자동 생성 폼과 SQL을 이용한 ERD 표현

ERD Representation using Auto-Generated Form and SQL

나영국
서울시립대학교 전자전기컴퓨터학부

Young-Gook Ra(ygra123@gmail.com)

요약

통상적으로 데이터베이스 어플리케이션을 구현하는 과정은 ERD(Entity Relationship Diagram)와 프로세스 모델을 산출하는 분석 과정을 거쳐 코딩, 테스팅 (testing)으로 프로젝트를 완성한다. 이 과정에서 가장 정형화되지 못한 부분이 분석과정으로 (1) 고객이 자신이 원하는 시스템의 세부 사항까지 알지 못한다; (2) 고객의 비즈니스 (business) 로직을 개발자가 이해하기 어렵다; (3) 분석 산출물인 ERD와 프로세스 모델을 고객이 이해하기 어렵다 등의 이유 때문이다. 본 논문은 분석 과정을 효과적으로 수행하기 위하여 가장 중요한 분석 산출물인 ERD (Entity Relationship Diagram)를 고객이 이해하기 쉬운 폼 형태로 제시할 것을 제안한다. 이 폼들은 데이터를 입력할 수 있게 하여 고객이 입체적으로 모델을 평가할 수 있어야 하며 분석 과정에서 발견되는 비즈니스 로직을 즉각 구현하여 고객이 이 폼들을 실행하면서 구현되는 로직을 평가, 이해하여 비즈니스 로직을 세부적으로 제공할 수 있게 하여야 한다. 이 목적을 위하여 고객이 제공하는 비즈니스 로직을 폼과 폼끼리의 참조를 포함하는 데이터 플로우(data flow) 형태로 우리의 폼 시스템에서 구현할 수 있어야 한다. 고객은 추상적인 ERD 대신에 데이터를 포함하고, 데이터 플로우 로직이 구현되어 있는 폼들을 실행 시켜 보면서 자신이 제공한 비즈니스 로직을 검토할 수 있으며 새로운 로직을 발견할 수 있다. 이러한 과정을 반복적으로 수행하면서 고객과 개발자는 충분히 세부적이고 모순이 없는 분석 과정을 성공적으로 수행할 수 있게 된다.

■ 중심어 : | 데이터 모델 | 관계형 모델 | SQL | 프로토타입 | 어플리케이션 프레임워크 |

Abstract

Generally, the development of the database application includes the requirement analysis phase of creating ERD (Entity Relationship Diagram) and process models, coding, and testing. From the above phases, the analysis phase is not most formalized. It is usually hard task because (1) customers don't know the details of the desired system; (2) developers can't with ease understand the business logic of the customers; (3) the outcomes of the analysis, which are ERD and process models, are not easy to understand to the customers. This paper propose that the executional forms, which are better to understand the systems, should be presented to the customers instead of the ERD. These forms should accept the data input so that customers can review the various aspects of the outcome models. The developers should be able to instantly implement the business logic and also should be able to visually demonstrate the logic in order to get the details of it. For this goal, the customer supplied business logic should be able to be implemented by the references between forms, actions, constraints from the perspective of the data flow. The customers try to execute the forms implementing the business logic and review their supplied logic find new necessary business logic of their own. Iterating these processes for the requirement analysis would result in the success of the analysis which is sufficiently detailed without conflicts.

■ keyword : | Data Model | Relational Model | SQL | Prototype | Application Framework |

접수번호 : #090216-003
접수일자 : 2009년 02월 16일
심사완료일 : 2009년 04월 22일
교신저자 : 나영국, e-mail : ygra123@gmail.com

I. 서 론

데이터베이스 어플리케이션을 구현할 때 가장 어려운 부분이 고객의 요구 사항을 파악하고 이를 모델링하는 일이다. 이를 위하여 개발자는 통상적으로 고객의 요구 사항을 ERD (Entity Relationship Diagram) 로 표현하여 고객에 보여주고 확인을 바라고 추가 요구 사항을 제시하여 주기를 바란다. 하지만 문제는 비록 ERD의 개념이 쉽고 간단하다 할지라도 복잡한 업무 지식을 표현할 때 고객은 그 암묵적인 (implicit) 의미를 파악하기 어렵고 고객에 따라 ERD를 전혀 이해하지 못하는 수가 있다. 그러면 고객과 개발자 사이에 대화의 매체로써의 ERD 역할을 하지 못하고 고객은 요구사항을 더 발전시키지 못하고 개발자는 고객의 비즈니스 (business) 로직 (logic)을 이해할 수 없는 교착 상태에 빠질 수 도 있다. 더욱이 대부분의 SI (System Integration)의 경우 고객은 대략적인 아이디어를 가지고 있을 뿐이고 세부적인 사항은 고려해 보지 않는 게 대부분이다. 개발자는 이러한 고객을 위하여 세부 사항을 제시해야 하는 부담이 있다.

이 문제를 극복하기 위하여 전 연구 [1]에서 ERD를 간단한 폼으로 비쥬얼하게 보여 주는 방안을 제안하였다. 더욱이 [1] 연구는 이를 폼으로 보여 줄 뿐만 아니라 폼과 폼 사이의 참조 로직을 구현하는 여러 개의 생성자 (constructor)를 제시하였으며 덧붙여 이를 SQL로 필터링하는 시스템을 구현하는 프로토타입 (prototype) 프레임워크 (framework)를 구현하였다. 이 프레임워크는 ERD에서 나타나는 대부분의 관계 (relationship)를 구현할 수 있으나 참조되는 속성이 관계에 직접 연결된 엔터티 (entity)가 아닌 경우를 구현하지 못하는 단점이 있으며 여러 개의 생성자는 개발자에게 이 툴 (tool)을 사용하기 어렵게 한다.

본 논문은 이전 연구 [1]의 단점을 개선하여 ERD의 모든 관계를 단일한 메커니즘 (mechanism)으로 구현한다. 이는 SQL (Structured Query language)과 매핑 (mapping)으로 관계를 표현함으로써 달성할 수 있었다. 이 논문이 제시하는 프레임워크는 선언적인 (declarative)인 데이터 정의 및 조작 언어인 SQL을 주로 사용한다. 프레임워크에서 (1) ERD의 관계 창 인터페이스의 데이터를 SQL로 정의하고 제약 조건을 SQL로 구현한다; (2) 실행 폼에서 한 필드 (A) 의 데이터 값이 다른 폼 필드 (B) 데이터 값을 참조할 경우 A 값은 SQL에서 의해서 정의되고 구현된다; (3) 계산 필드의 값이 SQL로 정의되고 구현된다.

ERD를 데이터베이스 어플리케이션으로 매핑 (Mapping)하는 프레임워크는 회계, MRP 등 업무 로직이 복잡한 영역을 제외하고는 주문, 제조, 영업, 재고 등의 기업의 핵심 프로세스를 구현할 수 있다. 그리고 업무 로직이 복잡하거나 사용자 편의성 때문에 인터페이스가 복잡한 경우는 매핑 프레임워크를 이용하지 않고 일반적인 프로그램밍 언어와 개발 도구를 이용하여 별도로 이를 개발하여 프레임워크에서 생성한 프로그램과 결합하여 최종 완성본을 생성할 수도 있다. 즉, 이 프레임워크는 확장성을 가진다.

이 논문 프레임워크 구현 기술의 가장 특이한 기술적인 부분은 조회 리라이트 (query rewrite)이다. 이 기술은 개발자가 입력한 조회를 수정하여 관계 창에 데이터를 제공하는 기술이다. 우리의 프레임워크에서 개발자는 각 관계에 대하여 SQL을 제공한다. 이 SQL에서 -2,000,000,000 은 관계에 참여하는 대상 개체의 ID를 의미하는 데 대상 개체의 창이 뜰 때에는 관계에 참여하는 대상 개체가 정해져 있지 않으므로 -2,000,000,000을 포함하는 프레디켓 (predicate)을 제거하여 실행하여 대상 개체 인스턴스의 집합을 대상 창에 제공한다. -1,000,000,000은 관계에 참여하는 주도 개체의 ID로써 주도 개체의 필드를 포함하는 제약 조건을 구현할 때 유효하다. 이 때 -1,000,000,000은 주도 개체의 ID로 치환된다.

