Entity-Relationship Meta-model Databases with Unknown Meta-Attribute Values

Liwei Xie

University of Tampere
School of Information Sciences
Computer Science
M.Sc. thesis
Supervisors: Zheying Zhang and Peter Thanisch
March 2015

University of Tampere
School of Information Sciences
Computer Science
Xie Liwei
M.Sc. thesis, 55 pages, 8 indexs and Appendix 83 pages
June 2015

Abstract

In the entity-relationship (ER) meta-model, modeling constructs such as entities, relationships, attributes, etc., in the concrete conceptual model are all represented in meta-entity classes, and items of information which qualify the modeling constructs (e.g. whether an entity class in the concrete conceptual model is strong or weak) appear as meta-attributes, associated with a particular meta-entity class. In an agile software development process, conceptual model construction and the consequent database design often proceeds on the basis of incomplete requirements specification, especially in the early iterations, and the information on a model construct, i.e. the concrete value of a meta-attribute, is hard to predict and specify. In this thesis, we deal with the unknown values by extending the ER meta-model to allow meta-attributes to have the value "unknown". These unknowns are taken into account in the physical database design. We have implemented the "triggers" to refactor the physical database when the unknown attribute values in the conceptual model become known. The thesis demonstrates that the database design based on the concrete conceptual model can be re-factored when missing information is identified and updated.

Keywords: Entity-relationship Model, Meta-model, Incomplete Information, Agile Software Develop, Database Refactoring

Contents

1. Introduction

In time-boxed, incremental development methodologies, such as Agile [Agile Alliance, 2015], the focus is on the early delivery of a preliminary version of the required system. Given the short duration of each time-boxed step (in some cases just a few days), it is not practical to expect that requirements analysis will be complete in the earliest increments. Thus, in the context of database development, activities such as conceptual modeling, logical design, physical design, implementation and developing database application software must all proceed on the basis of incomplete requirements and, consequently, incompleteness in each of the preceding activities. By contrast, the traditional approach to database development is based on a presumption that each step is completed before the subsequent step is commenced. This thesis looks at a way of adapting the database development process so that it works in the context of an incremental development methodology.

The thesis explores the possibility that an accelerated initial development cycle can be achieved by exploiting a distinctive feature of the Entity-Relationship (ER) model [Chen, 1976], namely that if requirements are incomplete, it is often possible to work around the incompleteness by producing a more generic conceptual model. That generic model will be more complex and less efficient, but it will actually work correctly. We describe a database development methodology, exploiting that feature of the ER model, which can be used in the context of incremental development methodologies. The methodology includes the processing of late-arriving requirements which can lead to the automatic update of the conceptual model and even to the automatic refactoring of the implemented database.

ER modeling is widely used for the conceptual modeling phase of the database development process. In common with other modeling formalisms, several groups of researchers have associated a meta-model with the ER model; see, for example, [Teorey et al. 1986] and [Fidalgo et al., 2012]. In this thesis, we shall only concern the meta-model of Fidalgo et al..[2012]. As well as helping us to understand the constructs in the ER modeling formalism, an ER meta-model is useful because it allows us to construct a generic meta-model database in which we can store any concrete ER model.

In a concrete ER model, there will be a set of entity classes, a set of relationships between entity classes, a set of attributes labeling entity classes or relationships, etc. Each of those sets of constructs in the concrete conceptual model is stored in a separate meta-entity class in the meta-model. In this arrangement, the information which qualifies those modeling constructs (e.g. whether an entity class is strong or weak, whether a relationship is mandatory or

optional or whether or not an attribute is an identifier) appear as meta-attributes, and each associated with a particular meta-entity class, in the meta-model. In this thesis, the ER meta-model is extended to allow meta-attributes to have the value “unknown”. This corresponds to the situation in which the conceptual modeler has already decided on the entities and relationships, but the requirements analysts have not yet been able to elicit the information relevant to assigning concrete values the meta-attributes. This situation occurs in time-boxed incremental development methodologies such as agile software development methods.

Consider the following scenarios. In the first scenario, suppose we have an incremental development project which does NOT use our approach. The conceptual modeler cannot delay releasing a conceptual model for use by the project, so the model will either (a) cover a limited area of the requirements for which full information is available on time or (b) the conceptual modeler must make guesses about the missing information. If any of those guesses are wrong, then the conceptual modeler must change the model in a subsequent increment, in which case the other people working on the project must change their database design and the application software. Now imagine that the same project used our approach. The conceptual modeler is able to mark up features of the model as "Unknown". The requirements analysts are able to obtain a list of questions for which they must find answers in order to resolve these unknowns. The conceptual model can then be updated automatically and the database design can be refactored automatically.

The content of the thesis is as follows. In Chapter 2 and 3, we review the relevant literature about agile software development and database refactoring. In Chapter 4, we describe the ER model and meta-model, including the implementation of the meta-model database. In Chapter 5, we describe the extension to the meta-model, which allows an instance of a meta-attribute to have the value “Unknown”. In Chapter 6, we document the trigger mechanism which has been implemented in order to allow an instance of a meta-attribute with the value “Unknown” to be updated to take on a particular value. In Chapter 7, we give our conclusions. In the Appendix, a representative selection of the code developed as part of this project is presented.

2. Agile software development

Agile software development is a software development approach in which requirements and solutions evolve through collaboration between self-organizing and cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to changes [Agile Alliance, 2015]. The agile approach borrows practices from other development process models, like iterations and increments [Beck, 1999]. It can be understood as integration based on the original software development method, and inherits many advantages from those methods. For example, "In the agile software development process, we can get working software every two weeks. This very short circulation makes the customers see the results of the software they paid for promptly and quickly." [Martin, 2003] In addition, the agile approach allows a development team to react to changing requirements, and to find out the risk earlier by the early iterative, etc. Drawing many advanced methods and experiences, the agile approach is widely applied in software development.

2.1. The core content of the agile software development

At the beginning of 2001, many software development groups fell into the process expansion problem [Martin, 2003]. To deal with the problem, seventeen industry experts got together and summarized several values and principles which would improve the development team's work efficiency, as well as its ability to react to changes. They presented a value statement, the manifesto of the agile alliance as below.
"Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan" [Agile Alliance, 2015]

2.1.1 Individuals and interactions over processes and tools

In the traditional development methods, the processes and tools are seen as the way to manage the product development and everything associated with it, people and the way they approach the work must confirm to the processes and tools. But this makes it hard to accommodate new ideas, requirements and thinking. However, the agile approaches value the people over process. This emphasis on individuals and teams puts the focus on people and their energy, innovation, and ability to solve problems [Layton, 2012]. The processes and tools are also used in agile project, but they are intentionally streamlined and directly support

product creation. The more powerful tools and processes, the more it should spend on its care. But with people front and center, the result is a leap in productivity. The agile is human-centric approach, and it allows each people to contribute unique value to their software development project depend on their ability. This makes the people easily adapted to new ideas and innovations. After several projects, every team members can improve their ability.

2.1.2 Working software over comprehensive documentation

Documentation is a means of communication in a development team. It records and describes what the software does or shall do, and how a development team proceeds with the software development, etc. Software development without documentation is a disaster. However, too many documents are much worse than too few. It takes a lot of time to compile the documents, and even more time to keep the consistency of the documents. In an agile project, documents should be composed to describe the principle and structure of the system, but these documents should be brief and topic-salient, namely, high level structure of the system and the preliminary design should be demonstrated within several pages. In this way, communication becomes quite important due to the brief documents. Old members train new ones by interacting and talking. The best document is code and teamwork. Many teams pay too much attention to the document instead of code, which usually delays the process. There is a simple rule, i.e.: "Produce no document unless its need is immediate and significant" [Martin 2003]. In an agile project, the developers submit the work project as soon as possible, and frequently.

2.1.3 Customer collaboration over contract negotiation

Typically, developers are reluctant to start developing software without fully understanding customers' requirements. This mode might be attractive to customers, but when the developers have to make decisions based on incomplete knowledge, there can be a higher risk of failure. In addition to the contract or statements of the expected work, frequent feedback is highly required. Cooperation between customers and the development group is the key to success. Requirements always change so that many provisions in the contract will become valueless before the project is completed. Therefore, only with close cooperation with developers as well as frequent and effective feedbacks, customers may eventually get satisfied with the delivered software.

2.1.4 Responding to change over following a plan

Change is a valuable tool for creating great products [Layton, 2012]. Project teams that can respond quickly to customers, product users, and the market in general, are able to develop relevant, helpful products that people want to use. However, traditional project development approaches use rigorous change management procedures and budget structures that cannot accommodate new product requirements, which make changes difficult. And traditional project team often blindly follows a plan, missing opportunities to make more valuable product. By contrast, agile projects accommodate change systematically. The flexibility of agile approaches actually increases project stability, because change on agile projects is predictable and manageable.

2.2. Agile development model

Since the "the manifesto of the agile alliance" was proposed in 2001, the software industry has drawn great attention to the agile software development approach. Many specialists and organizations of software domain proposed plenty of agile models such as XP (Extreme programming) [Beck and Fowler, 2000], Scrum [Hirotaka and Ikujiro, 1986], ASD (Adaptive software development) [Highsmith, 2002], DSDM (Dynamic systems development method) [Richards, 2007], FDD (Feature-driven development) [Coad et al, 1999], etc. Among these models, Scrum is the mostly used one.

To some extent, software development is just like developing a new product. It is impossible to specify the final version of the software at the very beginning. Research, creation and unsuccessful attempts are indispensable before the birth of good software. And there is no unique process that can ensure success. Scrum compares the development team to football team [Hirotaka and Ikujiro, 1986]. There is a clear common goal which is the brief description for the overall task that needs to be accomplished. It provides an undeviating goal which does not consider about specific constrains, and this goal should be clear and measurable. When disagreements or obstacles appear during the development process, this common goal can organize all the stakeholders as a group and help them to eliminate their cognitive conflict of the project direction and target. Developers should be familiar with the model and technique that will be used in the process. Every team member should be able to work independently but also cooperate and communicate closely with others when necessary. Scrum has good applicability, and it guarantees noticeable progress in every sprint, and further the completion of the final goal.

2.2.1 Scrum method principle

Scrum borrows some concept from industrial process control. It is believed that software development process is rather an empirical process than a defined one. A defined process is describable and predictable. This repeatable execution will produce an expected result which can also be optimized by using some scientific methods. An empirical process handles the development as a black box. Developers keep measuring the input and output of the black box and regulate it with judgments from former observation. In this way, the development will be always under control and a satisfactory result will be reached in the end. If managed as a defined one (such as waterfall model), the process will lack adaptability. It is because everything has been defined before the development, but there must be changes during the development process, then defined process cannot response these changes. By contrast, Scrum is an agile and smart process. It achieves the iterative and incremental development process through adaptive and empirical process management. This makes the Scrum process more flexible, effective and outstanding. It emphasizes on delivering business values which are approved by the stakeholders as soon as possible. It also emphasizes the priority of the delivered function. Meanwhile, through the constantly test and improve, to satisfy the key requirements of stakeholders.

2.2.2 The roles of Scrum

There are three fundamental roles [Schwaber, 1996] in a Scrum group, and they are the product owner, the Scrum master, and the Scrum team. All management responsibilities in a project are divided among these three roles.

A product owner represents the stakeholders of the product development project, also the voice of the customer. He/She cooperates with the Scrum Master and Scrum Team. His main task is to write customer-centric items (typically user stories). A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it [Cohn, 2004]. The product owner needs to prioritize user stories, and adds them to the product backlog. The product backlog represents what the customers and users are really interested in. The Product Owner achieves initial and ongoing funding for the project by creating the project's initial overall requirements, return on investment (ROI) objectives, and release plans. And he/she is responsible for using the Product Backlog to ensure that the most valuable functionality is produced first and built upon; this is achieved by frequently prioritizing the Product Backlog to queue up the most valuable requirements for the next iteration. This role is equivalent to

Product Manager in the traditional project management. Normally it is served by people familiar with the market.

The Scrum Master is the manager of Scrum team. The Scrum Master is responsible for the Scrum process, for teaching Scrum to everyone involved in the project, for implementing Scrum so that it fits within an organization's culture and still delivers the expected benefits, for ensuring that everyone follows Scrum rules and practices, and for managing daily meetings. The Scrum Master is responsible for the success of the project, and he or she helps increase the probability of success by helping the Product Owner select the most valuable Product Backlog, and also by helping the Scrum team to turn the Sprint backlog to functionality. The Scrum Master is responsible for removing any barriers between the development Teams and the Product Owner and customers so that the customers can directly drive development. The Scrum Master is also responsible for showing the Product Owner how to use Scrum to maximize project return on investment (ROI) and meet objectives of the project.

