생성해 낸다. 다음으로 생성된 엘리먼트 그래프를 기반으로 [그림 5.4]와 같은 릴레이션을 추론해 낼 수 있다.



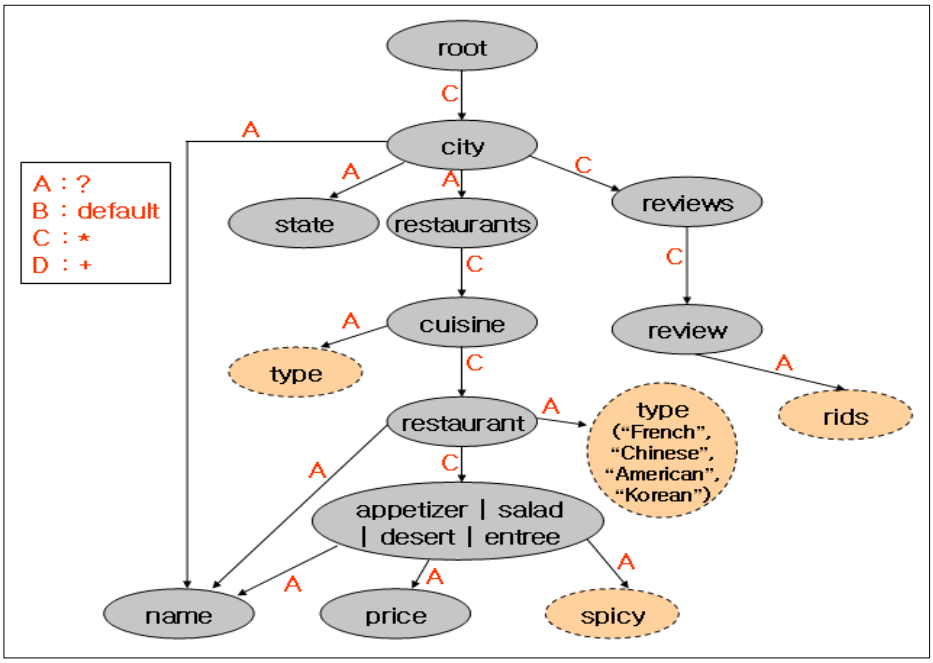
[그림 5.3] DTD 그래프부터 만들어지는 엘리먼트 그래프



[그림 5.4] Hybrid Inlining 기법에 의해 추론된 릴레이션

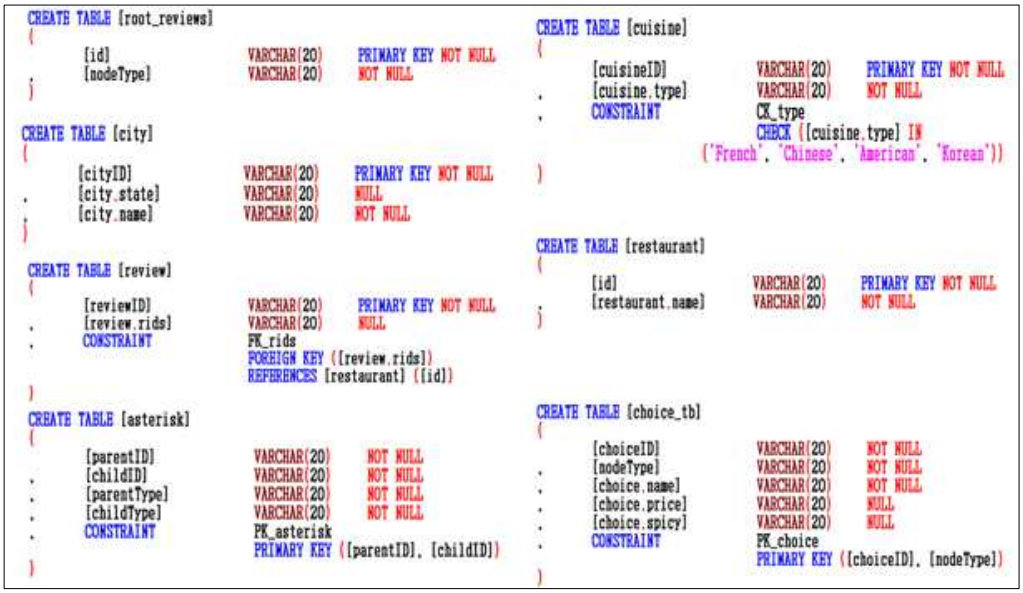
5.1.2 NCPI-MDS 기법에 의한 릴레이션 생성

본 논문에서 제시하는 NCPI-MDS 기법도 DTD를 파싱한 후 내부적으로 [그림 5.5]와 같은 DTD 그래프를 생성한다. 그러나, NCPI-MDS 기법에 의해 보존된 정보들을 DTD 그래프에 표현하기 위해 [그림 5.2]에서 본 DTD 그래프를 사용하지 않고 연구 [2]에서 보여주고 있는 Annotated DTD 그래프(ADG)를 사용하여 표현한다. 또한, 연구 [2]의 원래의 ADG와는 달리 "|" 연산자로 저의된 엘리먼트는 그래프에서 하나의 노드로 표현하고, 도메인 제약조건을 가지는 노드는 해당 노드에 한정된 값을 표현한다. 이는 실제 구현 시 내부적인 자료구조로 표현될 수 있다.