II. 관련연구

비즈니스 데이터베이스 어플리케이션에서 데이터 모델과 사용자 인터페이스의 유사점은 많은 연구자들의 주목을 받았다. 이 유사점을 기반으로 하여 데이터 모델로부터 사용자 인터페이스를 자동 생성하는 프로토

타입 시스템 생성기법이 제안되어 왔다. 이 연구는 GENIUS[2], JANUS[3], TRIDENT[4], GUIP[5] 등이 있다.

GENIUS[2]는 데이터 모델로부터 UI (User Interface)의 배치는 뷰 (view)를 정의함으로써 이루어지고 UI의 행동 (behavior)은 페트리넷 (PetriNet)을 기반으로 하여 스펙(spec)을 개발자가 선언적 (declarative)으로 정의할 수 있게 하였다. 이 연구의 가장 큰 특징은 UI와 데이터 모델 사이의 갭 (Gap)을 뷰를 사용하여 해결하려는 접근 방식이다. 예를 들어 고객 폼은 고객 엔터티뿐만 아니라 주문 엔터티를 포함할 수 있다. 이런 경우 엔터티 하나에 폼을 하나 정의하는 단순한 방법으로는 사용자가 원하는 폼을 생성할 수 없다. 이 갭을 사용자가 원하는 폼을 뷰로써 정의한다. 더구나 UI 행동을 개발자가 대화 넷 (dialogue net) 스펙을 입력함으로써 이루어진다. 이 연구의 문제점은 정적인 UI를 위한 뷰가 갱신 가능 (updatable)이어야 하는데 일반적으로 뷰는 갱신 가능하지 않다 [2]. 더욱이 UI의 동적인 부분을 대화 넷으로 완전히 구현할 수 있는지에 대한 논거가 없다. 이 시스템은 구현되지 않았다.

JANUS [3]는 객체 모델로부터 사용자 인터페이스를 추출하는 접근 방식을 택했다. 이 연구는 데이터베이스 어플리케이션뿐만 아니라 일반적인 소프트웨어의 UI를 추출하여 프로토타입을 생성하는 기법을 담고 있다 이 연구 역시 구현되지 않았으며 UI 행동을 다루지 않았다. TRIDENT [4]는 태스크 (task) 분석으로부터 프로토타입까지 일련의 과정을 다루고 있다. 이 시스템은 요구 사항 분석을 계층적으로 수행하여 하나의 태스크 안에 활동 (activity) 체인 (chain)을 이루어 낸다. 프로토타입을 생성하기 전에 인터액션 (interaction) 스타일, 프리젠테이션 정의 등이 구체적인 개체로써 입력된다. 역시 구현되지 않았으며 활동 체인이 수행하는 태스크에 비해 복잡하여 실제로 프로그래머의 생산성을 높일 수 있을지는 의문이다.

GUPIS [5]는 요구사항 분석에 초점을 맞추어 시나리오 (scenario)로부터 어플리케이션의 행동을 정의하여 실행 가능한 프로토타입을 생성이 목표이다. 좀 더 자세하게 설명하면, UML과 UI 정보로부터 시나리오를 추출하고 이는 프로토타입이 생성되어 좀 더 세련화된 기능은 UI 환경에서 정의된다. 이 연구는 알고리즘을 구현하여 여러 예에 대하여 시험됐다. 이 연구는 요구사항 분석에 치중하는 것이 본 논문과 유사한 UI 프로토타입을 생성하는 기능을 구현하지 않았으며 작은 시스템에서도 시나리오가 매우 복잡하여 큰 시스템으로의 확장성 (scalability)이 증명되지 않고 있다.

최근의 연구로 Puerta [6]와 김정옥 [7]이 있다. Puerta는 도메인 모델을 이용하여 UI의 정적인 배치와 동적인 행동을 자동으로 생성한다. 이 연구에서 도메인 모델은 데이터 모델을 확장하는 고수준의 지식을 의미한다. 이 연구는 도메인 모델 편집기를 제공하여 개발자가 모델을 쉽게 정의할 수 있으며 UI의 동적인 행동은 대화 생성기로 이루어진다. 이 연구 역시 구현되지 않았으며 대화 생성기로 모든 UI의 행동을 정의할 수 있는지에 대한 언급이 없다. 김정옥 [7]은 요구 분석부터 사용자 인터페이스의 전이 객체를 모델링하고, 사용자 인터페이스를 생성하기 위한 비즈니스 이벤트의 추상화 모델링 규칙과 알고리즘을 만들었다. 이 연구는 UIP (User Interface Prototype)를 생성하기 위한 단계로 역할 모델링, 태스크 모델링, 추상화 모델링을 제안한다. 역할 모델링은UML의 유스케이스 (use case)로부터, 태스크 모델링은 상태 다이어그램으로부터, 추상화 모델링은 비즈니스 이벤트로부터 이루어진다. 이 연구는 기존의 연구에서 잘 다루지 않았던 UI 행동에 초점을 맞추었으나 구현이 되지 않았으며 (알고리즘은 제시) 따라서 검증 실험이 이루어지지 않았으며 복잡한 비즈니스 로직에 대하여 생산성 향상 효과가 제시되지 않고 있다.

STATEMATE [8]는 시스템에 대해 구조, 기능, 행동에 관하여 세 가지 뷰를 제공한다. 각 뷰에 대하여 모듈-차트, 활동-차트, 상태-차트를 구성할 수 있는 세 가지 언어가 있다. 이 언어로 시스템을 정의한 뒤 프로토타입 시스템을 구성하는 Ada 코드를 생성한다. 개발자는 이 코드를 시작점으로 하여 최종 시스템을 개발할 수 있다. 이와 별도로 우리의 실행 모드와 비슷한 개념으로 시뮬레이션을 하여 요구 사항을 점검할 수 있는 기능이 있다. 그러나 STATEMATE는 관계형 테이블

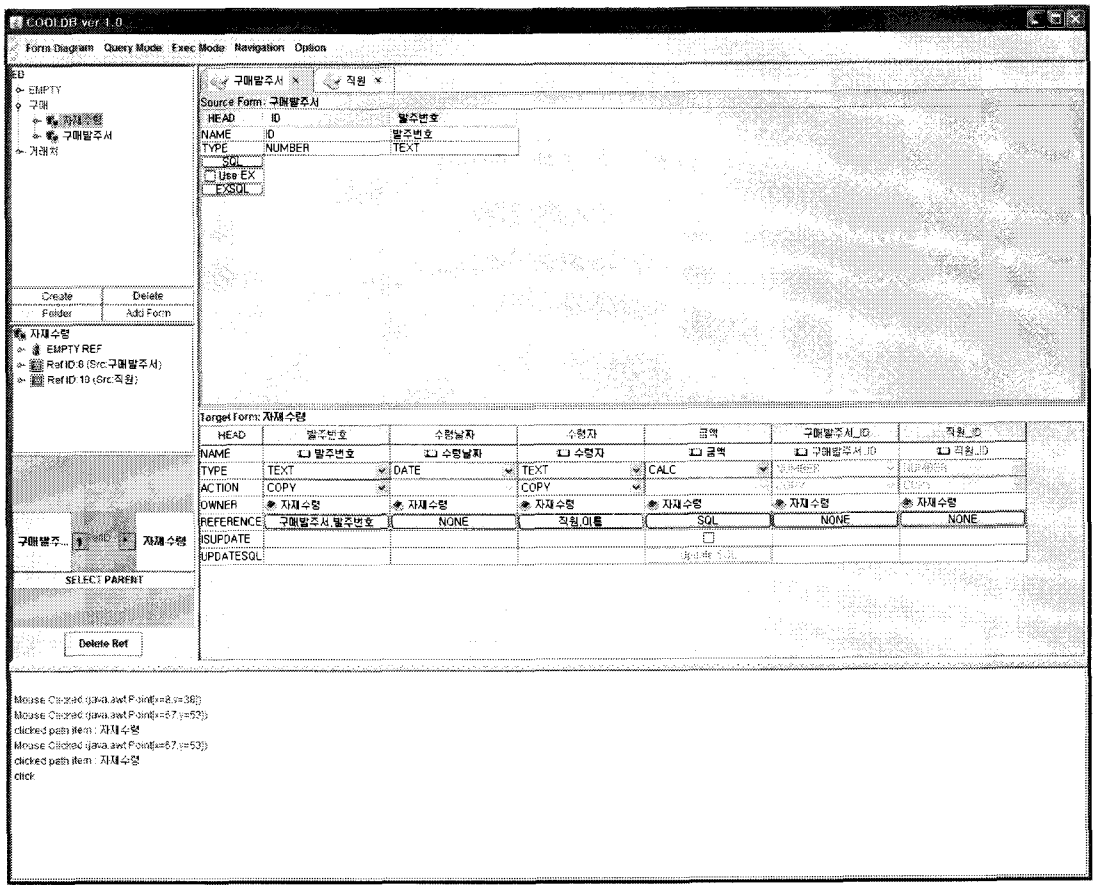

그림 1. ER 모델을 입력하는 편집 모드

을 반복적으로 정의하는 기능을 갖추지 못하였고 코드 생성은 코드와 개발자 코드와 혼합되어 관리가 쉽지 않은 단점을 가지고 있다.