The Scrum team is responsible for developing functionality. Teams are self-managing, self-organizing, they are responsible for figuring out how to turn Product Backlog into an increment of functionality within an iteration and managing their own work to do so. Teams are cross-functional, with all of the skills (analyze, design, develop, test, technical communication, document, etc.) as a team necessary to create a product increment. In Scrum, the Teams are responsible for figuring out how to maximize its productivity itself; the job of planning and executing the work belongs only to the Team. The Scrum Master and others can guide, advice, and inform the team, but it is the team's responsibility to manage itself.

In the traditional project team, the project manager manages and leads all the team members" activities and results, defines their roles and responsibilities, and allocates their jobs. The manager bears full responsibility to the result of the project. However, Scrum is structured to regularly make the state of the project visible to the three managers—the Product Owner, the Scrum Master, and the Team—so that they can rapidly adjust the project to best meet its goals.

2.2.3 Scrum method process

Scrum method includes three phases, as presented in Figure 1.

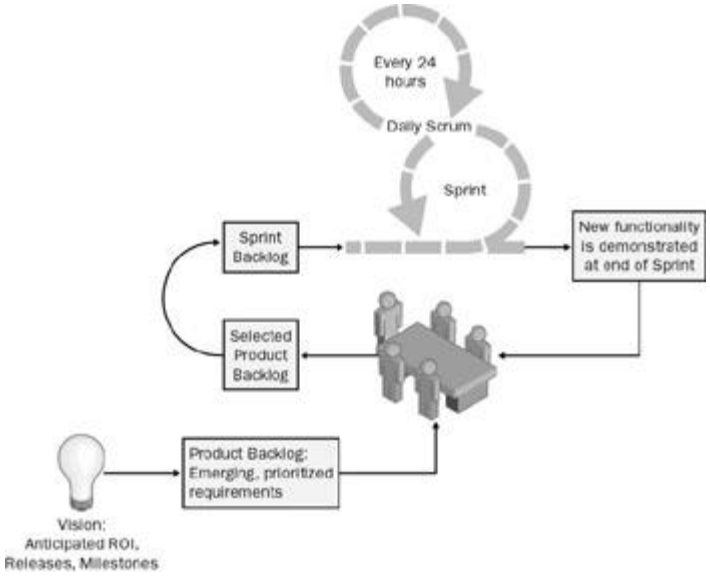


Figure 1 Scrum process overview [Schwaber, 2004]

1. Product Backlog. In Scrum, one project corresponds to one product backlog. The product backlog is an ordered list of the requirements for the project. It consists of rough description of functional and non-functional requirements of the product, bug fixes, improvement proposed by customers, competitive functions, technical updates, and so on. The product backlog will be divided into many question packets and each packet is a set of objects or components, as preparation for defining the sprint.

2. Sprint phase. The development process consists of many iterative sprint processes. These sprint processes go sequentially until the risk assessment tells the product is shippable. One sprint is a series of development activities in limited time (sprint period, normally 2 to 4 weeks), including analysis, design, code, test, etc. Each sprint has a special sprint backlog. The sprint backlog is the list of work the development team must address during the next sprint. The list is derived by selecting product backlog items from the top of the product backlog. When the team believes there is enough work to fill the sprint, the selection process ceases. At the heart of the solution is the team working without interruption for the Sprint. Having selected the Product Backlog for a Sprint, the team has mutually committed to turning it into an increment of potentially shippable product increment in a Sprint days. Once the team makes this commitment, the clock starts ticking. The Sprint is a time-box within which the team does whatever is necessary to meet its commitment. At the end of the Sprint, the team demonstrates the working functionality to the Product Owner.

3. Potentially shippable product increment. Once the risk assessment result shows the product is shippable, the process goes to this phase. The work in this phase includes

installation, system test, regression, final documentation, etc. Scrum process believes the development of software will always continue, unless the risk assessment shows it should be stopped. After the delivery of the product, there will be some consolidate work, for example, maintaining and improving, to sort out the ignored work under the pressure of sprint phase and get prepared for the next period.

Product backlog is the core of Scrum. The requirements are not complete in the early iterations, so the status product backlog is dynamic, and the product owner will constantly change it. These changes come from two sources, i.e. the changes from customers and market, new problems identified during the increment. This also requires the product owner to keep communicating with the team and customers. When new requirements are elicited, the product owner needs to add them to the product backlog, and addes them to the subsequent sprints according to the priority.

Briefly, Scrum is a smart and light weight process with the use of the iterative increment model. It is an agile development method which increases the software development efficiency a lot.

3. Refactoring and database refactoring

The word Refactoring has two definitions depending on context [Fowler et al, 1999]. The first definition is the noun form, i.e. a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. The other usage of refactoring is the verb form, i.e. to restructure software by applying a series of refactoring without changing its observable behavior. So the developer might spend a few hours on refactoring, during which he might apply a couple of dozen individual refactorings. It is often applied in agile software development to continuously clarify and simplify the design of existing code, without changing the external behavior, especially in the continuous integration process.

3.1. Code refactoring

The purpose of refactoring is to make the software easier to understand and modify. Developers can make many changes in software that make little or no change in the observable behavior. Only changes made to make the software easier to understand are refactoring. Refactoring does not change the observable behavior of the software. The software still carries out the same function that it did before. Any user, whether an end user or another programmer, cannot tell that things have changed. For example, changing a function name, replacing duplicated code segments with a function, or improving the code reusability by adding parameters into the interface of a function, etc.

Refactoring is a powerful technology to improve the internal structure of code and to improve software performance. The refactoring process, however, may introduce errors. This risk leads to a general criticism to refactoring technology, i.e. why should we change the internal code structure if it is correct?

3.1.1 Why refactoring

Although refactoring means a piece of extra work, it is worth doing. The benefits from refactoring are summarized as below.

Refactoring improves the design of software. Without refactoring, the design of the program will decay [Fowler et al, 1999]. The developer may change the code in many purposes, for example, changes to realize short-term goals or change made without a full comprehension of the design of the code, etc. The code loses its structure, and it becomes harder to see the design by reading the code. Refactoring is rather like tidying up the code,

removing bits that are not really in the right place. Loss of the structure of code has a cumulative effect. The harder it is to see the design in the code, the harder it is to preserve it, and the more rapidly it decays. Regular refactoring helps code retain its shape.

Poorly designed code usually takes more code to do the same things, often because the code does the same thing in several places. This eliminating duplicate code is an important aspect of improving design. The importance of this lies in future modifications to the code. Reducing the amount of code will not make the system run any faster, because the effect on the footprint of the programs rarely is significant [Fowler et al, 1999]. Reducing the amount of code does, however, make a big difference in modification of the code. The more code there is, the harder it is to modify correctly. There is more code to understand. The developer changes this bit of code here, but the system does not do what he expects because he did not change that bit over there that does much the same thing in a slightly different context. By eliminating the duplicates, the developer ensures that the code says everything once and only once, which is the essence of good design.

Refactoring makes software easier to understand. Refactoring supports smaller classes, shorter methods, less local variation and smaller system coupling. Refactoring requires developers to be more careful about a naming scheme which reflects the designer's intention. If a part of code is too complex to understand, it needs to be refactored. Refactoring helps developers to understand unfamiliar code. When the developer gets unfamiliar code, for instance, he joins a developing project group, his mind probably goes blank and he has no idea where to start. In this situation, he can try to estimate the code, and change the code to better reflect his understanding, and then he test that understanding by rerunning the code to see if it still works. As the code gets clearer, the developer finds he can see things about the design that he could not see before. Had he not changed the code, he might be never would have seen these things, because he is just not clever enough to visualize all this in his head. Refactoring leads developers to higher levels of understanding that otherwise they would miss.

Easier to discover bugs. Refactoring contributes to understanding the code, and also helps to discover bugs. After refactoring the code, the developer can understand the system more deeply. Checking the code again with this understanding, we will find errors more easily. Many bugs exist at very small places of the system. If the developer just read the code again and again, he may be never find bugs. Sometimes, it is still very hard to track those bugs even the toolkit from IDE is used. Refactoring makes the structure of code more clear. Every time we just focus on a single data or activity. This very small step with strict test makes bugs be found more easily.

Refactoring helps developers program faster. It is easy for people to understand that refactoring can help them to improve quality. Improving design, improving readability, reducing bugs, all these improve quality. People will ask does all this reduce the speed of development. The answer is no. A good design is essential for rapid software development. Indeed, the whole point of having a good design is to allow rapid development. Without a good design, developers can progress quickly for a while, but soon the poor design starts to slow them down. They spend time on finding and fixing bugs instead of adding new function. Changes take longer as developers try to understand the system and find the duplicate code. New features need more coding as they patch over a patch that patches a patch on the original code base [Fowler et al, 1999].

3.1.2 Refactoring principle

Kent Back's metaphor of two hats [Fowler et al, 1999] introduces two constantly alternating activities when refactoring is applied in software development, i.e.: adding function and refactoring. When the developer adds function, he should not be changing existing code; he is just adding new capabilities. The developer can measure his progress by adding tests and getting the tests to work. When the developer refactors, he makes a point of not adding function; he only restructures the code. He does not add any tests (unless he finds a case he missed earlier); he only changes tests when he absolutely needs to in order to cope with a change in an interface.

During the software development process, the developer may find himself swapping hats frequently. The developer starts by trying to add new function, and he realizes this would be much easier if the code were structured differently. So he swaps hats and refactors for a while. Once the code is better structured, he swaps hats and adds the new function. Once he gets the new function working, he realizes he coded it in a way that is awkward to understand, so he swaps hats again and refactors. All this might take only ten minutes, but during this time the developer should always be aware of which hat he is wearing.

Unit test is essential to examine the refactoring through test to make sure if it keeps the observable behavior of the code or not. If it can pass the test before but cannot after the refactoring, the observable behavior of the system is probably destroyed during the refactoring.

Unit test is part of the software, not the task given to an independent test department. It means the developer should do the test code himself. This is called self-test code. Passing the unit test means the code is correct, which indicates that the modification does not affect the original function. The developer should pay attention to test compiling and consider the test content.

Small step refers to the identification and realization of the smallest possible change in each refactoring, followed by the unit test. Refactoring changes the programs in small steps. If the developer makes a mistake, it is easy to find the bug [Fowler et al, 1999]. Large-scale refactoring is not recommended. If changes are significant, there is a risk of making mistakes that might impact negatively the functionality of the whole system software. The small steps often include activities such as: 1) finding out what needs to be refactored; 2) compiling unit test cases before the refactoring and updating the test cases according to the refactoring; 3) Running the unit test to ensure the original code is correct.

3.2. Database refactoring

Database is the basis and core of information management system. A database schema of a database system is its structure described in a formal language [Rybinski, 1987] supported by the database management system (DBMS) and refers to the organization of data as a blueprint of how a database is constructed.

Similar to code refactoring, database refactoring [Ambler, 2003] is a simple change to a database schema that improves its design while retaining both its observable and informational semantics. Informational semantics refers to the meaning of the information within the database from the point of view of the users [Ambler et al., 2006]. For example, the developer applies a database refactoring to a character-based phone number column to transform data such as (358)40-333 into 35840333. Although the format has been improved, requiring simpler code to work with the data, from a practical point of view the true information content has not. In other words, developer does not add new functionality or break existing functionality, and does not add new data or change the meaning of existing data. Developers could refactor either structural aspects of their database schema such as table and view definitions or functional aspects such as stored procedures and triggers. Stored procedures are database objects that encapsulate collections of Transact-SQL statements on the server for later repetitive use. They are the equivalent of subroutines and functions in other programming languages [Sunderic and Woodhead, 2002]. And database trigger is a

segment of procedural code that is automatically executed in response to certain events on a particular table or view in a database.

3.2.1 Database smells

Fowler [1997] introduced the concept of "code smell". It implies there are some problems in the code that needs to be refactored. Similarly, there are also some common database smells indicating that the database probably needs refactoring.

Tables with too many columns. When a table contains too many columns, it implies that the table may store values from several categories of entities. If yes, the different categories of entities can be separated into several tables, which will make the table more clear and easy to manage. For example, if a table stores many kinds of phone numbers (e.g. home numbers, work numbers, mobile phone numbers, etc.) and different addresses (e.g. shipping address, billing address, seasonal address, etc.), it is better to normalize this structure by shifting the address information into an Address table and the phone numbers into a Phone Number table, and defining foreign keys in the old table referencing the primary keys of the new tables.

Tables with too many rows. Large tables are indicative of performance problems. It takes people plenty of time to scan such a huge table. For example, it is time-consuming to search a table with millions of rows. Developers can split the table vertically by moving some columns into another table, or split it horizontally by moving some rows into another table. Both strategies reduce the size of the table, potentially improving performance.

Redundant data. Redundant data is a big problem for operational database. Because when data is stored in several places, the opportunity for inconsistency occurs. For example, it is quite common to discover that customer information is stored in many different places in a company. In fact, many companies are unable to put together an accurate list of who their customers actually are [Ambler et al., 2006]. The problem is that in one table there is a customer lives at Murtokatu and in another table he lives at Tapionkatu. In this case, this is actually the customer used to live at Murtokatu but recently moved to Tapionkatu, and he submits his address twice. To avoid this inconformity, the DBA needs to regularly check to find if there is duplicated data.