[그림 5.5] restaurants DTD의 Annotated DTD 그래프

[그림 5.5]의 DTD 그래프를 기반으로 NCPI-MDS 기법에 따라 릴레이션을 추론해 내면 [그림 5.6]과 같은 릴레이션을 만들 수 있다.



[그림 5.6] NCPI-MDS 기법으로 추론된 릴레이션

5.2 restaurants DTD의 정성적 비교 분석

5.2절에서 살펴 본대로 기존의 Hybrid Inlining 기법에 의해 추론되는 릴레이션과 본 논문에서 제시하는 NCPI-MDS 기법에 의해 추론되는 릴레이션은 많은 차이를 보이고 있다.

5.2.1 DTD 간소화 절차의 차이점

기존의 Hybrid Inlining 기법에서는 DTD 간소화 절차를 거치면서 [그림 5.2]에서 볼 수 있듯이 대부분의 DTD의 의미적인 정보를 손실하고, “*”, “?” 연산자의 의미만을 보존한다. “*”, “?” 연산자는 릴레이션이 새로 생성되어야 하는지에 대한 정보를 얻을 수 있는 연산자이므로 기존의 Hybrid Inlining 기법에서도 이 정보는 보존이 된다.

하지만, [그림 5.7]에서 볼 수 있듯이 본 논문에서 제시하는 NCPI-MDS 기법은 Cardinality, 도메인 제약조건, Choice 연산자 정보 등을 보존할 수 있다.