SCR [9]은 요구 분석을 지원하는 도구로 미션 크리티컬 (mission critical) 프로그램의 요구사항을 테이블에 입력하게 하여 시뮬레이션을 가능하게 한다. 이 도구는 포말 메소드 (formal method)를 기반으로 안정적이지만 특정한 부문의 소프트웨어에 특화되어 있다.

이 시스템의 이전 버전 [1]은 ERD를 입력하여 각각의 ERD에 대하여 실행 가능한 폼을 생성한다. 이때 참조에 대하여 제한 조건으로 SQL을 사용자에 입력하게 하였다. 이 시스템은 실행 폼에서 사용자가 대상 개체를 생성하고, 이때 현재 폼으로 선택된 데이터를 복사하거나, 계산 필드를 도입하여 실제 최종 폼의 로직을 구현하는 기능이 있다. 하지만 관계로 직접 연결되어 있지 않은 개체의 데이터를 복사할 수 없는 단점이 있고 트리거 기능을 프레임워크에서 지원하지 않아 구현하기 힘들고 복잡한 트리거 전용 언어를 사용해야 하는 단점이 있다. 트리거 언어를 새로 배워야 하는 것은 오버헤드 (overhead)이다.

ER 모델의 완전성에 관한 연구 [10]에서는 ERD 수정 연산들에 대해 필요한 노력 (effort)을 측정하고 이를 바탕으로 모델의 완전성에 대한 지수를 개발한다.

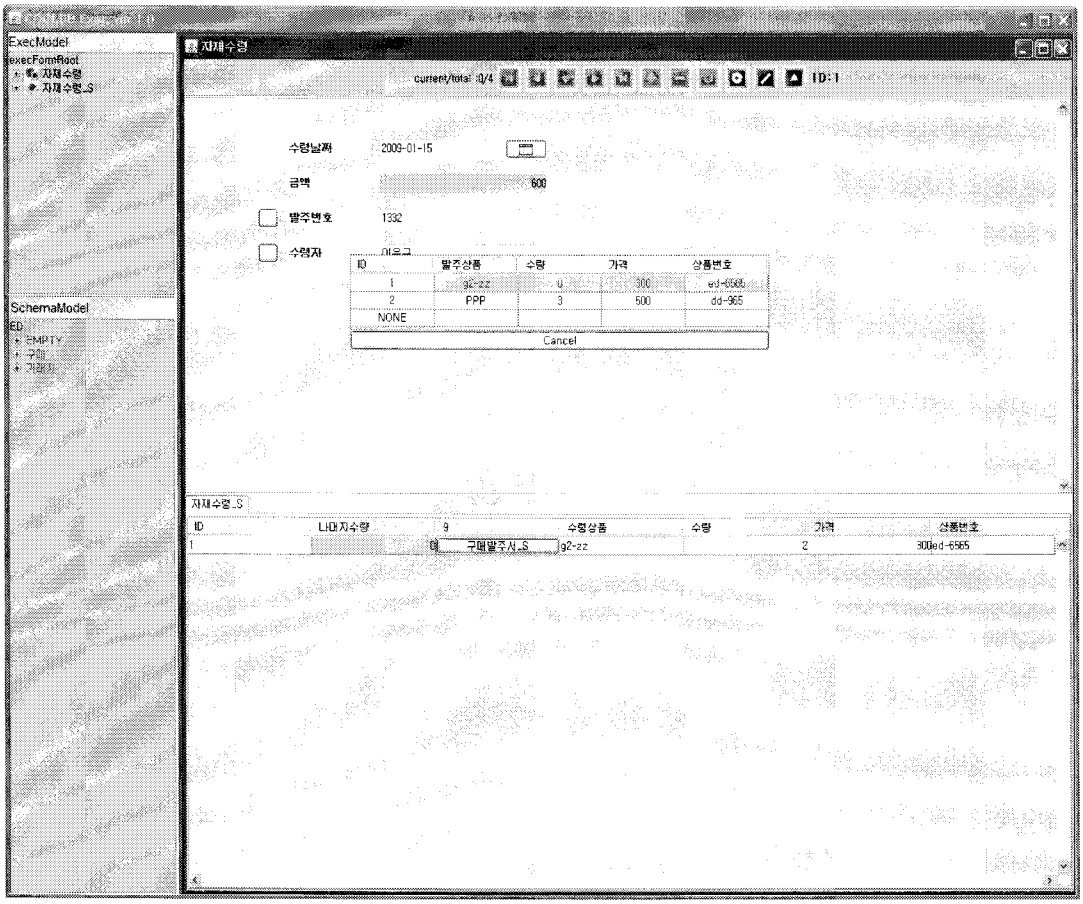

그림 2. 자동으로 폼을 생성한 실행 모드

본 시스템의 편집 모드는 ERD 수정을 쉽게 할 수 있도록 여러 제약 조건을 구현하고 있다. 특히 노력이 많이 들어가는 수정 연산을 집중 지원하고 있다.

ER 모델의 품질에 관한 연구 [11]는 모델의 수정 가능한 유연성의 주요 요소로 이해도 (understandibility)를 들고 있다. 본 시스템은 데이터 모델 수준에서 편집 가능하고 이에 테이블을 자동 생성하면서 데이터베이스 무결성 (integrity)을 확인한다는 면에서 수정이 용이하고 샘플 (sample) 데이터를 통하여 모델의 이해도를 높이므로 고품질의 모델 산출에 기여한다.

본 시스템과 같이 요구사항 분석 도구로 문자 데이터로부터 모델을 생성할 수 있게 하는 연구 [12]가 있다. 본 시스템이 ER 모델 산출을 지원하는 데 있는 반면 이 연구는 모델의 초기화에 중심을 두고 있다. 이 연구 시스템으로 모델을 초기화하면 본 시스템은 이를 확인하고 확장할 수 있다는 의미에서 상호 보완적이다. 모델 중심의 개발 방법론 MDSD (model driven software development)을 ER 모델에 적용한 연구[12]는 본 연구와 유사하다. 하지만 [12]는 개발 전 과정을 다루고 있는데 반해 본 연구는 모델의 검증 및 확장에 집중하고 있다. 또 다른 요구사항 발굴 시스템에 관한 연구 [13]는 GUI를 이용하여 사용자의 인터페이스 사용 (usage) 패턴을 발굴한다. 이는 패턴을 발굴하여 UML등의 모델에 반영하는 모델 주도 방법 (mode driven approach)의 하나이다. 이 연구 [13]는 사용자 요구사항을 발굴하려는 데 있어서 GUI를 사용하는 것은 본 논문과 비슷하나 GUI 자동 생성이 아니어서 GUI 생성에 많은 노력이 필요한 것이 단점이다.

III. 시스템의 개략적인 개념과 구현

제안 시스템은 자바 (java)로 코딩하였으며 데이터베이스는 오라클(oracle)을 사용하였다. 시스템은 대략적으로 편집 모드 [그림 1]와 실행 모드 [그림 2]로 나누어져 있다. 편집 모드는 데이터베이스 테이블을 생성하기에 편리하게 하였다. 또한 참조 (reference)를 입력할 수 입력할 수 있게 하여 관계 모델에서 비교적 어려운 외래키 (foreign key) 생성을 쉽게 하였다.