Fear of change. If the developer fears to change his database schema because he is afraid to break something, it normally means this database needs refactoring. The fear of changing

itself indicates this database schema is too weak and under great technical risk and the situation will get worse throughout the database life cycle.

When these bad smells have been noticed, the DBA has to look into, think about and even refactor it if that makes sense.

3.2.2 The difficulty of database refactoring

Database refactorings are clearly more difficult to implement than code refactorings. Code refactoring only needs to keep the observable semantic, but database refactoring also needs to retaining both its observable and informational semantics. What is worse, when developers refactor their database schema, not only must they rework the schema itself, but also the external systems [Ambler et al., 2006], such as business applications or data extracts, which are coupled to their schema. Generally, database refactoring includes the following tasks.

1) Change the database schema such as the definition of views and tables, stored procedures and triggers;
2) Data migration such as migration under the same data resource; and
3) Change the code for database access, and modify the external system coupling with the database.

So when developer describes the database refactoring, he has to describe all these three changes.

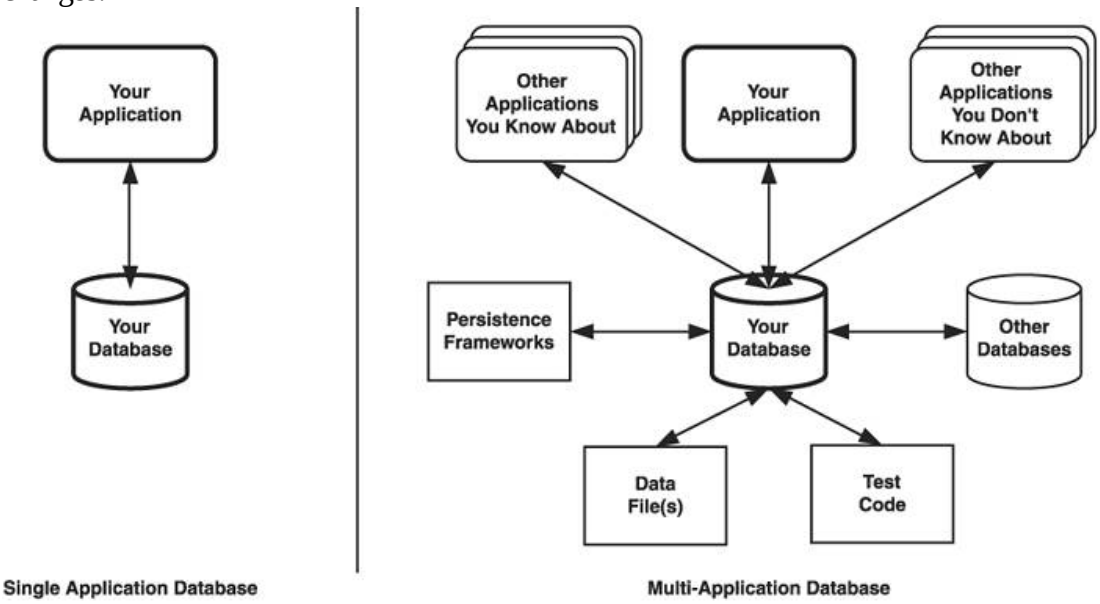


Figure 2 Two categories of database architectures [Ambler et al., 2006]

Database refactoring can become more complicated by the amount of coupling resulting from the database architecture, overviewed in Figure 2. Coupling is a measure of the dependence between two items. The more highly coupled two items are, the greater the chance that a change in one will require a change in another. As single application database is the simplest situation where database only works for one application. It means that database administrators can refactor both the database and the application in parallel and deploy both simultaneously. It is called a ‘stand alone application” or ‘stovepipe database” [Ambler et al., 2006]. The second architecture is much more complicated, because there are several external applications that are interacting with the database. Some of them are out of database manager’s control. In this situation, database manager cannot ensure that all the external applications can be deployed at the same time, so there should be a transition period , in which in this period, both the old schema and the new schema are supported in parallel.

How to put the complexity under control? An integral database refactoring process is one solution.

3.2.3 The process of integral database refactoring

This chapter represents the process of integral database refactoring. It helps the developer to put the complexity of database refactoring under control. When the developer realizes that the schema needs to be refactoring, this process starts. As shown in Figure 3, the process of integral database refactoring mainly has 10 steps.

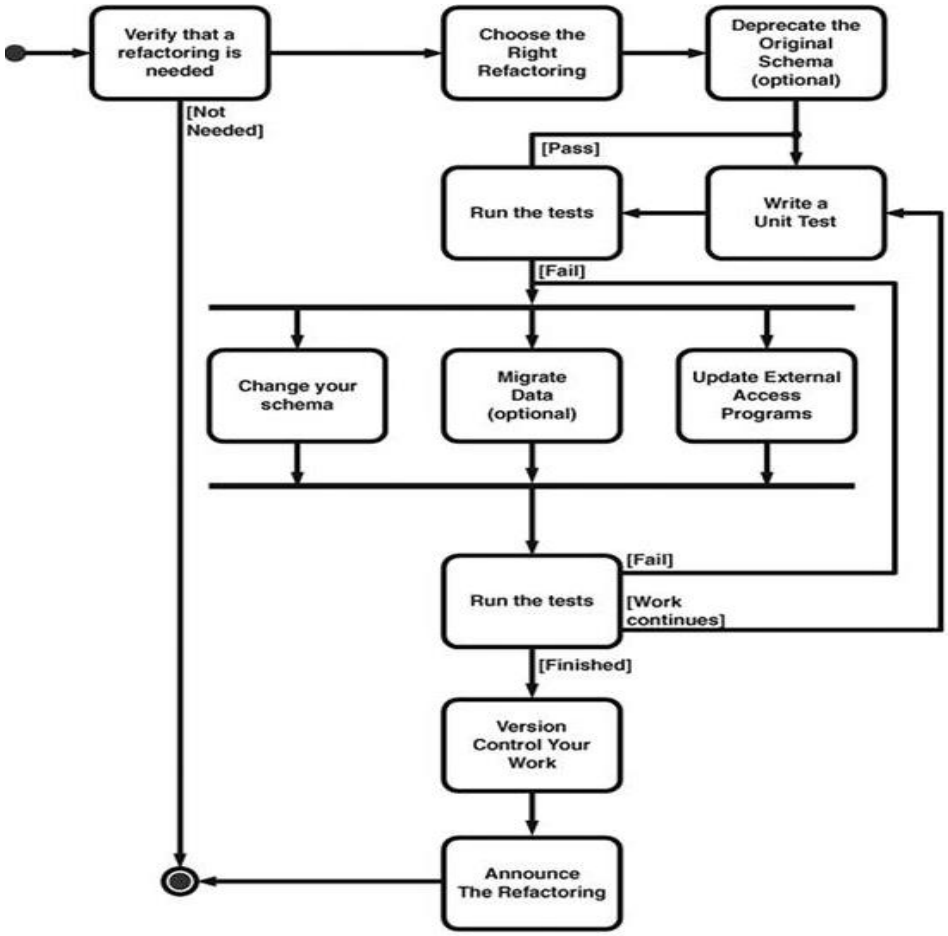


Figure 3 The process of database refactoring [Ambler et al., 2006]

1. **Verify that a database refactoring is appropriate.** Firstly, the database administer (DBA) needs to decide whether the suggested refactoring should be done or not. There are three questions to be taken into account at the verification phase, as discussed below.

Does the refactoring make sense? Perhaps the existing table is correct, but because the developer disagrees with it or just misunderstands it, this misunderstanding probably makes the developer believe it should be changed when it really does not. This requires the DBA to have a good knowledge of the project team's database and other corporate databases, to know who may have this issue. Therefore, the DBA is in a better position to decide whether the existing schema needs to be refactored. Furthermore, the DBA often stands at the bigger picture of the overall enterprise, providing important insight that may not be apparent when the developer looks at it from the point of view of the single project.

Is the change really needed now? This actually depends on the DBA's experiences. He has to communicate with application developer. Does the developer have a good reason for making the schema change? Can the developer explain how the change supports the business

requirements? Does the requirement feel right? With this assessment, the DBA may suggest the developer to refactor or not.

Is it worth the effort? The DBA should analyze the overall effects of the refactoring. To do this, the DBA should know how the external applications are coupled with this part of the database. When the DBA is not sure about the impact, he needs to advise the application developer to wait until he finds out the impact. His goal is to ensure that the database refactoring implementation will succeed. If hundreds of applications need to be updated, tested and redeployed to support this change, it is probably not viable for him to continue. He needs to analyze the business value of the refactoring to see whether it is enough to support the refactoring.

2. **Choose the most Appropriate Database Refactoring.** There are many categories of refactoring that can work on the database schema. To determine which one is most suitable for the database, the DBA has to analyze and understand the problem he faces. For example, there is an Employer table. The developer thinks this table should store the employer's income, therefore, he suggests the DBA to add a new column. However, what he does not know is that there actually is a column storing it but this column is in the Department table, which is arguably wrong place for it to be. The developer has identified the right problem but misidentified the solution. Based on the knowledge of the existing database schema and the understanding of the problem, the DBA suggests applying a Move Column refactoring.

3. **Deprecate the Original Database Schema.** If there are many external programs based on this database, there should be a transition period, also called "a deprecation period", as shown in Figure 4 [Ambler, 2005]. In this period, the original schema and the new one should both support the programs to provide time for those application teams to refactor and deploy their systems. Usually, the transition period will lasts for years, at least mouths, so automatically refactoring is the best method for reducing the risk caused by human resource changes.

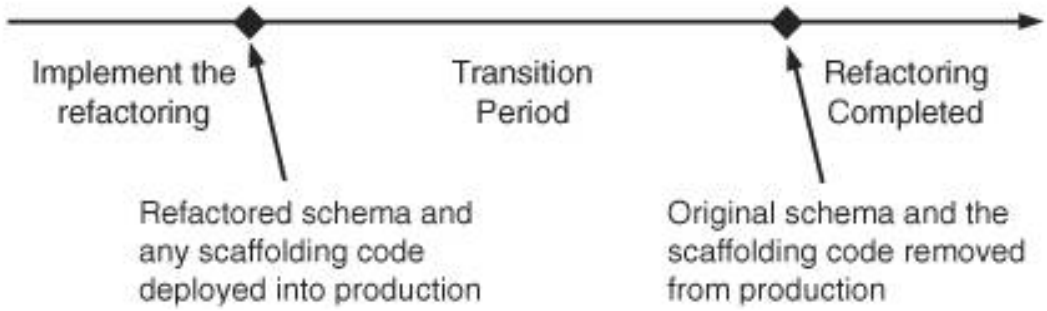


Figure 4 The life cycle of a database refactoring in a multi-application scenario

4. **Test before, During, and After.** The best way to change the database schema is using the Test-Driven Development (TDD) approach. With the TDD-based approach, the developer writes a test script before writing the code. Then he continues doing this until the refactoring is completed. The test mainly include aspects such as testing the database schema, testing the way the application uses the database schema, validating the data migration, and testing the external program code.

5. **Modify the Database Schema.** Small, focused scripts are easier to maintain. Applying each refactoring in an appropriate order to the database schema will make the change more stable, because some refactoring probably depend on the previous one, such as quoting the name of the new table. Version control is also an important method; different database instances will have different versions to make every change clearly.

6. **Migrate the Source Data.** Many database refactorings require operations on the data in some ways. Sometimes, the data should be moved from one place to another. For example, the Customer.Balance column needs to migrate to Account table.

7. **Refactor External Access Application.** When the database schema has changed, the external applications should be refactored if the access the changed part. For example, if an external application used to call the Customer.Balance column in the Customer table to get the Balance value, then after the refactoring, this column migrates to Account table. So the code of the external application should change this part to access Account.Balance.

8. **Run the Regression Tests.** Part of achieving the refactoring is testing. To make sure it works, the developer usually needs to keep testing and changing until the refactoring is done. Automatic testing is always the best choice. Because refactoring causes small changes, when the test fails, the developer can clearly know that the problem is from the last change.

9. **Version Control the Work.** When database refactoring is successfully operated, the work should be put under configuration management (CM).

10. **Announce the Refactoring.** A database is a shared resource. So after a refactoring, DAB should inform co-workers that the refactoring has been done and relevant documentation has been updated. This is because other teams need to know how the database schema evolves.

3.2.5 Categories of Database refactoring

There are six kinds of database refactoring [Ambler et al., 2006].

Structure Refactoring. Make changes to the definition of one or more tables or views. For example, move a column from one table to another or splitting a multipurpose column into several separate columns, each for one purpose.

Data Quality Refactoring. A change that improves the quality of the information contained in a database. For example, make a column "non-nullable" to ensure that it always contains a value or apply a common format to a column to ensure consistency.