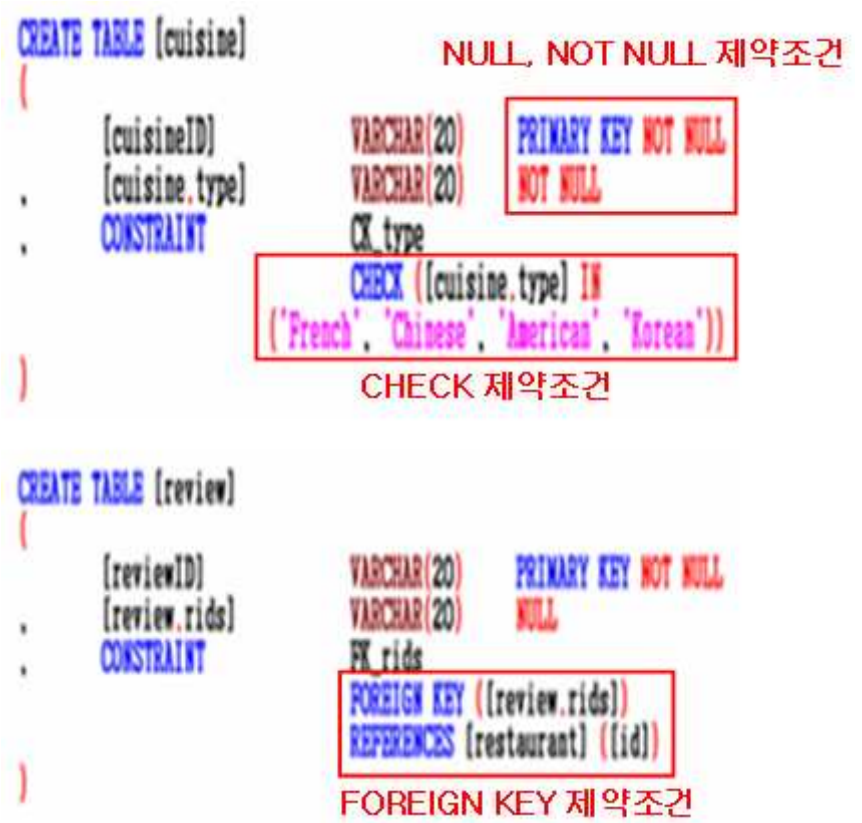
[그림 5.7] 보존되는 정보들

5.2.2 보존된 정보의 릴레이션으로의 매핑

기존의 Hybrid Inlining 기법의 문제점으로 언급했듯이 DTD 간소화 절차를 수정하여 의미적인 정보들을 보존했다 하더라도 이를 추론되는 릴레이션으로 매핑시킬 수 있는 방법이 필요하고, 본 논문에서는 4.3절에서 이를 언급했다. [그림 5.4]에서 볼 수 있듯이 기존의 Hybrid Inlining 기법에서는 의미적인 정보들을 보존을 할 수 없으므로 의미적인 정보들이 추론되는 릴레이션에 나타나지 않는다.

하지만 본 논문의 기법인 NCPI-MDS 기법은 수정된 DTD 간소화 절차를 통해 보존된 정보들을 [그림 5.8]과 같이 추론되는 릴레이션으로

매핑시킬수 있다.