3.1 편집 모드

[그림 1]의 편집 모드는 테이블과 외래키 생성뿐만 아니라 참조에서 발생하는 매핑 (mapping)을 입력하는 도구로 유용하게 쓰인다. [그림 1]에서 오른쪽 위 페인 (pane)은 참조의 소스 (source) 테이블들을 탭으로 표현하고 오른 족 아래는 참조의 타겟 (target)을 의미한다. 하나의 타겟 테이블에 여러 개의 소스들이 존재할 수 있다. 각 소스는 오른 쪽 위의 탭에 보인다. 탭을 선택하는 것은 관계를 선택하는 것과 같다. 탭을 선택하면 소스 테이블 탭의 하나가 자동 결정되고 이에 매핑을 입력할 수 있다 [그림 1]. 매핑은 오른쪽 위의 그리드 (gird)에 나타난 필드와 오른 쪽 아래 그리드의 필드 사이에 정의된다. [그림 1]에서 오른쪽 위 그리드는 소스 테이블을 포함한 '관계 SQL'을 입력할 수 있게 하였다. 관계 SQL은 최종 사용자가 A 폼에 서 B 폼으로 관계를 형성할 때 A 폼의 개체를 선택할 수 있도록 하는 A 폼 개체의 집합이다. 예를 들어, [그림 2]에서 '자재 수령' 폼에서 '구매발주서' 폼을 선택할 때 발주 번호 버튼을 누르면 뜨는 창 (window)이 보인다. 이 창에서 행을 선택하면 두 폼 사이의 관계가 맺어진다. 이 때 이 창을 관계SQL로 정의한다.

관계 SQL 버튼을 입력하면 창 (window)이 뜨고 이 창은 생성된 테이블들과 그 관계를 보여주어 관계 SQL을 쉽게 입력할 수 있게 하였다. 관계 SQL은 반드시 소스 테이블의 ID를 포함하여 매핑의 필요한 모든 필드를 출력하여야 한다. 이때 필드들은 소스 테이블 필드 외로 다른 테이블의 필드를 포함할 수도 있고 스칼라 값을 반환하는 표현식 (expression)일 수도 있다. [그림 1]에서 오른 쪽 아래의 그리드의 각각의 행은 타겟 테이블의 필드와 그 계산 필드를 의미한다. 이 그리드에서 타입을 선택할 수도 있으며 아래 열에서 소스 필드를 선택하여 매핑을 생성한다. 매핑은 action을 선택할 수 있는데 그 값은 'copy'와 'link'가 있다. copy는 소스 필드의 값을 해당 타겟 필드 값으로 복사하는 것을 의미하며 link는 소스 필드의 값이 해당 타겟 필드 값을 대신하는 것을 의미한다. copy 매핑의 경우 하나의 타겟 필드가 여러 개의 소스 필드에서 값을 가져 올 수 있으며 이 때 가장 최신의 복사가 타겟 필드의 값으로 결정된다.

3.2 실행 모드

실행 모드는 [그림 2]에서 나타나 있다. 편집 모드에서 입력한 테이블, 필드, 관계 SQL, 매핑 ([그림 1]에서 지정한)을 이용하여 실행 모드는 사용자 폼을 동적으로 (dynamic) 생성한다. [그림 2]의 왼쪽 트리의 테이블 이름을 더블 클릭하면 폼이 동적으로 생성되어 사용자에 보인다. [그림 2]의 예에서 '자재수령' 폼이 선택되었고 사용자는 이 폼에 데이터를 입력하면서 본인이 작성한 데이터 모델을 입체적으로 검토할 수 있게 된다. 이 폼에서 '수령날짜'는 저장 필드이며 '금액'은 계산 필드 (4.3절에서 자세히 설명)이며 가운데 보이는 창은 '발주번호' 옆의 버튼을 누르면 나타난다. 이 창에서 하나의 행을 선택하면 '구매발주서' 폼과 관계가 형성된다.

본 제안 시스템은 관계 SQL과 매핑만으로 비즈니스 로직을 실행하여 데이터를 처리한다. 우리의 구현에서는 실행 모드가 로드 (load)될 때 메모리 내에 '폼-참조-필드' 트리 형식의 모델을 만들어 MVC 패턴 [14]을 이용하여 데이터 처리를 수행한다. 메모리내의 모델과 GUI의 컴포넌트 사이의 동기화 (synchronization) 문제는 모든 계산 필드와 link 참조 필드는 같은 폼에 있는 임의의 저장 필드 (copy 참조 필드 포함)가 갱신 될 때 리프레시 (refresh)하는 정책을 취했다. 이런 정책은 성능 (performance)을 심각하게 훼손하여 2-30개의 데이터 행을 가진 폼이 눈에 띄게 늦어지는 악 영향을 미친다. 하지만 우리 시스템의 목적이 최종 사용자 시스템이 아니라 요구 분석 단계에서 데이터 모델을 검토하고

고객과의 대화 (communication)를 용이하게 하여 새로운 요구사항을 도출하는 것이 목적이기 때문에 낮은 성능이 이슈 (issue)가 되지 못 한다.

3.3. 관계 구현

우리의 모델에서는 모든 개체는 하나의 테이블로 매핑되고 모든 관계는 외래키로 구현한다([표 1]). 시스템을 간단히 하기 위하여 모든 테이블의 기본키는 Id로 결정하였다. 카디널러티 (cardinality)가 1-1인 경우 tgFrgnFld는 tgForm의 외래키로써 이름은 A_ID_{REFID}로 정해지고 REFID는 참조의 번호로써 일련번호이다. 즉, A 폼의 ID 필드의 값과 같다. 1-N인 경우 tgFrgnFld는 tgForm의 외래키로써 이름은 A_ID_{REFID}로 정해서 A 폼의 ID 필드를 참조한다. N-1인 경우 srcFrgnFld는 srcForm의 외래키로써 이름은 B_ID_{REFID}로 정해서 B 폼의 ID 필드를 참조한다. M-N 인 경우 관계를 구현하는 폼은 A_B라 불린다. M-N인 경우 srcFrgnFld는 A_B 폼의 외래키로써 이름은 A_ID_{REFID}, 이고 tgFrgnFld는 A_B 폼의 외래키로써 이름은 B_ID_{REFID}이다.

표 1. 관계에서 자동으로 외래키 생성

REFID	cardinality	srcForm	tgForm	srcFrgnFld	tgFrgnFld
1	1-1	A	B		A_ID_1
2	1-N	A	B		A_ID_2
3	N-1	A	B	B_ID_3	
4	M-N	A	B	A_ID_4	B_ID_4

IV. 관계 구현 방법

이 프레임워크의 주목할 만한 기술적인 특징인 관계 구현은 (1) ERD의 관계 제약조건을 SQL로 표현 하고 (SQL 관계 제약 조건); 2. 한 개체의 값을 관계를 통하여 다른 개체로부터 얻는다 (매핑 정의). 또 다른 기술적인 특성으로 계산 필드의 값이 SQL로 정의되는 점을 들 수 있다(계산 필드 정의). 이러한 기술적인 특성을 예를 들어 자세히 설명한다.

4.1 SQL 관계 제약 조건

이는 예를 들어서 가장 잘 설명된다. 아래의 고객-주문 예에서 주문의 버튼 ([그림 4]에서 고객 라벨 옆의 검은색 네모) 을 누르면 고객 테이블의 데이터를 보여주는 것이 아니라 고객-주문 관계에서 미리 정의된 SQL을 실행하여 고객 창의 데이터를 채운다. 그리고 사용자가 고객 창에서 하나의 행을 선택하면 선택된 행의 고객 ID 값만 주문 폼에서 저장하여 관계를 맺게 된다. 미리 정의된 SQL을 관계 SQL이라 하며 이 예에서는 다음과 같다.

```
Select 고객.ID, 이름, 주소
From 고객, 고객 신용
where 고객.ID = -2,000,000,000
(-2,000,000,000의 의미는 5장에서 설명)
```

[그림 4]의 폼들은 [그림 3]의 ERD의 엔터티에서 자동 생성된다. 각각의 엔터티들 ([그림 4]의 고객과 주문)은 같은 이름을 갖는 각각의 폼들로 매핑되고 둘 사이의 관계는 주도 개체 - [그림 4]의 관계에서 화살표의 시작점을 주도 개체라 하고 끝 부분을 대응 개체라 정의한다 - 의 버튼으로 매핑된다. 주도 개체 버튼은 [그림 4]의 예에서 주문의 버튼 (검은색 네모) 이며 이 버튼을 누르면 [그림 3]의 '고객' 창이 뜬다. 이 창에서 사용자는 대응 개체를 선택하여 주문과 고객 폼들 사이의 관계가 형성된다.