Integrity Refactoring. A change that ensures there is a referenced row existing in another table, and ensures the row that is no longer needed is removed appropriately. For example, add a trigger to enable a cascading delete between two entities. This code was implemented outside of the database before.

Architectural Refactoring. An overall manners change to an interactive mode between external programs and the database. For instance, replace an existing Java operation in a shared code library with a stored procedure in the database. Make it as a stored procedure so that it can be accessed by non-Java applications.

Method Refactoring. Change the method (stored procedure, stored function or trigger) to improve its quality. Many code refactorings are applicable to database methods. For example, rename a stored procedure to make it easier to understand.

Non-Refactoring Transformation. They do not belong to refactorings, because they change the database schema and its semantics. For example, add a new column to an existing table.

Beside these categories, actually, the structure refactoring and data quality refactoring are enough to solve all the problems of database smells as represented before. The integrity refactoring, architectural refactoring and method refactoring are more like a kind of supplement to help developers access the database more easily by using the stored procedures.

4. The Entity-Relationship Model and its Meta-Model

Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database [Teorey et al, 2011]. Database design incorporates five phases, i.e. requirements analysis, conceptual design, logical design, physical design, and normalization [Hernandez, 2003]. Among that, it takes the most time and intelligence in the conceptual phase during the database design. Meanwhile, it is also the most critical phase in the database design process.

A conceptual design phase is to abstract the brief descriptions of user's data requirements, which are developed in the requirements analysis phase. The modeler can use the conceptual model to create an abstract representation of the situation under investigation [Thalheim, 2012], to make the representation more accurate and uniform. A conceptual data model represents the orients objective world and users. It is a bridge between things in the real world and the data in the system, and it is the first abstraction from the real world to the information world. A conceptual data model is independent of the software system, and does not involve any physical implementation. It emphasizes what the data describes, not how. It also focuses on the logic of the data description.

A conceptual data model comprises a number of concepts and their relationships for database design [Thalheim, 2011]. It does not involve any physical implementation, and makes the designer to focus on the semantic meaning and relationships among the identified data constructs. Meanwhile, it is easy to understand which eases the communication between designers and users. A conceptual data model can be further converted to the logical model, such as the relational model [CODD, 1970].

On the one hand, a conceptual data model should have strong semantic representation ability, to represent every semantic concept of the system conveniently and directly. On the other hand, it should be simple, clear, and easy to understand and communicate.

4.1. The ER Model and its Meta-Model

There are many conceptual data model formalisms, among which the ER model is the most commonly used one. The ER model adopts the more natural view that the real world consists of entities and relationships. It incorporates some of the important semantic information

about the real world [Chen, 1976], and uses graph to represent the information. It completely reflects the two features of conceptual data model. Because of its intuitive and simple structure by using graph, it has become the most popular data modeling tool in the conceptual abstract level.

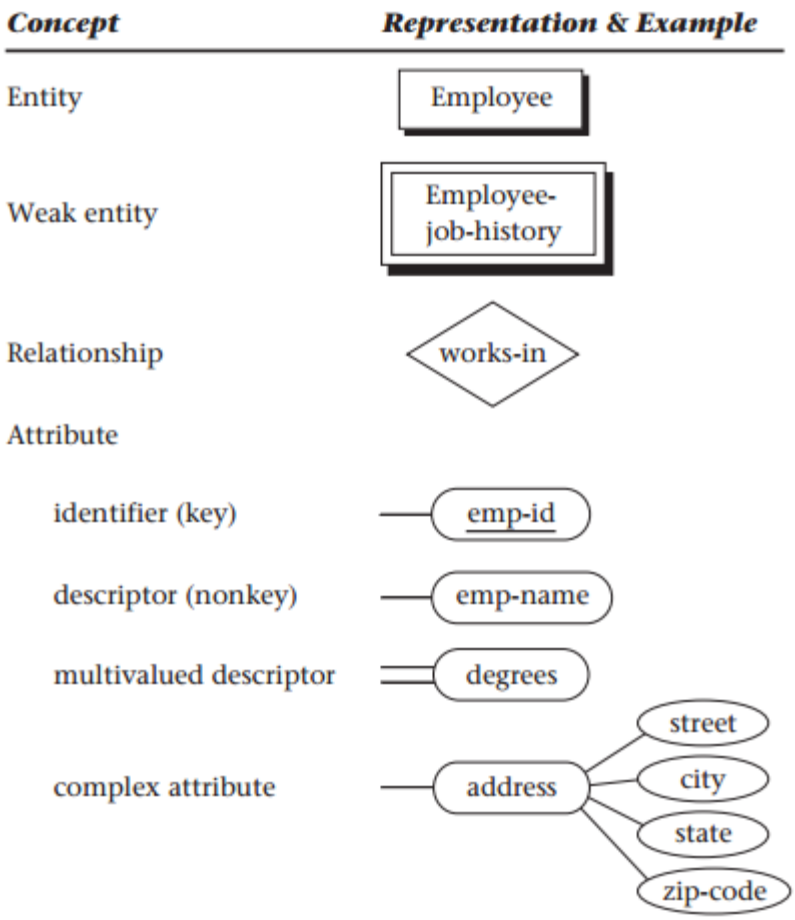


Figure 5 The basic ER model [Teorey et al, 2011]

An ER model consists of entities, relationships and attributes. Figure 5 shows the basic graphical constructs of an ER model.

An entity is a "thing" which can be distinctly identified [Chen, 1976]. It could be a physical object, such as a person, a company, or a conceptual object like birthday. Entities are classified into different entity sets such as Employee and Department. There is a predicate associated with each entity set to test whether an entity belongs to it [Chen, 1976]. For example, if we know an entity is in the entity set Department, and then we know that it has the properties common to the other entities in the same entity set. The entity construct is a rectangle as depicted in Figure 5. The entity name is written inside the rectangle.

There are two entity types, strong entity and weak entity. Strong entity also called Independent entity. It means every entity in this entity set can identify itself, does not need to depend on other entities in other entity sets. Weak entity is also called dependent entity. It means every entity in this entity set depends on another entity in other entity set to identify itself. Strong entities have internal identifiers that uniquely determine the existence of entity instances, but weak entities derive their identity from the identifying attributes of one or more "parent" entities. Weak entities are often depicted with a double-bordered rectangle (see Figure 5), which denotes that all occurrences of that entity depend on an associated strong entity for their existence in the database. For example, in Figure 5, the weak entity Employee-job-history is related to the entity Employee and dependent upon Employee for its own existence.

A relationship is an association among entities [Chen, 1976]. In the real world, there are connections in the interior of the thing or between things. They become the relationship in an entity set or between different entity sets when these connections map to the information world. For example, a "marriage" relationship describes the connection between two entities in the Person entity set, and a "works_in" relationship represents the connection between entities in the Employee entity set and the ones in the Department entity set.

There are two types of relationships, i.e. a specific relationship and a non-specific relationship. A specific relationship clearly defines how many entities in an entity set are related to entities in another entity set. A non-specific relationship, also called a many-to-many relationship, means every entity in any entity sets in the relationship has the possibility to relate to 0, 1 or many entities in another entity set. The non-specific relationships often exist in the beginning of the conceptual modeling process. They help to represent the map of entities in the real world. When the modelling process goes on, a non-specific relationship is replaced with a specific one, and representing the logical model for a physical database implementation.

Attributes are characteristics of entities that provide descriptive detail about them. A particular occurrence of an attribute within an entity or relationship is called an attribute value [Chen, 1976]. For example, "Peter", "blue", "20" are values. They are classified into different value sets, such as Name, Color and Age. The names of the value sets are attributes. The attribute construct is an ellipse with the attribute name inside (or an oblong, as shown in Figure 5). The attribute is connected to the entity it characterizes.

The ER model is widely used in industry for database design. Also, various researchers have published extensions to the ER model and have also innovated in the graphical representation

of the constructs in the ER model. The extensions to the original ER model which have gained widespread acceptance incorporate the notions of (a) specialization/generalization and (b) subclass/superclass. In this thesis, most of the examples and the implementation use basic concepts in the ER model, but the approach is equally applicable to the enhanced entity relationship (EER) model.

4.2.1 Meta-Model

Modeling languages are used to create models that aim to raise the level of abstraction and hide implementation details, and the abstract syntax of a modeling language is formalized by means of a meta-model, which also serves as a basis to interchange models with other tools [Fidalgo et al., 2013]. The specification of a meta-model is one level higher than normal models, in other words, a meta-model is the model of a model.

The graphical representation of an EER Model is a node-link diagram [Irani et al., 2001; Ware and Bobrow, 2004]. The nodes represent the elements in the model, i.e. Entities, Associative Entities, Attributes, Relationships, Inheritances, and Categories. The nodes include rectangle, diamond, ellipse, etc. For example, rectangle represents entities, and diamond represents relationship. The links in the model are the lines to connect different elements. They include Attribute Link, Relationship Link, Generalization Link, and Specialization Link, as shown in Figure 6. For example, if an entity has an attribute called entityName, in the EER Model, there will be a line connect the entity (in the rectangle) and the attribute (in the ellipse), this line is called Attribute Link.

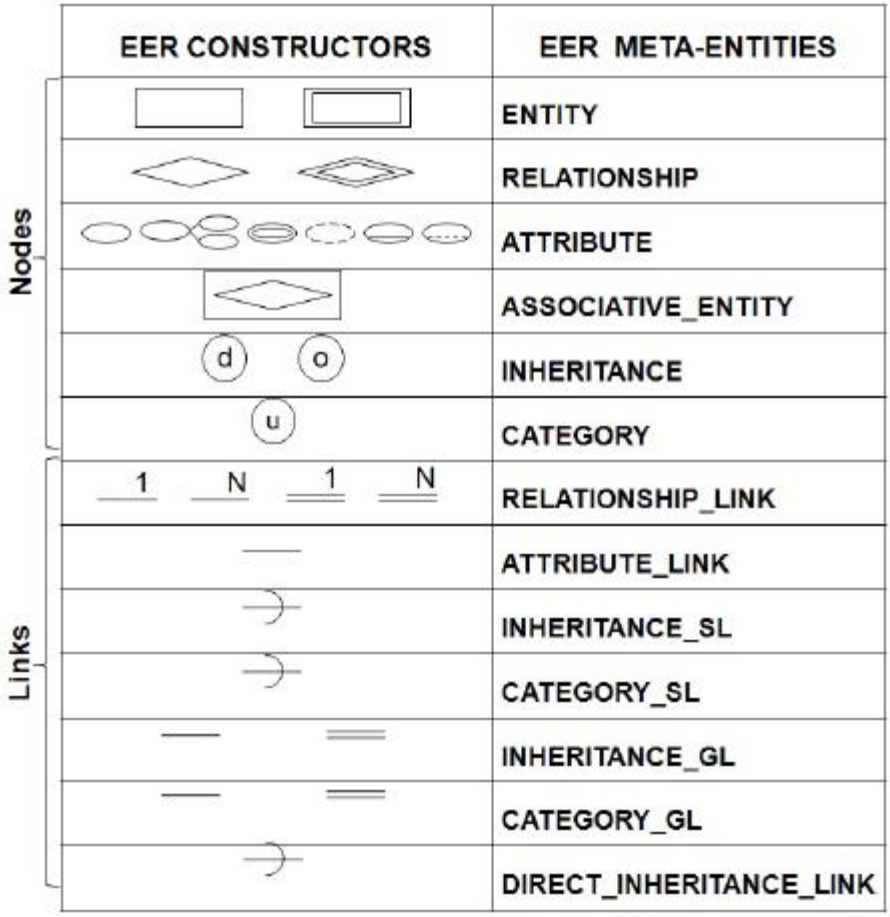


Figure 6 EER meta-entities and constructors

Using the node-link abstraction and the EER notation, a graphical representation of the meta-model named enhanced entity relationship meta-model (EERMM) proposed by Fidalgo et al is shown in Figure 7.