[그림 5.8] 릴레이션으로 매핑되는 제약조건

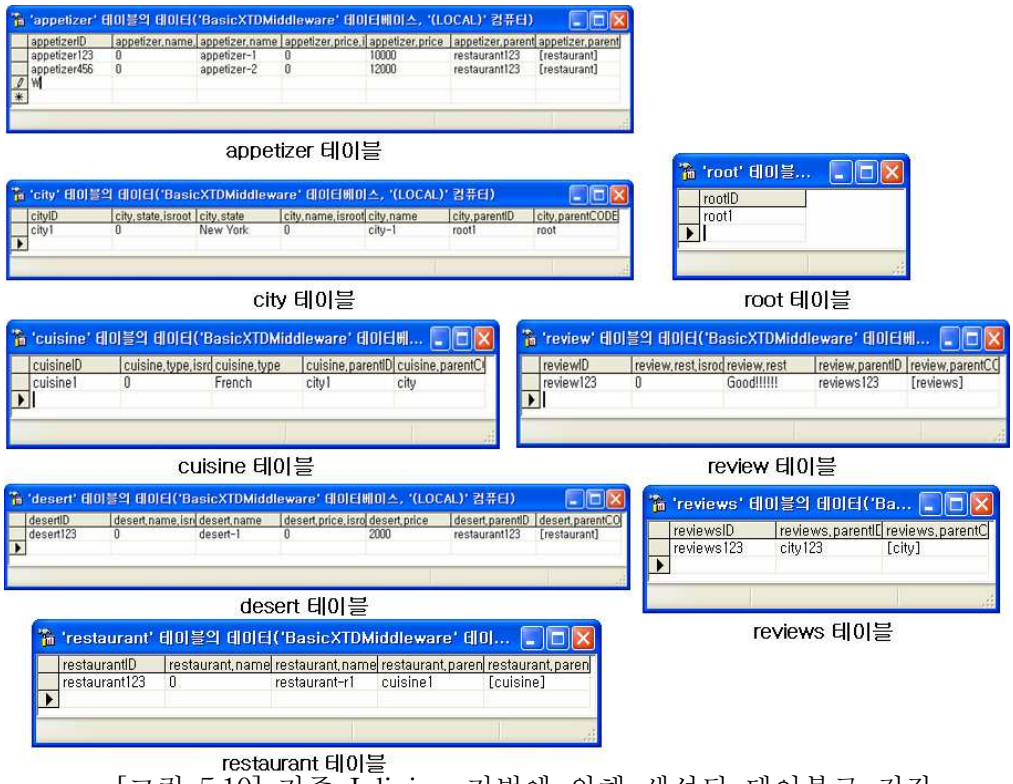
5.2.3 효율적 릴레이션 생성

기존에 제시된 Inlining 기법은 효율적인 릴레이션 생성에 관한 고려가 없기 때문에 많은 릴레이션이 생성이 되고, 이로 인해 질의 처리 시 조인 비용이 커지게 된다. [그림 5.9]와 같은 XML 문서를 생각해 보자. [그림 5.9]의 XML 문서는 [그림 5.1]의 DTD를 따른다.

```
<?xml version="1.0" ?>
<root>
    <city>
        <state>New York</state>
        <restaurants>
            <cuisine type="French">
                <restaurant id="r1">
                    <appetizer>
                        <name>appetizer-1</name>
                        <price>10000</price>
                    </appetizer>
                    <appetizer>
                        <name>appetizer-2</name>
                        <price>12000</price>
                    </appetizer>
                    <desert>
                        <name>desert-1</name>
                        <price>2000</price>
                    </desert>
                    <name>restaurant-r1</name>
                </restaurant>
            </cuisine>
        </restaurants>
        <reviews>
            <review rids="r1">
                <rest>Good!!!!!!</rest>
            </review>
        </reviews>
        <name>city-1</name>
    </city>
</root>
```

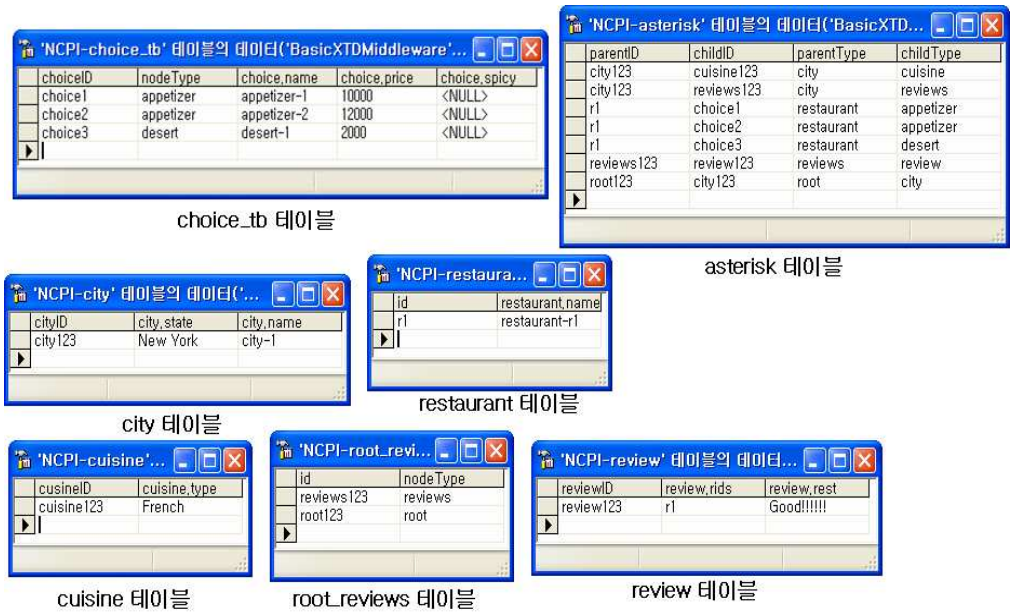
[그림 5.9] restaurants XML 문서

[그림 5.9]의 XML 문서가 기존의 기법에 의해 생성된 릴레이션의 실제 튜프로 삽입되었을 경우는 [그림 5.10]과 같다.



[그림 5.10] 기존 Inlining 기법에 의해 생성된 테이블로 저장

다음으로 본 논문에서 제시하는 NCPI-MDS 기법에 의해 생성되는 릴레이션에 [그림 5.9]의 XML 문서가 튜플로써 삽입된 경우는 [그림 5.11]과 같다.