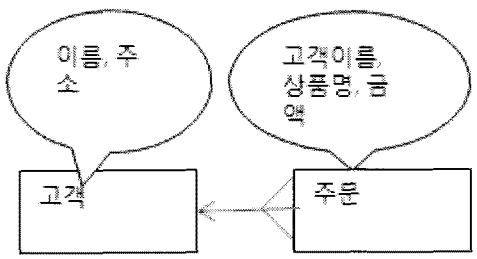

그림 3. 주도 개체와 대응 개체의 확장 표기법

이 자동 생성은 개체의 필드에 하나의 폼 필드를 생성하는 간단한 작업이다. 이 폼 생성과 함께 고객과 주문에 해당하는 데이터베이스 테이블을 각각 자동 생성

한다. 우리의 예에서는 [그림 5]의 테이블이 생성된다.

[그림 5]의 예에서 고객_ID 외래키 (foreign key)는 고객 ID를 참조한다. 폼 생성과 이에 따른 테이블 생성을 하면 이제는 어떻게 ERD의 관계를 구현하느냐 하는 문제가 남는다. 이를 위하여 우리는 주도 개체인 주문 폼에 상대 개체인 고객 폼과 관계를 맺게 하는 버튼을 삽입한다. 이 버튼을 누르면 상대 개체의 객체들이 아래와 같이 보이고 사용자는 이 들 중의 하나를 선택한다. 예를 들어 [그림 1]에서 사용자가 '홍길동'을 선택하면 주문 테이블의 '고객_ID' 에 '1'이 저장되어 주문과 고객의 관계 인스턴스가 생성된다.

고객		
ID	이름	주소
1	홍길동	서울
2	이순신	분당

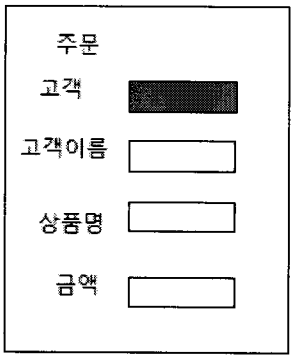

그림 4. 주문과 고객 폼 예제

이렇게 SQL을 활용하면 두 가지의 장점이 있다. 첫째 대상 폼의 창에서 대상 폼의 인스턴스뿐만 아니라 관련된 다른 폼 필드도 함께도 함께 보여주어 사용자의 선택을 용이하게 한다. 두 째 관계 생성 시에 관계를 맺을 수 있는 대상 폼의 개체에 대한 제약 조건의 표현력을 극대화 한다. 첫째 장점을 설명하기 위해서 예를 들어 [그림 6]의 ERD의 고객-고객신용-주문 예에서 고객 신용의 신용등급 필드를 고객 창에서 보여 준다. 이 예에서 고객 창에서 고객 선택 시 신용등급 정보를 포함하여 보여주려 한다고 한다. 그러면 우리는 다음의 SQL로 고객 창을 채운다.

```
Select 고객.ID 이름, 주소
From 고객, 고객 신용
```

이 SQL에 해당하는 고객 창은 [그림 7]과 같다. 즉, 고객 버튼을 누르면 고객 테이블의 인스턴스가 보이는 것이 아니라 관련 SQL을 실행하여 결과를 보인다. 그렇기 때문에 사용자에게 고객 선택을 더 용이하게 하기 위하여 다른 테이블의 필드도 함께 (고객신용의 신용등급) 보여 줄 수 있게 된다. 이때 SQL은 고객의 ID를 포함하여야 한다. 왜냐하면 이 고객 ID가 주문 테이블의 고객_ID 저장되어야 하기 때문이다.

고객

ID	이름	주소

주문

ID	고객이름	상품명	고객_ID

(금액은 계산 필드이므로 테이블에 나타나지 않는다)

그림 5. 자동 생성된 주문과 고객 테이블 예제

두 째 장점은 관계 생성 시에 관계를 맺을 수 있는 대상 폼의 개체에 대한 제약 조건의 표현력을 극대화 한다는 점이다. 이때 제약 조건을 구현하는 프레디켓 (predicate)이 다른 폼의 필드 (위의 예에서 신용등급)를 포함할 수 도 있다. 이 장점도 예를 들어 설명하겠다. 신용 등급이 우량인 고객만 주문 할 수 있게 할 수 있다. 주도 개체의 필드를 포함하는 더 복잡한 제약 조건이 SQL로 표현 될 수 있다.

[그림 6]의 ERD에서 우리는 우량 고객에게만 주문을 받는다고 가정하자. 그러면 주문 폼의 고객 버튼을 클릭하면 우량 고객만 보여야 한다. 우리는 이러한 제약 조건 문제를 SQL을 버튼에 관련시켜 해결하였다. 우리의 예에서 고객 버튼에 관련된 SQL을 아래와 같이 정의한다.

```
Select 고객.ID 이름, 주소
From 고객, 고객 신용
Where 고객신용.고객_ID = 고객.ID
And 고객정보.신용등급 = '우량'
```

우리는 대상 폼 (고객) 대신에 SQL을 관련시킴으로써 우량 고객만 보여 질 수 있게 하는 표현력을 얻을 수 있게 됐다.

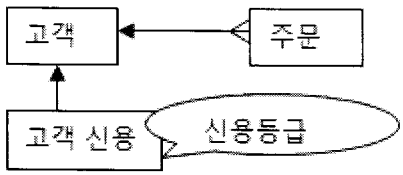

그림 6. 주문, 고객, 고객 신용 ERD 예제

고객			
ID	이름	주소	신용등급
1	홍길동	서울	A
2	이순신	분당	B

그림 7. 관계를 맺는 고객 선택 창

좀 더 복잡한 예를 든다. 우리는 주문 할 수 있는 고객이 우량 고객 또는 매출이 100 개 이상으로 가정하자. 그러면 아래의 SQL이 이러한 조건을 만족시킨다.

```
Select 고객.ID 이름, 주소
From 고객, 고객 정보, (Select 고객.ID as 고객ID,
        sum(수량) From 고객, 주문
Where 고객_ID = 고객.ID Group by 고객.ID
        having sum(수량) > 100)
Where 고객_ID = 고객.ID
And 고객정보.신용등급 = '우량'
```

이때 SQL은 반드시 고객.ID를 포함하여야 하며 시스템은 반드시 이 고객.ID를 주문의 외래키인 고객_ID이 저장시켜야 한다. 왜냐하면 이 절차의 목적은 고객과 주문의 관계를 맺는 것이기 때문이다.

카디널러티가 일대일 일대다 외에 다대다 관계를 예를 들어 보자 ([그림 8]).

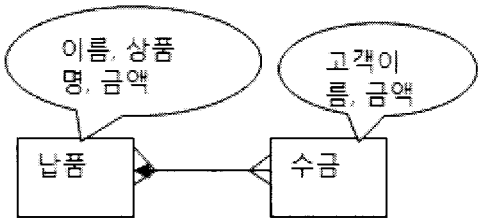

그림 8. 다대다 관계의 납품, 수금 ERD 예제

납품				
ID	이름	상품명	금액	나머지
1	홍길동	마스터	100	50
2	이순신	룩스	200	100
나머지	50		할당	30

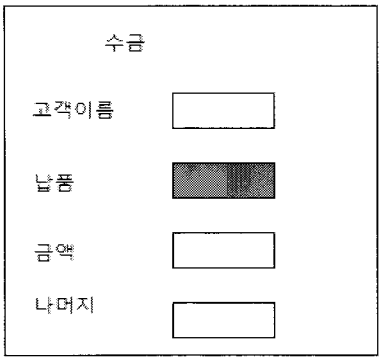

그림 9. 다대다 관계의 납품, 수금 폼의 실행 폼 예제

[그림 9]에서 수금의 납품 버튼을 누르면 윗부분의 창에 납품 리스트가 뜬다. 왼쪽 폼의 마지막 행에서 나머지는 선택된 납품 건의 수금이 아직 안 된 금액이고 할당은 수금된 금액 중에서 얼마를 선택된 납품 건에 할당 시킬 것인가를 입력한다. 이 경우 납품 창을 구성하는 납품 인스턴스들을 SQL로 정의함으로써

1. 관계가 있는 다른 개체의 필드들을 덧붙일 수 있다
2. 계산 필드를 더할 수 있다
3. 프레디켓을 이용하여 다양한 제약 조건을 구현할 수 있다

위의 주문 고객 예에서는 1번과 3번의 경우이고 바로 위의 예에서는 2번과 3번의 장점을 취한 경우이다. 물론 1, 2, 3번의 모든 장점을 취한 경우도 많이 있다.