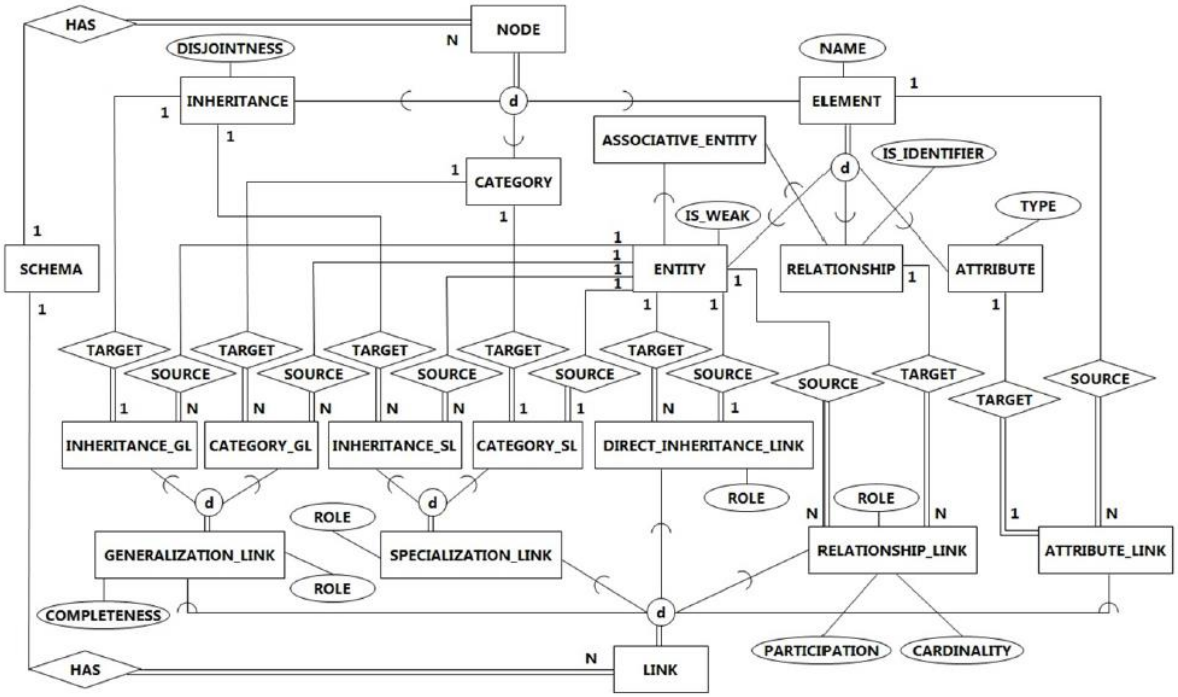


Figure 7 Conceptual view of Enhanced ER Meta-Model [Fidalgo et al., 2012]

4.2.2 Meta-entities

The meta-model has three main meta-entities: Schema, Node and Link. Schema is the root meta-entity, and it corresponds to the drawing area of the EER schema. So it can have many instances of Node and Link. Node and Link cannot exist without Schema.

There are three specialized meta-entities for the Node, i.e. Element, Category and Inheritance. The Element meta-entity has three specialized meta-entities, i.e. Entity, Relationship and Attribute. Besides, there is an Associative_Entity, and it is a specialization of the Entity and Relationship meta-entities. The ER model does not offer direct support to many-to-many relationships, even though such relationships happen frequently in normal usage. The solution to this problem is the creation of another table to hold the necessary information for this relationship. This new table is called an associative entity table. The Inheritance meta-entity captures the "inheritance" concept and the Category meta-entity captures the "category" concept [Fidalgo et al., 2012]. Inheritance represents the inheritance relationship between superclass and subclass. Category represents a collection of instances that is a subset of (partial Category) or the union of (total Category) distinct entities.

This meta-model has five specialized meta-entities for the Link, i.e. Attribute_Link, Relationship_Link, Specialization_Link, Generalization_Link and Direct_Inheritance_Link. They address the links for an attribute, a relationship, a specialization, a generalization and a

direct inheritance [Fidalgo et al., 2013]. Theses meta-entities are explained as follow.

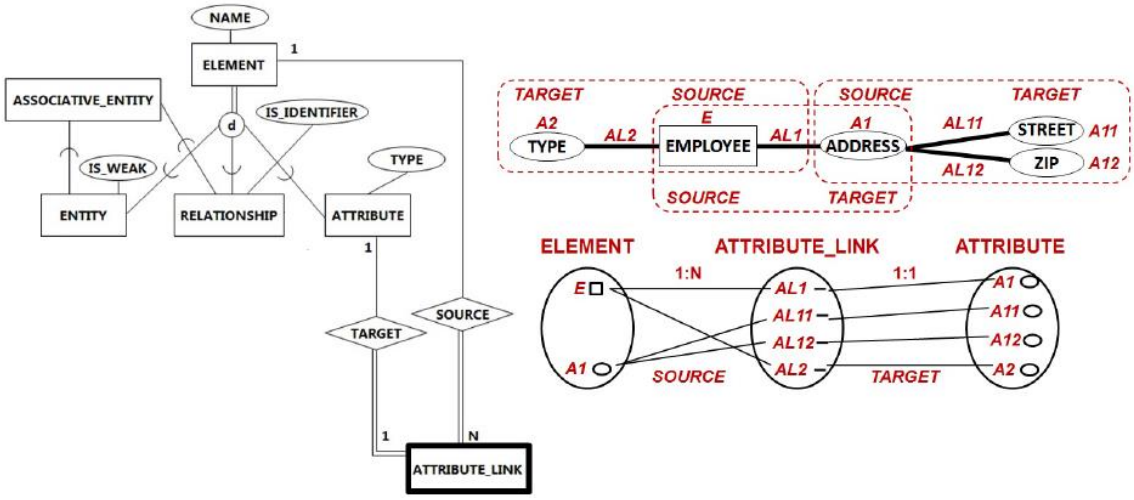


Figure 8 An example of Attribute_Link [Fidalgo et al., 2013]

As shown in Figure 8, the Attribute_Link meta-entity has a Source relationship with the Element meta-entity. The meta-model must reflect the fact that in the ER model it is not just entities that can have attrbutes: a relationship and even an attribute can have an attribute. This latter case allows the construction of complex data items, such as a street address. An instance of Attribute_Link has only one instance of Element as source, but an instance of Element can be source for many instances of Attribute_Link, like many attributes on entities, many attributes on relationships and many composite attributes. Besides, the Attribute_Link meta-entity has a Target relationship with the Attribute meta_entity. An instance of Attribute_Link has only one instance of Attribute as target and an instance of Attribute can only be target for one instance of Attribute_Link, an Attribute cannot be linked to more than one Element [Fidalgo et al., 2012]. For example, as represent in the right part of Figure 8, is one entity called EMPLOYEE and an attribute called ADDRESS. The ADDRESS is an attribute of EMPLOYEE. , and is connected to the EMPLOYEE through an Attribute_Link called AL_1. The EMPLOYEE has two attributes, i.e. TYPE and ADDRESS. Moreover, the ADDRESS has two attributes, i.e. STREET and ZIP. It is clear that every element can be source for more than one Attribute_Link, but every Attribute_Link targets to only one attribute.

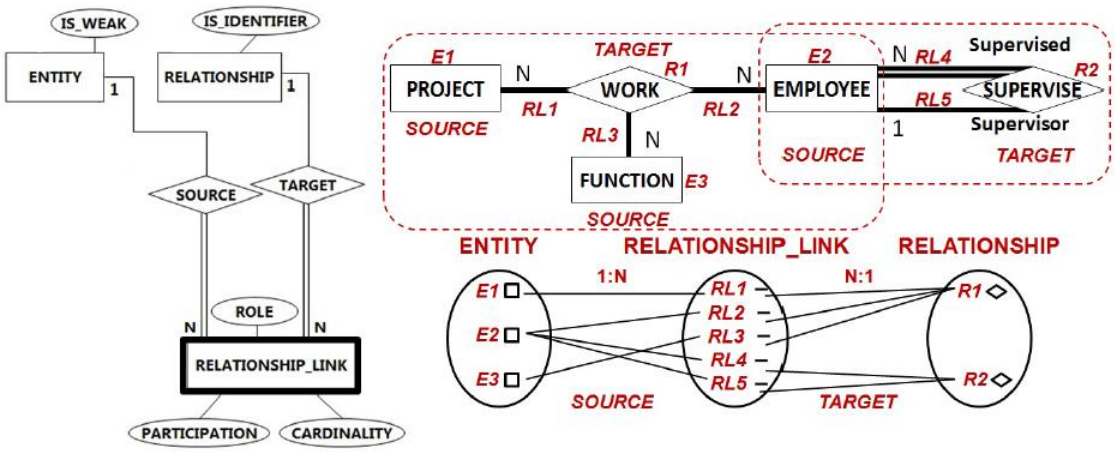


Figure 9 An example of Relationship_Link [Fidalgo et al., 2013]

As represented in Figure 9, the Relationship_Link meta-entity has a source relationship with Entity meta-entity. An instance of Relationship_Link has only one instance of Entity as source, but an instance of Entity can be the source for many instances of Relationship_Link. Besides that, the Relationship_Link meta-entity has a target relationship with Relationship meta-model. An instance of Relationship_Link has only one instance of Relationship as target, but an instance of Relationship can be target for many instances of Relationship_Link. Therefore, an instance of Relationship_Link connects one instance of Entity and one instance of Relationship, but an instance of Entity or Relationship can be connected to many instances of Relationship_Link. It is because one entity can be linked to many relationships, and a relationship can consist of many entities [Fidalgo et al., 2012]. The right part of Figure 9 is an example to prove this discussion.

As shown in Figure 7, there are also Specialization_Link, Generalization_Link and Direct_Inheritance_Link which contain the corresponding information, but they are not concerned in this thesis.

4.2.3 Meta-Attributes

Meta-Attribute is the attribute of meta-entity. As shown in Figure 7, the Element meta-entity has a meta-attribute called name which defines the name of an element. The Entity meta-entity has a meta-attribute called is_Weak, which has two values, “Yes” and “No”. The Relationship meta-entity has a meta-attribute called is_Identifier that is also a Boolean type. The Attribute meta-entity has a meta-attribute called type that specifies whether an instance of Attribute is “common”, “identifier”, “discriminator”, “derived” or “multivalued” [Fidalgo et al., 2012].

The Inheritance meta-entity has the meta-attribute called disjointness. It has two values: “disjoint” and “overlap”, which define whether an inheritance is mutually exclusive or not [Fidalgo et al., 2012].
The Relationship_Link meta-entity has three meta-attributes: participation, cardinality and role. Role is a text to describe the function of an entity in a relationship. Participation has two values, i.e. “total” and “partial”. “Partial” means that an instance of the related entity class can exist without being associated with an instance of the relationship link. “Total” means that every instance of the related entity class must be associated with an instance of the relationship link. Cardinality also has two values: “one” and “many”. “One” means that, for any given instance in the related entity class, that instance can be associated with at most one instance of the relationship link. “Many” means that, for any given instance of the related entity class, there is no limit on the number of instances of the associated relationship link with which the entity instance can be associated.

4.2. A Meta-Model database

On the basis of the meta-model proposed by Fidalgo et al., we have implemented the Enhanced Entity-Relationship Meta-Model (EERMM) database, which can store the conceptual model for a design. The ER Meta-model is implemented as an ordinary relational database, using the SQL Server 2012 DBMS. The implementation allows a conceptual model, as represented in an ER diagram, to be stored and updated.

The database implementation has a separate table for each of the meta-entities and meta-relationships, as shown in Figure 7. The names of the tables in the implementation are the same as the labels of the rectangles and diamonds in the diagram, prefixed with “tbl”, as shown in Figure 10. There are 20 tables generated from Fidalgo’s meta-model, among them, 11 tables represent the meta-entities, i.e. tblAssociativeEntity, tblAttribute, tblCategory, tblElement, tblEntity, tblFuntionalDependencies, tblInheritance, tblLink, tblNode, tblRelationship, tblSchema. Others represent the meta-relationships, i.e. tblAttributeLink, tblCategoryGL, tblCategorySL, tblDirectInheritanceLink, tblGeneralizationLink, tblInheritanceGL, tblInheritanceSL, tblRelationshipLink, tblSpecializationLink. The columns in these tables represent the meta-attributes.