[그림 5.11] NCPI-MDS 기법에 의해 생성된 테이블로 저장

[그림 5.9]의 XML 문서에 대한 semi-structured 질의언어 중에 XQuery를 통한 질의 중 [그림 5.12]와 같은 질의는 “레스토랑 이름이 restaurant-r1인 레스토랑의 모든 요리의 이름과 가격, spicy는?”라는 결과를 반환하는 XQuery이다.

```
<restaurant>
        {for $p in document("rrxs.xml")//restaurant
        where $p/name[text() = "restaurant-r1"]
        return
                <appetizer>
                {
                        for $b in $p/appetizer
                        return
                                <name>
                                {$b/name/text()}
                                </name>,
                                <price>
                                {$b/price/text()}
                                </price>
                }
                </appetizer>
                <desert>
                {
                        for $c in $p/desert
                        return
                                <name>
                                {$c/name/text()}
                                </name>,
                                <price>
                                {$c/price/text()}
                                </price>
                }
                </desert>
                <salad>
                {
                        for $c in $p/salad
                        return
                                <name>
                                {$c/name/text()}
                                </name>,
                                <price>
                                {$c/price/text()}
                                </price>
                }
                </salad>
                <entree>
                {
                        for $c in $p/entree
                        return
                                <name>
                                {$c/name/text()}
                                </name>,
                                <spicy>
                                {$c/@spicy}
                                </spicy>
                }
                </entree>

        }
</restaurant>
```

[그림 5.12] XQuery 질의어

RDBMS에 저장된 XML 문서의 데이터에 대해 [그림 5.12]의 질의어가 이루어지려면 적절하게 SQL 문장으로 변환이 이루어 져야 한다. 변환 과정은 본 논문의 범위를 넘어가므로 더 이상 언급하지 않는다. 그러나, 각각의 기법에 맞게 변환이 이루어진 SQL 문장을 예상해 보면 [그림 5.13], [그림 5.14]와 같다.

```
SELECT [appetizer.name], [appetizer.price], null FROM [appetizer]
        WHERE [appetizer.parentID] = (SELECT [restaurantID] FROM [restaurant]
                                      WHERE [restaurant.name] = 'restaurant-rl')

UNION

SELECT [desert.name], [desert.price], null FROM [desert]
        WHERE [desert.parentID] = (SELECT [restaurantID] FROM [restaurant]
                                      WHERE [restaurant.name] = 'restaurant-rl')

UNION

SELECT [salad.name], [salad.price],null FROM [salad]
        WHERE [salad.parentID] = (SELECT [restaurantID] FROM [restaurant]
                                      WHERE [restaurant.name] = 'restaurant-rl')

UNION

SELECT [entree.name], null, [entree.spicy] FROM [entree]
        WHERE [entree.parentID] = (SELECT [restaurantID] FROM [restaurant]
                                      WHERE [restaurant.name] = 'restaurant-rl')
GO
```

[그림 5.13] 기존 Inlining 기법에 맞는 SQL 질의어

```
SELECT [choice.name], [choice.price], [choice.spicy]
        FROM [NCPI-choice_tb]
        WHERE [choiceID] IN
                (SELECT [childID] FROM [NCPI-asterisk]
                        WHERE [parentID] = (SELECT [id] FROM [NCPI-restaurant]
                                WHERE [restaurant.name]='restaurant-rl'))
GO
```

[그림 5.14] NCPI-MDS 기법에 맞는 SQL 질의어

[그림 5.13]과 [그림 5.14]의 질의어에서 볼 수 있듯이 기존의 기법에서는 조인 4번의 결과의 UNION 연산이 들어가므로 많은 비용을 발생시킨다. 그러나 NCPI-MDS 기법에서는 단지 세 번의 조인 연산으로 같은 결과를 반환할 수 있다.

또한, 기존 기법의 문제점으로 지적한 많은 중복 데이터의 발생도 NCPI-MDS 기법에서는 [asterisk]라는 릴레이션을 통해 문제를 해결하고 있다. 예를 들어, 한 곳의 레스토랑이 여러 요리법을 가지고 있는 정보를 XML 문서를 표현하고, 이를 추론된 릴레이션으로 삽입할 경우 기존의

기법에서는 요리법의 개수에 따라 레스토랑 정보가 [restaurant] 릴레이션에 중복적으로 저장이 된다. 반면 NCPI-MDS 기법에서는 [asterisk] 릴레이션을 이용하여 부모-자식 관계 정보(키 정보)를 저장함으로써 데이터가 중복 저장되는 것을 방지할 수 있다.