4.2 매핑의 정의

필드의 종류 세 가지 (저장 필드, 참조 필드, 계산 필드) 중 관계에 의해서 생성 되는 참조 필드를 SQL로 값을 얻을 수 있다. 개발자가 각 관계에 제공하는 SQL은 주도 개체 각각에 대하여 관계를 맺는 대상 개체 인스턴스를 의미하는 것으로 1-1, 1-다 관계에서는 그 결과가 하나의 행으로 나타난다.

이 행은 관계에 참여하는 대상 개체 인스턴스의 ID를 포함하여 여러 개의 필드 값을 나타낸다. 이 필드는 주도 개체에서 참조 필드에 그 값을 제공한다. 우리의 고객-주문 예에서 (실행 모드의 고객-주문) SQL는 고객의 ID, 이름, 주소 값을 제공하고 이들 중 이름은 주문 폼의 고객이름 필드의 값으로 매핑된다. 다-1, 다-다 관계에서는 하나의 주도 개체 인스턴스에 대응 되는 대상 개체 인스턴스가 여러 개이므로 주도 개체의 참조 필드로 매핑될 수 없다. 이 경우는 관계의 상태를 추적하는 나머지 필드를 양 측 개체에 추가하는데 이 나머지 필드는 계산 필드로 분류된다.

4.3 계산필드의 정의

계산 필드를 SQL로 구현하는 기술을 의미한다. 각 계산 필드에 개발자는 이를 구현하는 SQL을 제공하는데 이 때의 SQL은 단지 하나의 값을 제공하여야 한다. 특이점은 개체의 인스턴스들 중에서 현재 폼에서 열린 개체를 의미하는 -1,000,000,000을 SQL에 포함시키고 실행 시에 개체의 현재 인스턴스의 ID 값으로 치환되는 기술을 사용한다. 계산 필드를 SQL로 구현하는 방법은 언뜻 보기에는 기존의 방법 필드들 사이의 연산 식으로 표현 에 비해 복잡해 보이지만 집합 값 (aggregate)을 표현하는 데 적합하다. 예를 들어 주문의 총금액은 주문 항목 (item)의 금액 각각의 합으로 정의 하는 것은 기존의 방법 보다 SQL로 표현하면 간단히 해결된다.

또 다른 계산 필드의 주요한 활용은 이들이 다-다, 다-1 관계의 상태를 추적하는 데 사용된다. 예를 들어 납품-수금은 다-다 관계이다. 이때 납품과 수금에 나머지 필드를 추가하는데 이는 계산 필드로써 납품 측 나머지는 납품을 하였으나 수금 되지 않은 금액 부분을 나타내고 수금 측 나머지는 수금을 하였으나 아직 납품 건에 할당 되지 않는 금액 부분을 나타낸다. 이러한 나머지 값을 계산 하는 데는 SQL이 무척 용이하다.

[그림 9]의 예에서 납품 창의 나머지 필드는 계산 필드이다. 이 계산 필드도 SQL로 정의할 수 있다

```
납품_나머지 =
select 납품.ID as 납품ID, 금액 - sum(value) as 나머지
from 납품, 납품_수금
where 납품.ID = 납품_ID group by 납품.ID, 금액
```

```
납품 창 전체는 다음의 SQL로 정의된다.
납품_창 =
select 납품ID, 이름, 상품명, 금액, 나머지
from 납품, 납품_나머지
where 납품.ID (+) = 납품ID
```

이에 "수금의 고객과 납품의 고객은 같아야 한다." 하는 조건을 덧붙일 수 있다. 이를 SQL로 구현하려면 아래의 프레디켓을 첨가 할 수 가 있다.

```
"이름 in (select 고객이름 from 수금
where 수금.ID = -1,000,000,000)"
```

위의 predicate에서 -1,000,0000,000은 현재 열린 수금 창의 ID에 해당한다. 위의 납품, 수금 창 그림에서 사용자는 현재 열린 수금 건에 선택된 납품 건을 관계 맺으면서 할당 값을 (예로 30) 입력하면 이는 납품_수금 테이블에 저장된다.

insert into 납품_수금 values(*납품_ID, 수금_ID, 30)*

V. 실행모드에서의 조회 리라이트 (query rewrite)의 활용

폼의 필드는 크게 세 종류로 나눌 수 있다. 첫째는 저장 필드로 데이터베이스에 값이 직접 저장된다. 둘째는 계산 필드로 이 필드를 정의하는 SQL이 실행 시에 수행되면서 그 값을 직접 계산한다. 셋째는 참조 필드로

이는 관계를 맺은 다른 폼의 필드 값을 가져오는 필드로 값을 가져오는 작업을 관계에 정의된 SQL을 이용하여 다른 폼에서 값을 가져온다. 예를 들어 [그림 1]에서 주문 폼의 고객이름 필드는 참조 필드로써 관계를 표현하는 고객 버튼을 눌러 고객 인스턴스를 선택하면 이에 따라 그 값이 채워진다. 고객 인스턴스가 한번 정해지면 주문 테이블의 고객_ID의 값이 정해진다. 그러면 주문 폼의 실행 시에 고객이름 필드는 이 고객_ID 값과 고객과의 관계에 설정된 SQL을 이용하여 자동으로 그 값이 유추된다. 이를 예를 들어 설명하면, 고객 관계에 설정된 SQL은 다음과 같다.

Select 고객.ID as ID, 이름, 주소
From 고객 Where 고객.ID = -2,000,000,000

위에서 -2,000,000,000 값은 실제에서 발생하지 않는 값으로 조회 리라이트용으로 쓰인다. 고객 버튼을 누르면 고객 창이 뜨는데 고객 창의 구성은 위의 SQL이 만들어 낸다. 다시 말하면 위의 SQL에서 "고객.ID = -2,000,000,000" 프레디켓을 지우고 이를 수행한다. 우리 예에서

Select 고객.ID as ID, 이름, 주소 From 고객

로 query rewrite 되고 이 것이 수행되어 그림 10의 창이 수행되고

고객		
ID	이름	주소
1	홍길동	서울
2	이순신	분당

그림 10. 자동 폼의 버튼에서 실행되는 고객 창

사용자가 행을 선택하면 ID 값 (예를 들어, 2) 이 고객 테이블의 고객_ID에 저장되면서 관계 인스턴스가 생성된다. 일단 고객_ID 가 결정되면 주문 폼은 참조 필드인 고객이름을 위의 관계에 설정된 SQL을 이용하여 자동 유추된다. 이는 SQL의 -2,000,000,000 값을 사용자 선택한 고객.ID 값 즉, 주문 테이블의 고객_ID 값 (이 예에서, 2)으로 치환하면서 이루어진다. 즉

Select 고객.ID as ID, 이름, 주소
From 고객 Where 고객.ID = 2

가 실행되고 그 결과 값 (이 예에서 이순신)이 주문 폼의 고객필드로 자동 매핑된다. 즉, 참조 필드는 (1) 관계에서 설정된 SQL에서-2,000,000,000 프레디켓을 제거하고 실행하여 대상 개체의 인스턴스들이 보이는 대상 창이 뜬다. (2) 사용자가 인스턴스를 선택하여 대. 상 개체를 참조하는 주도 개체 테이블의 외래키에 대상 개체 ID가 저장된다. (3) 주도 개체 폼이 실행되면 참조 필드는 관계 SQL에서 "-2,000,000,000"을 대상 개체 ID로 치환한다. (4) 결과 값을 참조 필드로 자동 매핑한다.

이 치환 방법은 계산 필드에서도 이용된다. 주문 폼에서 수량을 SQL로 정의한다.

Select 단가 * 수량 as 금액 From 상품, 주문
Where 주문.상품_ID (+) = 상품.ID
And 주문.ID = -1,000,000,000

이렇게 정의된 계산 필드 정의는 실행 폼이 수행되면서 -1,000,000,000을 현재 열린 주문 폼의 ID (예를 들어, 7)로 치환 실행하여 값을 유추한다. 예를 들어, 위의 SQL은

Select 단가 * 수량 as 금액
From 상품, 주문
Where 주문.상품_ID (+) = 상품.ID
And 주문.ID = 7

치환 실행하여 결과 값을 주문 폼 (ID가 7인) 의 금액 필드에 돌려준다. 즉 계산 필드의 값은 (1) 정의된 SQL에서 -1,000,000,000을 현재 열린 주도 개체의 폼의 ID로 자동 치환한다. (2) 치환된 SQL을 실행하여 값을 자동으로 구한다.