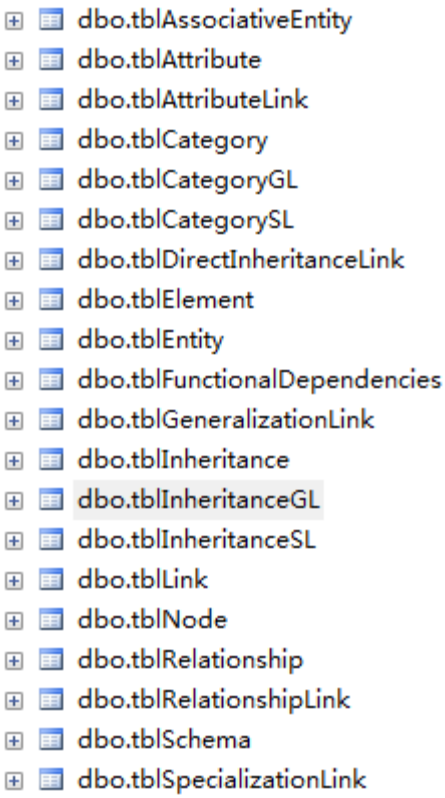


Figure 10 The EERMM database implementation

Each record in tblEntity corresponds to an Entity Class in the ER model. The columns represent the value of meta-attributes associated with the Entity. As shown in Figure 11, in this table, every entity has an entity_ID as its primary key attribute. As represented before, an entity is part of an element, so every entity also has an element_ID, this is a foreign key reference to tblElement class. EntityName and entityName_plural are the name values of an entity, in the situation of singular and plural. And the last meta-attribute is isWeak, which defines the entity is weak or not.

entity_ID	element_ID	entityName	entityName_plural	isWeak
1	1	Employee	Employees	Unknown
2	2	Department	Departments	Unknown

Figure 11 Example of tblEntity class

Each record in tblAttribute corresponds to an attribute in an Entity Class or a Relationship in the ER model. As shown in Figure 12, every attribute has an attribute_ID as its primary key attribute. And like tblEntity, attributes have an element_ID, too. The attributeName is the name of the attribute and identifierDiscriminatorOrdinary is a type of the attribute. The attribute has many types, and identifierDiscriminatorOrdinary is only one of them. These types will be represented in next chapter. The attributeLink_ID is a foreign key reference to tblAttributeLink. As shown in Figure 13, the tblAttributeLink class has an attributeLink_ID

as its primary key attribute, and two foreign keys, i.e. attribute_ID and element_ID. So from these foreign keys, the database can understand the connection between the attributes and entities. For example, the first attributeLink in Figure 13 has an attribute_ID 1, it corresponds to the attribute EmployeeNumber, and it also has an element_ID which corresponds to the entity Employee, so the database will understand EmployeeNumber is an attribute of entity Employee. And the attributeLinkName is the name of the attributeLink.

attribute_ID	element_ID	attributeLink_ID	attributeName	identifierDiscriminatorOrdinary
1	3	2	EmployeeNumber	Discriminator
2	4	4	DepartmentNumber	Identifier

Figure 12 An example of tblAttribute class

attributeLink_ID	attribute_ID	element_ID	attributeLinkName
2	1	1	Employee_EmployeeNumber
4	2	2	Department_DepartmentNumber

Figure 13 An example of tblAttributeLink class

Each record in tblRelationship corresponds to a relationship in the ER model. As shown in Figure 14 the relationship has a primary key attribute called relationship_ID and a foreign key element_ID. And the name of the relationship is defined in the attribute relationshipName. It also has two attributes, i.e. isIdentifier and degree, which contain the information of is the relationship is an identifier or not, and the degree of the relationship. Figure 15 shows the tblRelationship_Link class, this class has represented which entities are involved in the relationship. This class has a primary key attribute relationshipLink_ID and its name is defined in relationshipLinkName. RelationshipLink_ID is a foreign key reference to tblRelationship class. This class also has two attributes, participation and cardinality, the meaning of them has represented before. Role_sglr and role_plrl defines the role information, which in the situation of singular and plural. For example, in Figure 14, the first relationship is Employment, and in Figure 15, two entities are involved in the relationship according to the relationship_ID. They are Employee and Department. So the relationship is Employee works in Department, and Department employs Employee.

relationship_ID	element_ID	relationshipName	isIdentifier	degree
1	5	Employment	Unknown	binary
2	6	Dependency	Unknown	binary

Figure 14 An example of tblRelationship class

relationshipLink_ID	relationshipLinkName	entityName	relationship_ID	participation	cardinality	role_sglr	role_plrl
1	Employment_Employee_works in	Employee	1	Unknown	Unknown	works in	work in
2	Employment_Department_employs	Department	1	Unknown	Unknown	employs	employ
3	Dependency_Employee_has	Employee	2	Unknown	Unknown	has	have
4	Dependency_Department_belongs to	Department	2	Unknown	Unknown	belongs to	belong to

Figure 15 An example of tblRelationship_Link class

Our meta-model database helps the developer to manage his physical database. When the developer creates the physical database based on this meta-model database, he needs to insert the name of the database in a record in tblSchema, which implements the Schema class. He can create the physical database manually, or use the create script. When the developer updates values of the meta-model database, and that update will impact the physical database, the system will check the tblSchema class and get the name of the physical database, then dose the corresponding activities to the physical database. For example, here we create a database named EmpsAndDepts, the create script will insert EmpsAndDepts to tblSchema, as shown in Figure 16. The create script is given in Appendix f.

schema_ID	schemaName
1	EmpsAndDepts

Figure 16 Example of tblSchema class

For each table, there is a sequence associated with the primary key. When inserting rows into a table, the insert transaction does not need to provide a value for the primary key attribute as this is generated automatically.

We need stored procedures which can insert new entities, relationships, attributes and generalization hierarchies into the design. The stored procedures code is given in Appendix b.

5. Extending the EER Meta-Model: Meta-Attributes with Value "Unknown"

5.1. Incompleteness in a conceptual model

In the software development process, it is impossible to elicit all the requirements in the beginning. Because it is just too hard to predict all the details required by a project, only small portion of requirements are specified before iterations start [Zhang et al., 2014]. And in the requirements phase, unresolved decisions about needs, incomplete understanding of the problem domain and disagreements among stakeholders all produce uncertainty [Salay et al., 2013].

Models represent the state of comprehension or knowledge of a user [Thalheim, 2012]. When those uncertainties remain from the requirements phase to the modeling phase, the modeler has to face the situation that (s)he needs to construct models with the incomplete requirements. In the incremental development methodologies such as agile software development methods, the analysis, modeling and design activities occur along with the development activities. So the modeler also needs to consider how the working database should be designed with incomplete requirements specification.

To discuss how the missing requirements impact the modelling and the follow-up software development activities, we shall clarify what are the known requirements and what are the unknown ones, i.e. missing requirements.

There are four states of knowledge at any given time point in a project. They are known-known (KK), known-unknown (KU), unknown-known (UK) and unknown-unknown (UU) [Zhang et al., 2014].

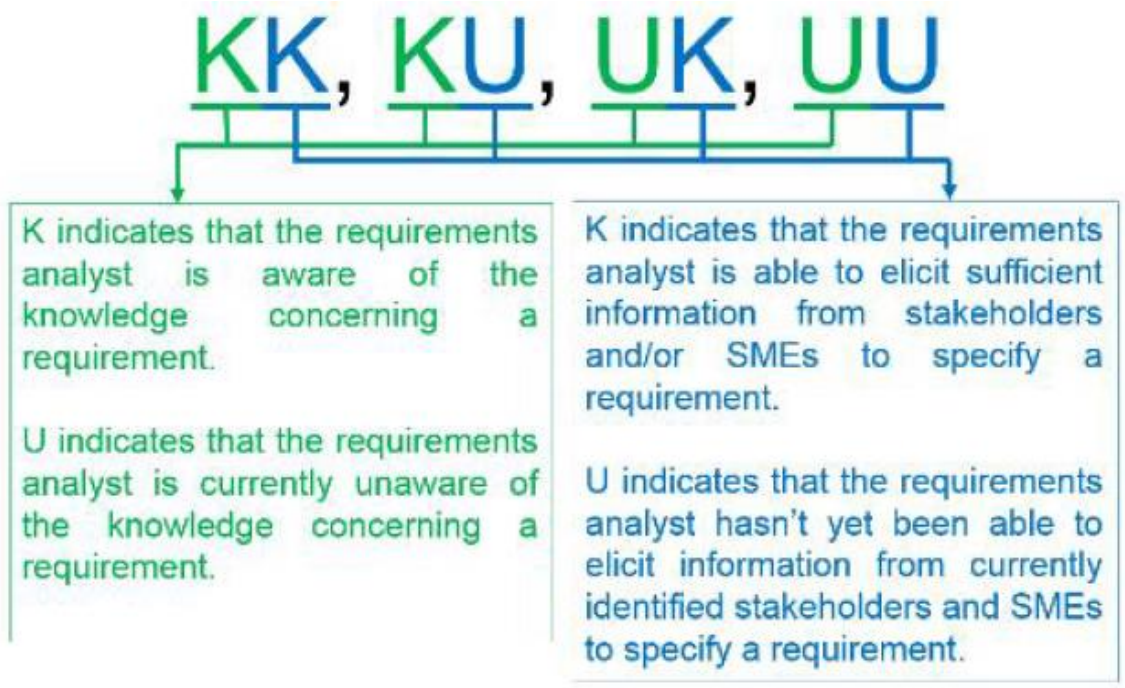


Figure 17 States of requirements analyst's knowledge concerning requirements at any given time [Zhang et al., 2014]

As shown in Figure 17, the left-hand K or U indicates the requirements analyst's awareness of the knowledge concerning a requirement. The right-hand K or U indicates whether or not the needed knowledge is possessed by stakeholders and/or Subject Matter Experts (SMEs) at a given point in time [Zhang et al., 2014].

KK means the requirement analyst is aware of the knowledge concerning a requirement and he/she is able to elicit from the stakeholders and/ or SMEs. KK is explicit and clear requirement, which will be implemented in the subsequent development.
KU means the requirement analyst is aware of the knowledge concerning a requirement but is not able to elicit from the stakeholders for now. In other words, there are difficulties to obstruct the analyst to obtain the required knowledge, but he/ she is already knows there is a missing requirement. The simplest form of a KU can be a missing business rule [Wan-Kadir and Loucopoulos, 2004], which likely exists only in the heads of individuals [Wiegers, 2003].

UK refers to the knowledge that can be elicited from the stakeholders and/ or SMEs, but the analyst is unaware at the time. It is the knowledge which the stakeholders and or SMEs can provide, but has not been discussed with the requirement analyst. Unless the analyst realizes this knowledge is valuable, it can be elicited. Then it changes to KK.

UU refers to the knowledge which the analyst is unaware of, and is not possessed by the currently identified stakeholders and SMEs [Zhang et al., 2014]. As the software system

becomes increasingly complex and faces constant changes, it is impossible to predict and specify the large body of knowledge about the system and complex environment where the system operates [Cheng and Atlee, 2007]. A UU is unknowable [Gomory, 1995], unpredictable, and emerges during the project life cycle.

In this thesis, we focus on how to implement a database with known-unknown information, and how to automatically refactor the database when known-unknowns transform to known-knowns. In the meta-model, there is no support for explicitly indicating the known-unknown information in an ER model. In order to record the known-unknowns, the EERMM database uses "Unknown" value. It makes the requirements analyst aware of the missing information, and allows the modeler to take the missing information into account when he converts the conceptual model to a physical database design.

Suppose there is a known-unknown about the cardinality of a relationship. For a concrete example, there is a relationship Works-In connecting two entity classes, i.e. Employee and Department. Obviously a Department can have more than one Employee, but the requirements analyst must discover whether there is a requirement to implement this relationship to allow an Employee to work in more than one Department. Until this known-unknown is resolved, the provisional implementation of the relationship is Many-to-Many. Many-to-Many relationships are harder to implement, less efficient and harder to maintain than Many-to-One relationships. The Many-to-Many relationship requires a so-called "Bridge Table" to connect the two entity classes. Hence they should be avoided wherever possible.

This chapter will describe all of the classes of meta-attributes for which it is possible to assign an "Unknown" value to a particular instance of the meta-attribute. The discussion focuses on the tblEntity class, tblRelationship class, tblAttribute class and tblRelationship_Link class.

5.2. Entity

In the tblEntity class, the meta-attribute "isWeak" can have value "Yes", "No" and "Unknown". When the value of "isWeak" is set as "No", it means this entity class is a normal entity class; when the value is set as "Yes", it is a weak entity class; and when "isWeak" is set as "Unknown" means that this information is a known-unknown.

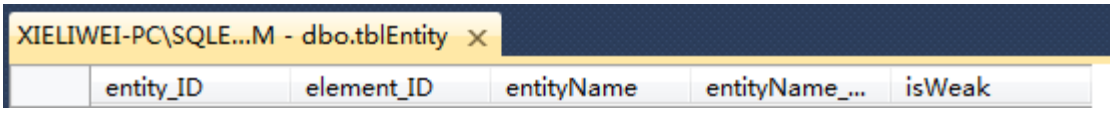


Figure 18 The tblEntity class

A weak entity class has a primary key which includes a foreign key reference to the owning entity class (also known as the identifying entity class), which is related to the weak entity class via an identifying relationship. A weak entity class always has a total participation constraint with the identifying relationship. A weak entity class normally has a partial key which can identify between the various weak entities instances associated with a particular identifying entity instance.

5.3. Relationship

In the tblRelationship class, the meta-attribute "isIdentifier" can have value "True", "False" and "Unknown". When "isIdentifier" is set as "True" it means that the relationship is an identifying relationship; when "isIdentifier" is set as "False" it means that the relationship is a normal relationship; and When "isIdentifier" is set as "Unknown" it means that the information of this is an identifying or normal relationship is a known-unknown.

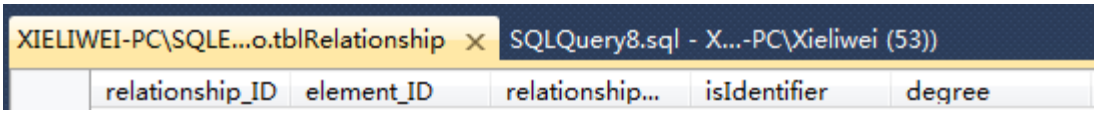


Figure 19 The tblRelationship class

A normal relationship is that all the entities associated in the relationship are strong entities. An identifying relationship is that one of the entity classes which are linked to the relationship is a weak entity class. And the identifying relationship can have a degree greater than two. The degree of a relationship is the number of entities associated in the relationship. Binary and ternary relationships are special cases where the degrees are 2 and 3, respectively. An n-ary relationship is the general form for any degree n. The notation for degree is illustrated in Figure 20. The binary relationship, an association between two entities, is by far the most common type in the natural world. In fact, our system uses only this type. So the value of degree in this class will always be binary.