5.3 정리

본 논문에서 제시하는 NCPI-MDS 기법이 기존의 기법보다 더 많은 정보를 릴레이션으로 보존하고, 효율적으로 릴레이션을 생성한다. 이를 정리하면 [표 5.1]과 같다.

종류	예시
도메인 제약조건	cuisine 엘리먼트의 type 속성이 도메인 제약조건을 가짐. 이는 [cuisine] 릴레이션에서 볼 수 있듯이 SQL CHECK 절로 보존이 됨.
대응수 제약조건	DTD 그래프에서 대응수의 타입에 따라 릴레이션 생성 시 NULL, NOT NULL 제약조건으로 보존됨.
Inclusion Dependencies	review 엘리먼트의 rids 속성이 Inclusion Dependencies를 가짐. 이는 [review] 릴레이션에서 기본키, 외래키 제약조건으로 보존됨.
효율적인 릴레이션 생성	[choice_tb] 릴레이션이 appetizer, salad, desert, entree 엘리먼트의 모든 데이터를 저장함. 기존의 Hybrid Inlining 기법이 많은 릴레이션을 생성하던 것과는 달리 하나의 릴레이션에 데이터를 저장함으로써 탐색 비용을 줄일 수 있음.
중복 데이터의 제거	[asterisk] 릴레이션을 통해 중복 데이터를 제거할 수 있고, 상향●하향 탐색에 도움을 줄 수 있음.

[표 5.1] NCPI-MDS 기법의 결과

제 6장 결론

XML 문서를 RDBMS에 저장하기 위한 기법 중 Hybrid Inlining 기법은 DTD로부터 보존이 될 수 있는 많은 의미적인 정보들을 손실할 뿐만 아니라, 실제 XML 문서를 저장할 때, 많은 데이터의 중복이 발생하는 문제점을 보인다.

이에 본 논문에서는 DTD로부터 추론할 수 있는 의미적인 정보들을 보존하기 위해 수정된 DTD 간소화 절차를 제시하고, 이를 관계형 릴레이션으로 매핑할 수 있는 방법을 제시하였으며, 데이터의 중복을 방지하고, 데이터의 질의 시 탐색에 도움이 될 수 있는 효율적 릴레이션 생성을 다룬 NCPI-MDS 기법을 제시하였다.

향후 연구 방향은 본 논문에서 제시한 NCPI-MDS 기법에 의해 생성된 릴레이션에 대해 질의응답시간이나 조인수와 같은 다양한 측정기준을 가지고 성능평가가 이루어 져야 할 것이다. 또한, 릴레이션을 추론하기 위해 DTD를 사용하는 것이 아니라, DTD의 문제점을 극복하기 위해 제시된 XML Schema를 사용하는 릴레이션 추론기법도 연구되어야 한다.

참고문헌

[1] Bourret, R. "XML and Database", Internet Document, December, 2004, http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[2] D. Lee, and W. W. Chu, "Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema", International Conf. on Conceptual Modeling/the Entity Relationship Approach, 2000.

[3] D. Chamberlin, P. Fankhauser, M. Marchiori, J. Robie, "XML Query (XQuery) Requirements", W3C Working Draft, http://www.w3.org/TR/xquery-requirements/, 2003.

[4] D. Florescu, D. Kossmann, "Storing and Querying XML Data using an RDBMS", Bulletin of the IEEE Computer Society Technocal Committee on Data Engineering, 22(3):27-34, 1999.

[5] H. V. Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks V. S. Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, Nuwee, "TIMBER : A Native XML Database", In Proc. of VLDB Conf., 2003.

[6] H. Schoning, "Tamino - a DBMS Designed for XML", In Proc. of IEEE ICDE, 2001.

[7] J. Shanmugasundaram et al. "Relational databases for querying XML documents : Limitations and opportunities", In Proc. Of VLDB, Edinburgh, Scotland, 1999.