이 뿐만 아니라 -1,000,0000,000 과 -2,000,000,000을 혼용하여 쓸 수도 있다. 예를 들어 아래의 고객, 주문 예에서 창고소재지가 속한 광역권에 위치한 고객에게만 주문을 받는다고 한다. 이 경우 고객 관계는 다음의 SQL로 정의한다.

Select 고객.ID as ID, 이름, 주소 From 고객
Where 고객.ID = -2,000,000,000
And extractBigArea(주소) in (select 창고소재지 from 주문
where 주문.ID = -1,000,000,000)

이 SQL,은 주도 개체에서 대응 개체를 참조할 때 대상 창에 보이는 대응 개체를 제한한다. [그림 11]에서 주문 폼에서 고객 폼의 창고 소재지가 '경기도'인 상태에서 고객 버튼을 누르면 고객 창이 뜨는데 이 때 모든 고객이 보여 지는 게 아니라 주소가 경기도인 인스턴스만 보이고 사용자 는 이들 고객 만 선택할 수 있기 때문에 우리는 관계 생성 시에 제한 조건을 위 SQL로 구현할 수 있음을 보여준다.

고객		
ID	이름	주소
1	홍길동	경기도 성남시 서안 아파트
2	김갑순	경기도 안산시 대우 아파트

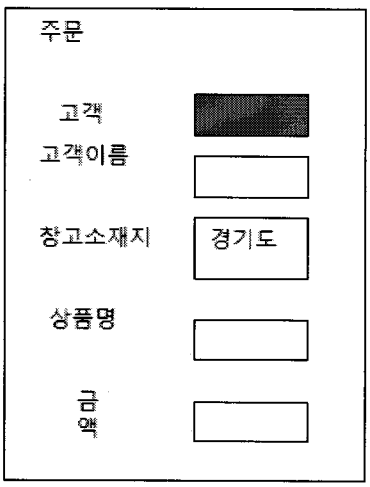

그림 11. 주문 폼에서 고객 폼으로 관계를 맺는 예제

위의 SQL에서 -2,000,000,000이 속한 절 "고객.ID = -2,000,000,000"을 제거하여 SQL을 실행 시키면 고객 창에서 필요한 데이터를 얻을 수 있다. 그리고 사용자가 홍길동을 선택하면 열린 주문 개체와 사용자가 선택한 고객 인스턴스 사이에 관계가 생성된다. 그러므로 주문 폼의 고객이름 필드는 이 SQL에서-2,000,000,000을 '1' 로 치환하고 SQL의 주문.ID를 현재 열린 폼의 주문 ID(예를 들어, 7)로 치환하여 SQL을 실행하여 고객의 '이름' 필드 값을 주문 폼의 '고객이름' 필드 값으로 매핑하면 그 값을 얻을 수 있다.

즉, 위의 관계 생성 알고리즘 (1) (2) (3) (4)을 약간 수정하면 (1) 관계에서 설정된 SQL에서 -2,000,000,000을 포함한 절을 제거하고, -1,000,000,000을 현재 열린 주도 개체 ID로 치환 하여 SQL을 실행한다;.(2) 변화 없으며; (3) -2,000,000,000을 관계 설정된 대상 개체 ID로 치환하면 -1,000,000,000을 현재 열린 주도 개체 인스턴스 ID로 치환한다; (4) 변화 없다.

다대다 관계에서도 같은 치환 방법론이 적용된다. 위의 수금, 납품 예에서 관계에서 설정된 SQL은 아래와 같다.

납품_창 = select 납품ID, 이름, 상품명, 금액, 나머지
from 납품, 납품_나머지
where 납품.ID (+) = 납품ID and
이름 in (select 고객이름 from 수금 where 수금.ID =
-1,000,000,000)

위의 SQL에서 -1,000,000,000을 현재 열린 수금 건의 ID (예를 들어, 8)로 치환하면 같은 고객의 수금 대상 창의 데이터를 얻을 수 있다. 다대다 관계에서 주도 개체에서 관계 상태를 추적하는 계산 필드가 필요하다. 우리의 예에서는 현재 열린 수금 금액이 얼마나 납품 건에 할당 되었고 그 나머지를 알 필요가 있다. 이 계산 필드를 수금_나머지라고 한다.

수금_나머지 =
select 수금.ID as 수금ID, 금액 - sum(value) as 나머지
from 수금, 납품_수금
where 수금.ID = 수금_ID and 수금.ID =
-1,000,000,000
group by 납품.ID, 금액

위의 계산 필드에서 -1,000,000,000을 현재 열린 수금 ID (예를 들어 8)로 치환하면 현재 수금 건에서 납품 건에 할당 되고 남은 금액을 추적할 수 있게 된다. 다대다 관계에서는 대상 개체 (넓게 정의하면 대상 개체의 ID를 주키로 하는 SQL) 의 값을 주도 개체의 값으로 직접 매핑할 필요가 없기 때문에 -2,000,000,000이 쓰이지는 않는다. 대신에 대상 개체와 주도 개체에서 다대다 관계를 추적하는 나머지 계산 필드를 정의한다. 이 나머지 필드들은

납품_나머지 =
select 납품.ID as 납품ID, 금액 - sum(value) as 나머지 할당
from 납품, 납품_수금
where 납품.ID = 납품_ID and 납품.ID = -1,000,000,000
group by 납품.ID, 금액

로 정의되고 이 납품_나머지가 납품 창의 SQL에서 쓰일 때는 납품.ID = -1,000,000,000 절을 지우고 조인된다. 이때 이는 개발자가 관계 설정 SQL을 정의할 때 매뉴얼로 직접 입력한다.

VI. 실험

우리의 프레임워크[15]를 2008년도 2학기 두 개의 실습 과목을 통해서 실험하였다. 12개의 팀이 대상 조직을 선택하여 프레임워크를 통하여 DB 설계 및 업무 프로그램 프로토타입을 생성하였다. 방법론은 제임스 마틴의 정보공학방법론 [16]을 사용하여 '프로세스 분할도'를 먼저 작성하고 ERD를 그린 후에 ERD를 프레임워크에 입력하고 관계에 SQL을 정의하여 프로토타입을 생성하게 하고 실험 데이터를 입력하면서 프로토타입을 검토하였다. 우리는 이러한 과정에서 ERD를 그리는 작업이 반복적인 과정을 거쳐 대상 업무를 좀 더 깊이 이해하면서 ERD를 완성시켜 나가는 과정이라는 것을 발견하였다. 특히 프로토타입에 데이터를 입력하면서 모델과 업무를 좀 더 잘 이해하게 되었고 설계 과정의 여러 이슈 (issue)를 발견할 수 있었다.

표 2. 실험 기업과 데이터모델 크기 (up는 unit process로서 단위 프로세스로 불린다)

대상 기업	스포츠센터	통관 업무	마케팅	소비자보호원	초등학교	컨텐츠	식재료납품	건설업	육군인사처	학원	보험대리점	주점
up	23	4	16	8	20	5	15	5	6	10	4	9
엔터티	45	29	31	24	42	20	35	36	32	36	31	23
관계	44	28	34	37	15	19	34	35	31	35	30	22

실험의 특성상 정량적인 분석이 어려워 다음과 같은 실험자 평을 기술하였다.