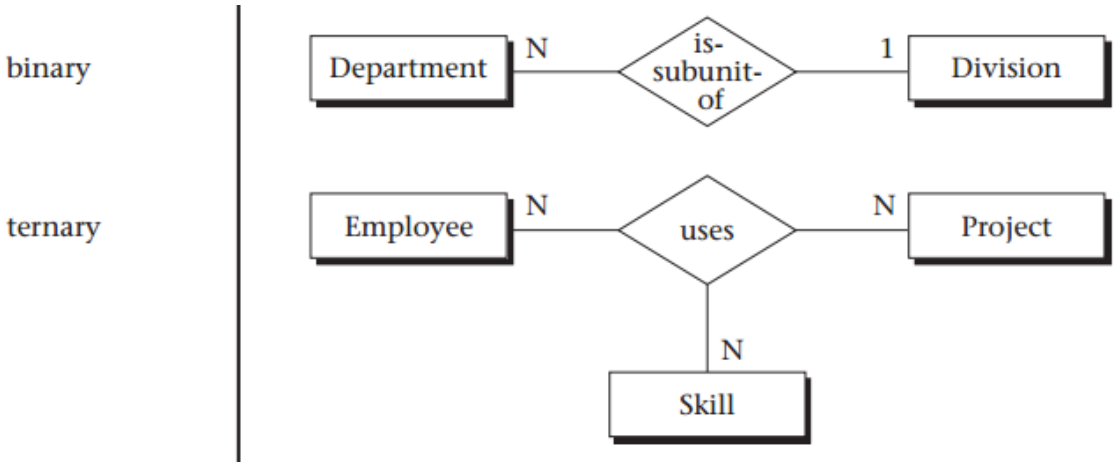


Figure 20 The example of binary and ternary relationship

5.4. Attribute

There is one problem that Fidalgo et al. define a single meta-attribute "Type" for the Attribute meta-entity, but that single meta-attribute has several different components, i.e. single-valued versus multi-valued, simple versus composite, identifier versus discriminator versus ordinary, etc. It is easier for us to implement the meta-model if we treat each of those components as separated meta-attributes. So there are 5 categories of known-unknown associated with an attribute in our database. They are "identifierDiscriminatorOrdinary", 'simpleOrComposite", 'singleOrMultivalued", 'storedOrDerived" and "Nullable".

attribute_ID	element_ID	attributeLink_ID	attributeName	identifierDiscriminatorOrdinary	simpleOrComposite	singleOrMultivalued	storedOrDerived	Nullable

Figure 21 The tblAttribute class

Meta-attribute "identifierDiscriminatorOrdinary" has four values, i.e. "Identifier", "Discriminator", "Ordinary" and "Unknown". An identifier means that the attribute's value uniquely identifies the entity class's entity instances. In other words, it is a key attribute. A discriminator can only occur in a weak entity class. It is a partial key which must be combined with an identifier from the owning entity class in order to form a key which uniquely identifies the individual entity instances in the weak entity class. An "ordinary" attribute is neither an identifier nor a discriminator. "Unknown" is a known-unknown information. And when the value of "identifierDiscriminatorOrdnary" is "Unknown", the system will treat the entity which has this attribute as a strong entity.

Meta-attribute 'simpleOrComposite" has three values, i.e. 'simple", "Composite" and "Unknown". A composite attribute can be divided into smaller sub-parts, which represent

more basic attributes with independent meanings. For example, The Address composite attribute can be sub-divided into Street, City and Postal Code. A simple attribute is not divisible. Composite attributes can form a hierarchy [Elmasri and Navathe, 2010]. For example, Address can be subdivided into three simple attributes, Number, Street, and ApartmentNumber.

Meta-attribute ‘singleOrMultivalued” has three values, i.e. ‘single”, “Multivalued” and “Unknown”. Most attributes have the ‘single” value for a particular entity. Such as an Age is an attribute with a single value of a Person. A “multivalued” attribute may have lower and upper bounds on the number of values allowed for each individual entity [Elmasri and Navathe, 2010]. For example, a car may have several colors, so the Color attribute is a multivalued attribute.

Meta-attribute ‘storedOrDerived” has three values, i.e. ‘stored”, “Derived” and “Unknown”. Sometimes two or more values are related, such as Age and BirthDate. “Derived” value in the table can be computed at the moment the query is posed like Age, the value of which is computed by subtracting the BirthDate from date at the time that the query is evaluated. BirthDate is a ‘stored” attribute.

Meta-attribute “Nullable” has three values, i.e. “Yes”, “No” and “Unknown”. It is about whether or not the attribute value is allowed to be a NULL. For example, suppose that “Nullable” takes the value “Unknown” for an attribute column that contains foreign key references to some other entity class. If the Participation meta-attribute of the RelationshipLink is assigned the value ““Total” then we need to update the value of “Nullable” to “No”. If, however, Participation is assigned the value “Partial” then “Nullable” is updated to the value “Yes”. “Unknown” means it is a known-unknown, but normally when the “Nullable” is set as “Unknown”, it is default that there could be NULL values for the attribute.

5.5. Relationship_Link

In the tblRelationship_Link class, there are three meta-attributes can have “Unknown” value and they are “Participation”, “Cardinality” and “isIdentifier”. “Role_sglr” and “role_plrl” are the role values of the relationship in singular and plural situation, so they will have determined value. And “isIdentifier” has the same value as the “isIdentifier” in tblRelationship class, which the relationship this relationship_Link associated in.

Figure 22 The tblRelationship_Link class

The participation meta-attribute can take one of three values: "Partial", "Total" and "Unknown". "Partial" means that an instance of the related entity class can exist without being associated with an instance of the relationship link. "Total" means that every instance of the related entity class must be associated with an instance of the relationship link. "Unknown" means that the requirements analyst has not yet been able to ascertain whether participation is partial or total.

The cardinality meta-attribute can take one of three values: "One", "Many" and "Unknown". "One" means that, for any given instance in the related entity class, that instance can be associated with at most one instance of the relationship link. "Many" means that, for any given instance of the related entity class, there is no limit on the number of instances of the associated relationship link with which the entity instance can be associated. "Unknown" means that the requirements analyst has not yet been able to ascertain whether cardinality is "One" or "Many"

6. Updating the Meta-Model Database and the Database Design

6.1. Database refactoring

Throughout this thesis, there is an assumption that the conceptual model must be constructed and modelers and developers are aware of all of the missing information in requirements. Hence the extended meta-model database can contain the value "Unknown" in several meta-attributes associated with any of the meta-entities in the meta-model for the database. In subsequent increments, when requirements analysts elicit the needed requirements from stakeholders, the modeler can update the conceptual model by replacing the "Unknown" value of a particular meta-attribute instance with a known one. In addition, the meta-model database and the physical design of the target database shall be updated accordingly.

Unsurprisingly, simply updating the meta-attribute values in the conceptual model and the follow-up database design can be done automatically and easily. When there are many types of meta-attributes in a conceptual data model, the update to the conceptual model can be used to refactor the physical database design of the target database and even to restructure the data according to the new design in the concrete database.

Although the approach described in this thesis is general-purpose, in the sense that it can be applied to any database design, the current implementation requires strict adherence to naming practices. In our code, all primary keys are single-column fields of data type Integer. The name of a primary key is the same as the name of the table, but with "key" prefixed to the table name. For example, for the Employee table, the primary key is "keyEmployee". The name of a bridge table comprises the name of the relationship concatenated to the string "_Bridge". Thus the Employment relationship is associated with the bridge table Employment_Bridge. There is nothing in the approach which requires this strict naming practice, but it helps to keep the code much simpler. We also point out that the methodology described in this thesis is for use during development. Once development is complete, the database administrator is free to make cosmetic changes to the table definitions. Furthermore, it is strongly advocated that updates to the target database should be made through stored procedures, rather than directly to the tables themselves. Thus it is possible to shield the applications software developer from the strict naming conventions and also from changes in the physical design of the target database which are made as a result of refactoring when meta-attribute values change from "Unknown" to a known value. So we also implemented the stored procedures for those updates. In this thesis, all the activities of creating table and

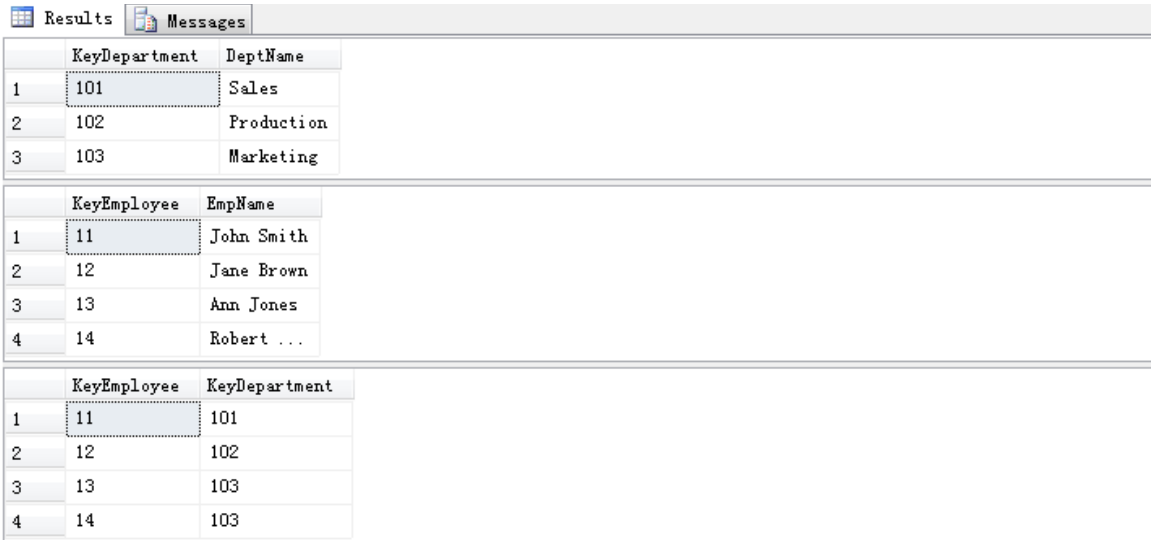
updating database are using the T-SQL script to achieve, and those scripts are in Appendix c and e.

During the process of refactoring database by using our approach, we partially follow the integral database refactoring process model. (1) In our approach, the database developer records the known-unknowns as "Unknown" in the meta-model database, and all these known-unknowns must be solved, so the refactoring is appropriate. (2) We choose the most appropriate database refactoring methods in the triggers. (3) In our case, we only focus on the database, and there is no external application, so the original database schema will be deprecated immediately. But in the real cases, the developer who uses this approach to refactor his database still needs the deprecation period. (4) We do the test whenever the updates happen to make sure our approach running correct. (5) We use the triggers and stored procedures to modify the database schema. (6) When the source data needs to be migrated, the corresponding triggers will run automatically. (7) The same reason as step 3, we only focus on the database. But in the real cases, the external application developers need to refactor their applications. (8) We have implemented the regression test script, which are put in Appendix b. It is the same script that also can insert the elements to the EERMM database. (9) In the real cases, the developer needs version control. (10) The same as step 9, the developer needs to announce the refactoring in the real cases.

6.2. Update the meta-attribute's values

This chapter presents some examples to illustrate how one change to the conceptual model can cause the physical database refactoring, i.e."cardinality" and "participation", or cause other changes to the conceptual model, i.e. "isWeak".

As an example, we implemented a database called EmpsAndDepts, the implementation code is in Appendix f. As shown in Figure 23, there are three tables in this database, and they are Department, Employee and Employment_Bridge.



Results Messages

	KeyDepartment	DeptName
1	101	Sales
2	102	Production
3	103	Marketing

	KeyEmployee	EmpName
1	11	John Smith
2	12	Jane Brown
3	13	Ann Jones
4	14	Robert ...

	KeyEmployee	KeyDepartment
1	11	101
2	12	102
3	13	103
4	14	103

Figure 23 Department Table (top), Employee Table (middle), Employment_Bridge Table (bottom)

6.2.1 Cardinality

At the beginning, the value of "cardinality" is "Unknown". We assume that an agile development methodology is used. This means that development proceeds in time-boxed increments. Thus a provisional database design, incorporating the Many-to-Many relationship, will already be in use by the development team. So we require the bridge table to model the relationship between the pair of entity classes Department and Employee.

Now suppose that the requirements analyst discovers that there is no requirement for a Many-to-Many relationship in the proposed application because there is a business rule that an Employee can only work in one Department at a time.