[8] J. Clark, S. DeRose, "XML Path Language (XPath) Version 1.0",

W3C Recommendation, http://www.w3.org/TR/xquery/, 2004.

[9] Kurt Cagle, Dave Gibbons, David Hunter, Nikola Ozu, Jon Pinnock, Paul Spencer, "Beginning XML", 2-7, 2000.

[10] P. J. Marron, G. Lausen, "On Processing XML in LDAP", In Proc. of VLDB Conf., 2001.

[11] S. Lu, Y. Sun, M. Atay, and F. Fotouhi, "A New Inlining Algorithm for Mapping XML DTDs to Relational Schemas", In Proc. of the 1st International Workshop on XML Schema and Data Management, 2003

[12] S. A. T, Lahiri, J. Widom, "Ozone : Integrating Structured and Semistructured Data", In Proc. of DBPL Conf., 1999.

[13] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, J. Simeon, "XQuery 1.0 : An XML Query Language", W3C Working Draft, http://www.w3.org/TR/xquery/, 2005.

[14] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergean, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommecdation, http://www.w3.org/TR/2004/REC-xml-20040204/, 2004.

[15] Y. Chen, S. Davidson, and Y. Zheng, "Constraint Preserving XML Storage in Relations", In WebDB, 2002.

[16] Y. Chen, S. Davidson, C. Hara, and Y. Zheng, "RRXS : Redundancy reducing XML storage in relations", In Proc. of 29th International Conf. on VLDB, 2003.

ABSTRACT

NCPI-MDS:New Constraints-Preserving Inlining method with Modified DTD Simplification

An, Sung-Chul
Major in Computer Engineering
Dept. of Computer Engineering
Graduate School of
Hansung University

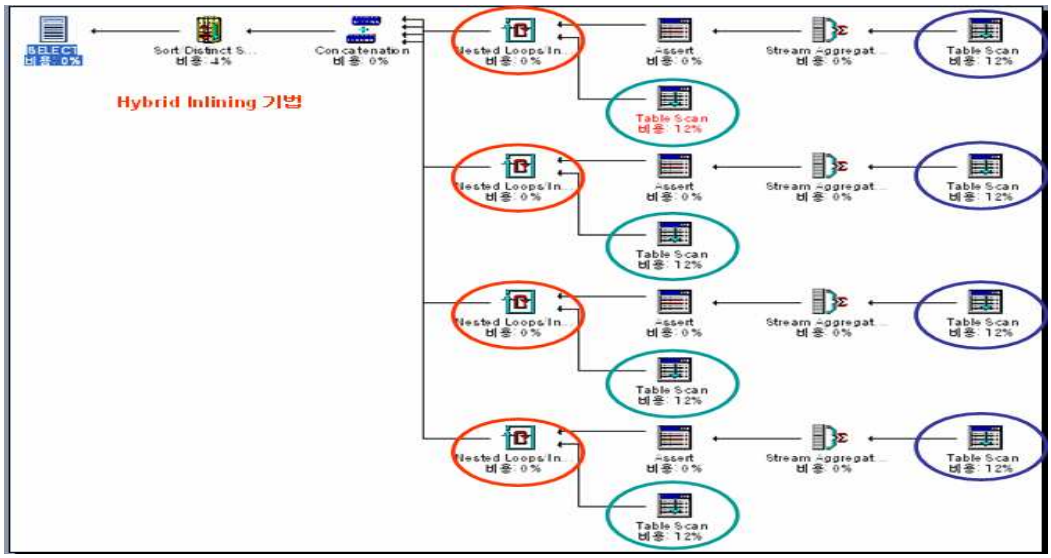
XML is a standard language to express and exchange the data over the web. Recently, researches about techniques that storing XML documents into RDBMS and managing it have been progressed. These researches use the techniques that are receiving the DTD document as an input and generating the relational schema from it. Existing researches, however, do not consider the semantic preservation because they only focus on the simplification of the DTD. Further, because existing studies only focus on the preservation techniques to store XML data such as content and structure. there is a troublesomeness that have to use the stored-procedure or trigger for the data integrity.

This paper process a improved Inlining technique to create effective relations and to preserve semantics which can be inferred from DTD.