1. ERD나 업무 프로세스를 프레임워크를 통해 더 잘 이해할 수 있었다 (스포츠센터).
2. 개체와 개체 간의 관계를 육안으로 쉽게 확인할 수 있었다 (통관 업무).
3. 관계 표현이 간편하였고 SQL 입력에 많은 시간 소모가 필요 없었다 (마케팅).
4. 테이블과 테이블 사이의 관계를 고민하게 하였다. 한 테이블에서 여러 개의 테이블 나뉜 정보가 다른 테이블로 합해져 가는 것을 구현 할 수 없었다 (소비자 보호원).
5. 생활 기록부에서 학생의 신상, 출결 사항, 학년 반 이력을 동시에 불러오는 기능을 구형 할 수 없었다 (초등학교).
6. 트랜잭션 업무를 쉽고 효율적으로 확인해 볼 수 있다는 점과 ERD 작성 시 전체적인 흐름을 프로토타입으로 짐작해 볼 수 있어 좋았다 (컨텐츠).
7. 업무 상 필요로 하는 속성을 발견할 때 많은 도움이 되었다. SQL의 문법 체크 기능이 없어 불편하였다 (식재료 납품).
8. 자료를 기입한 후 다시 볼 때 입체적으로 검토할 수 있었다. 각각의 폼 연결 시 SQL 방식을 사용하여 누구나 할 수 있도록 정형화 되었다 (건설업).
9. 엔터티의 관계를 이해하고 파악하는데 도움이 되었다. 자료를 대응 개체에서 주도 개체로 가져오는 기능은 다른 프로그램과 차별화되는 기능이다.

업무 프로세스를 이해하는 데 도움이 되었다. 한 엔터티의 속성이 서로 다른 속성으로부터 값을 가져와야 하는 경우를 구현 못하는 단점이 있다. 데이터베이스 스키마를 이해할 수 있도록 제작된 프로그램으로 스키마와 업무 로직을 학습하는 프로그램으로 유용하다. 계산 필드는 업무 로직 구현이 가능하다 (육군 인사처).

10. 관계를 맺을 때 사용하는 SQL이 어렵지 않았다. 데이터 입력 형식이 복잡하지 않아 쉽게 DB를 짤 수 있다. 테이블 행이 20개를 넘어가면 커서 이동의 지연이 있는 등 시스템이 늦어 졌다 (학원).
11. 테이블 사이의 두 구조 사이의 사상 (mapping)이 편리하고 구조가 쉽게 되어 있다. 철저한 요구조건의 분석을 할 수 있었다 (보험 대리점).
12. 업무 프로세스 분석을 통해 도출해낸 ERD를 프레임워크에 입력하면서 자기 ERD의 문제점을 알게 되었고 이를 통해 업무 프로세스 분석이 얼마나 중요한지 알 게 되었습니다. 자체적인 메모장 기능을 추가 한다면 좀 더 편리할 것 같습니다. 계산 필드의 값을 저장할 수 있었으면 좋겠습니다 (주점).

VII. 결론 및 미래 연구

본 제안 시스템은 요구 사항 분석 시 가장 중요한 산출물인 ERD에서 프로토타입 시스템을 생성한다. 이 프로토타입은 비주얼 (visual)하게 엔터티를 보여주고 관계는 SQL로 정의하여 데이터를 처리하는 로직을 갖추고 있어 ERD를 입체적으로 검토하게 할 수 있다. 더욱이 고객은 이해하기 어려운 ERD 보다는 비주얼 화면을 통하여 본인의 요구사항을 확인할 수 있으며 새로운 요구사항을 도출할 수 있게 된다. 특히 실 데이터를 입력하고 처리하는 부분은 고객의 목표 시스템에 대한 이해를 증대시킨다.

본 시스템의 한계는 GUI 즉 프리젠테이션 로직 [17]을 처리하지 못해 사용자에 전달되는 최종 시스템으로 발전시키지 못한다는 데 있다. 본 시스템에 의해 생성된 데이터 테이블을 최종 시스템에서 사용될 수 있으나 본 시스템이 포함하는 비즈니스 로직, 관계 SQL과 매핑으로 구현되는, 은 최종 시스템에서 사용 할 수 가 없다. 즉, 고객이 원하는 GUI를 작성하지 못하기 때문에 본 제안 시스템으로 요구분석이 끝나면 비즈니스 로직을 새로 구현하여야 하는 단점이 있다. 이 단점을 극복하기 위하여 본 시스템의 비즈니스 로직을 클래스의 메소드로 전달하고 최종 시스템 개발 시에 미 메소드들을 사용하게 하는 아이디어가 논의되고 있다. 이 아이디어를 구체화시키면 본 시스템을 사용하여 생성한 테이블과 비즈니스 로직 둘 다를 최종 시스템의 출발점으로 사용할 수 있어 요구 분석에서 코딩 단계로 자연스럽게 (seamless) 넘어 갈 수 있게 된다.

참 고 문 헌

[1] 나영국, "데이터 모델링 평가 (Review)를 위한 실행 폼 프로토타이핑 (prototyping) 시스템", 데이터베이스연구지 계제 예정, 제25권, 제1호, 2009.

[2] J. Christian, W. Annette, and X. Jurgen, "Generating User Interfaces from Data Models and Dialogue Net Specification," INTECHI, pp.24-29, 1993(4).

[3] H. Balzert, "From OOA to GUIs: The Janus System," IEEE Software, Vol.8, pp.43-47, 1996(2).

[4] F. Bodart, S.-M. Hennebert, J. M. Leheureux, I. Provot, and J. vVanderdonckt, "A Model-based Approach to Presentation: A Continuum form Task Analysis to Prototype," Proceedings of Eurographics Workshop on Design, Specification, Verification of Interactive Systems, Carra, Italy, Focus on Computer Graphics, Springer-Verlag, Berlin, pp.77-94, 1994(6).

[5] M. Elkoutbi, I. Khriss, and R. K. Keller,

"Generating User Interface Prototypes from Scenarios," Proceedings of IEEE International Symposium on Requirement Engineering, pp.150-158, 1999.

[6] R. P. Angel, E. Henrik, H. G. Jhon, and A. M. Mark, "Beyond Data Models for Automated User Interface Generation," Proceedings of British HCI. 1994.

[7] 김정옥, 유철중, 장옥배, "사용성 중심설계를 지원하기 위한 사용자 인터페이스 프로토타입의 생성기법", 제31권, 제1호, 정보과학회논문지 2004(1).

[8] D. Harei, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, a. Shtull-Trauring, and M. Takhtembrot, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," IEEE Transactions on Software Engineering, Vol.16, No.4, pp.403-414, 1990(4).

[9] C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj, "SCR*: A Toolset for Specifying and Analyzing Software Requirements," Proc. of the 10th Annual Conference on Computer Aided Verification, (CAV'98), Vancouver, Canada, pp.526-531, 1998.

[10] H. Tauqeer and M. A. Mian, "An Effort-based Approach to Measure Completeness of an Entity-Relationship Model," Seventh IEEE/ACIS Int. Conf. on Computer and Information Science, pp.463-468, 2008.

[11] G. Marcela, P. Geert, and P. Mario, "Defining and Validating Metrics for Assessing the Understandibility of Entity-Relationship Diagrams," Data and Knowledge Engineering Vol.64, pp.534-557, 2008.

[12] D. Hanbi, L. Ee-Peng, W. L. Handy, and P. Hweehwa, "Visual Analytics for Supporting Entity Relationship Discover on Text Data," ISI 2008 Workshops, LNCS 5075, pp.183-194, 2008.

[13] F. Mathias, H. Philip, and L. Stefan, "Model-driven Instrumentation of Graphical User Interfaces," Second Int. Conf.. on Advances in Computer-Human Interactions, pp.19-25, 2009.

[14] H. Shu-qiang and Z. Huan-Ming, "Research in Improved MVC Design Patterns Based on Struts and XSL," Proc. of the 2008 International Symposium on Information Science and Engineering," Vol.1, pp.451-455, 2008.

[15] E. F. Mohamed, "Introduction to the Computing Surveys' Electronic Symposium on Object-Oriented Application Framework," ACM Computing Surveys (CSUR), Vol.32, No.1, 2000(3).

[16] M. James, "Information Engineering I, II, III," Prentice Hall, 1990.

[17] Brad A. Myers, "User Interface Software Tools," ACM Transactions on Computer-Human Interaction, Vol.2, No.1, pp.64-103, 1995(3).

저 자 소 개

나 영 국(Young-Gook Ra) 정회원

- 1987년 2월 : 서울대학교 전자과(공학사)
- 1989년 5월 : 펜실베니아 스테이트 컴퓨터공학과(공학석사)
- 1996년 12월 : 미시간대학 컴퓨터과학과(공학박사)
- 1997년 1월 ~ 2002년 11월 : 삼성SDS, 한경대학교
- 2002년 12월 ~ 현재 : 서울시립대학교 전자전기컴퓨터공학부 교수

<관심분야> : 데이터베이스, DB 어플리케이션