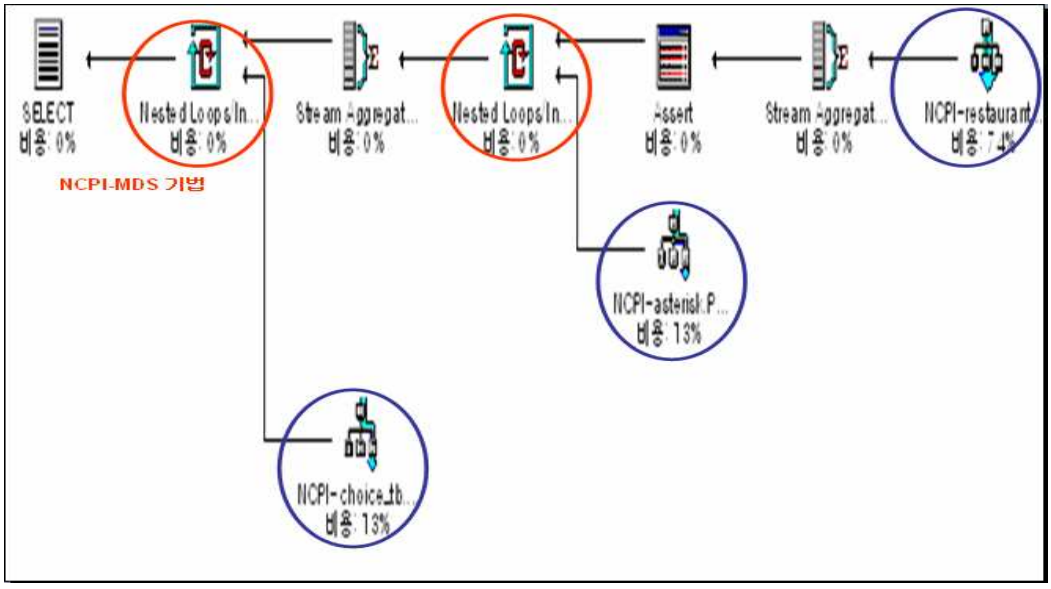
별첨

MS-SQL 2000 SERVER를 이용한 질의처리 성능

<실행 계획 - 기존 기법>



<실행 계획 - NCPI-MDS 기법>

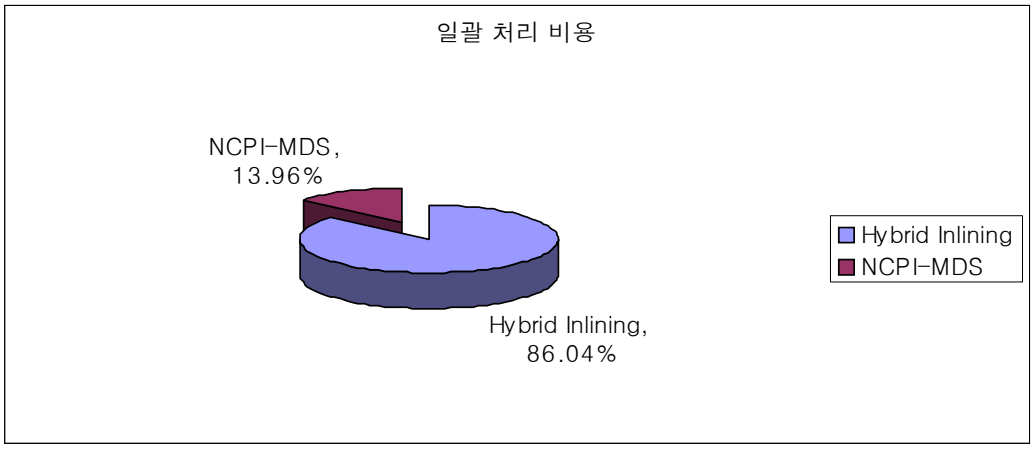


별첨

<실행 계획 비교>

	JOIN 수	Table Scan 수	Index Scan 수	Index Seek 수
기존 기법	4	8	0	0
NCPI-MDS	2	0	1	2

<질의어 일괄 처리 비용>



<질의 처리를 위한 CPU 시간 및 실행 시간>