To update this, we shall modify the cardinality value from "Unknown" to "One" in the table "tblRelationshipLink" in the EERMM database. When we open it we can see that there is a relationshipLinkName called "Employment_Employee_works in", and as shown in the Figure 24, now the cardinality here is set as "Unknown".

relationshipLi...	relationshipLinkName	link_ID	relationship_ID	entityName	participation	cardinality
1	Employment_Employee_works in	3	1	Employee	Unknown	Unknown

Figure 24 The relationship_Link table before update

The developer needs to update the meta-model database, replacing the current value of "Unknown" in the relationship link between the employee entity class and the Works-In

relationship with the new value “One”. Here we use the T-SQL script to update the value. The script will be put in Appendix c. This update script will use the stored procedure called ‘sp_updateRelationshipLink_cardinality”, which is put in Appendix f. this stored procedure will check (a) that tblRelationshipLink contains a record with these values, and (b) that the current value of the cardinality attribute for this record is “Unknown”. Only when the update code meets these two conditions, it runs. If there is no corresponding record, the system shows error message, as shown in Figure 26. And if the value of cardinality for this record is not “Unknown”, the system also shows error message, as shown in Figure 27. This script may be more difficult than directly changes to the table itself, it is because in our example, the database is too simple. However, in the more complex database, for example, with thousands columns, updates the database through the stored procedures is much more accurate and secure. The table changes to Figure 25.

relationshipLi...	relationshipLinkName	link_ID	relationship_ID	entityName	participation	cardinality
1	Employment_Employe...	3	1	Employee	Unknown	One

Figure 25 the relationship_Link table after update

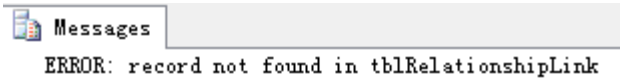


Figure 26 Error message if the record not found

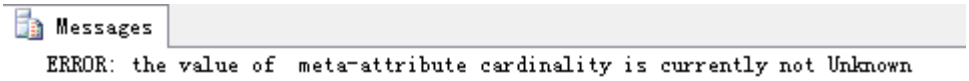


Figure 27Error message if cardinality is not “Unknown”

To process such updates automatically requires the implementation of database software known as a "trigger". This trigger is named “tr_Cardinality_Update”. The trigger is executed whenever the value of “cardinality” updates from “Unknown” to “One”. The trigger then automatically determines that the Many-to-Many relationship can be replaced by a Many-to-One relationship. The code of “tr_Cardinality_Update” will be put in Appendix d.

The trigger automatically adds a new column to the Employee table, and populates the new column with the foreign key references to the Department table, as shown in Figure 28. Now we can see which employee works in which Department. And it is a Many-to-One relationship now. The Bridge table is redundant and when application software has been adapted, the bridge table can be dropped.

keyEmployee	EmpName	keyDepartment
11	John Smith	101
12	Jane Brown	102
13	Ann Jones	103
14	Robert Bruce	103

Figure 28 New Employee table

6.2.2 Participation

As an example, we still use the EmpsAndDetps database.

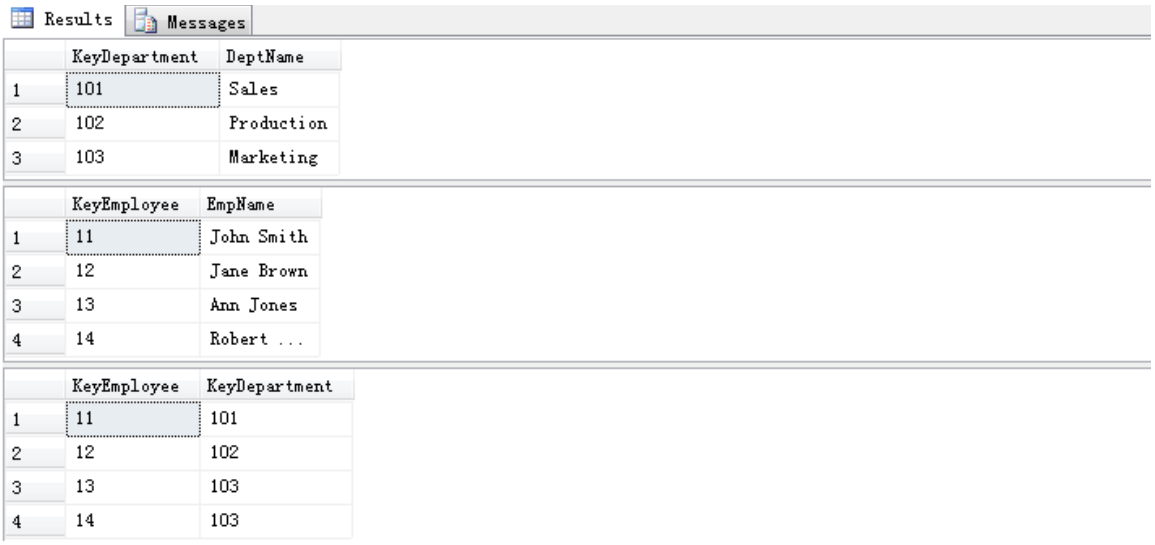


Figure 29 Database EmpsAndDetps

In the beginning, the value of meta-attribute "participation" is "Unknown". In this situation, it means an employee somehow can be unassigned to any department. It is interesting that we could set an employee's department value as "NULL" in the database. As shown in Figure 30, after we run the update script, which is in Appendix c, we can see the employee John Smith now does not belong to any department.

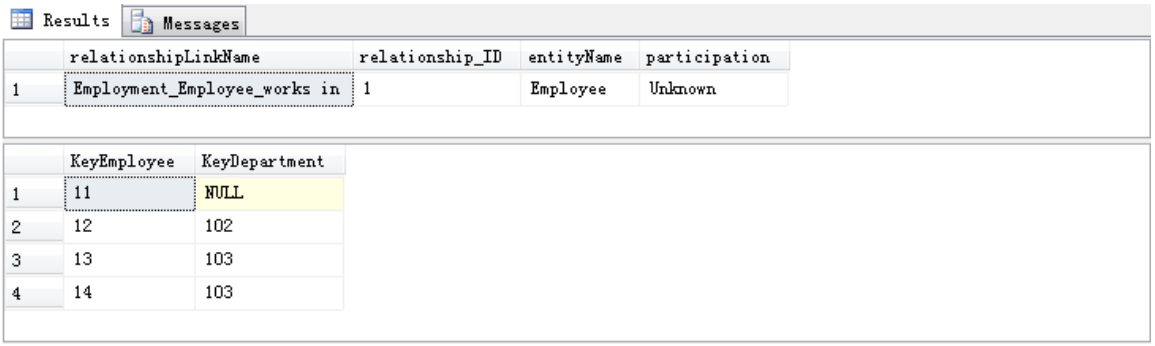


Figure 30 Update the keyDepartment value to "NULL"

Now suppose we have discovered that all employees must be assigned to a department. In this scenario, it means we need to update the value of the meta-attribute participation from "Unknown" to "Total", and the "KeyDapartment" cannot take a value of "NULL". Here we use the script to update the value of "participation", and also change the value of "KeyDeparment" from "NULL" to "101". This script will be put in Appendix c. After we run the script, the value of "participation" became to "Total", as shown in Figure 31. This update script will use the stored procedure called 'sp_updateRelationshipLink_ participation", which is put in Appendix e. This stored procedure will check (a) that tblRelationshipLink contains a record with these values (b) that the current value of the participation attribute for this record is "Unknown". Just like cardinality, if there is no corresponding record, the system shows an error message, as shown in Figure 32. And if the value of participation for this record is not "Unknown", the system also shows error message, as shown in Figure 33. Whenever we do this update, the trigger named "tr_Participation_Uplode" will run automatically, and then the trigger will set the values of "keyDepartment" to "NOT NULL".

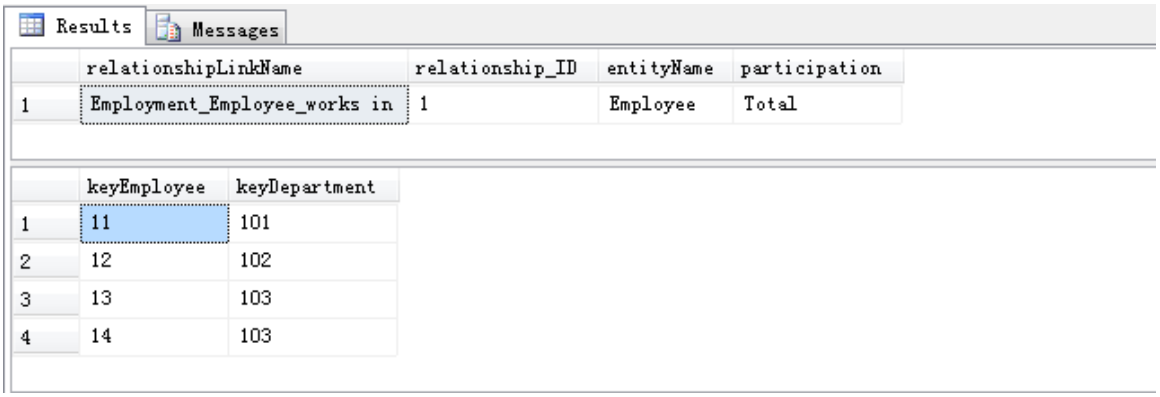


Figure 31 Update the value of participation

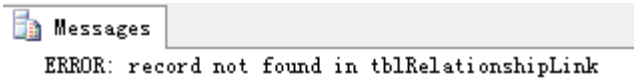


Figure 32 Error message if the record not found

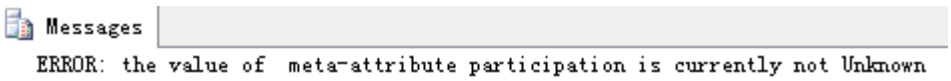


Figure 33 Error message if participation is not "Unknown"

To test that, we run the script and try again to update the value of keyDepartment to "NULL". However, at this time, John Smith's department value cannot be changed to "NULL", as

shown in Figure 34. The system will show "Cannot insert the value NULL into column 'KeyDepartment'", and the update will be canceled.

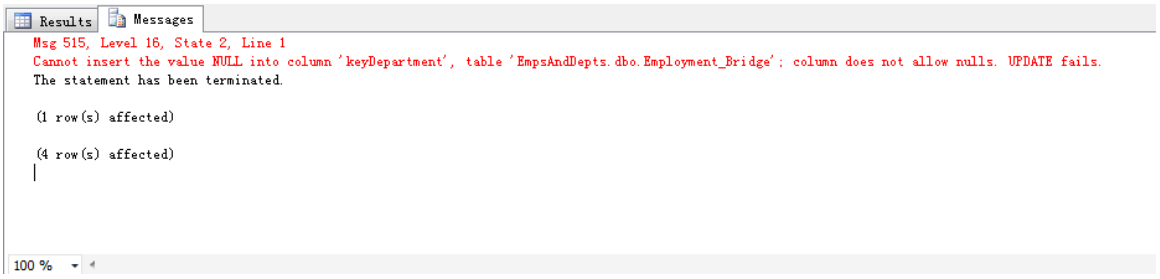


Figure 34 Error message

6.2.3 isWeak

As mentioned before, there is a relationship between the meta-attribute "isWeak" in the Entity table and the meta-attribute "identifierDiscriminaterOrdinary" in the Attribute table. Here we are going to illustrate how one meta-attribute value change to the conceptual model can cause other changes. Here is an example scenario. Suppose we have an Employee entity class and a Department entity class. As we known, the value of "identifierDiscriminaterOrdinary" will keep "Unknown" as long as the value of "isWeak" is Unknown.

Here we use T-SQL script to insert the two entities "Employee" and "Department" to the tblEntity class, as shown in Figure 35. The values of "isWeak" and the value of "identifierDiscriminatorOrdinary" are both set as Unknown.

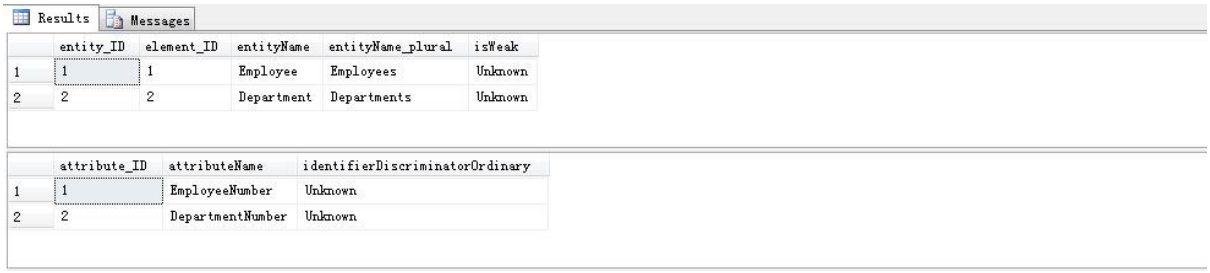
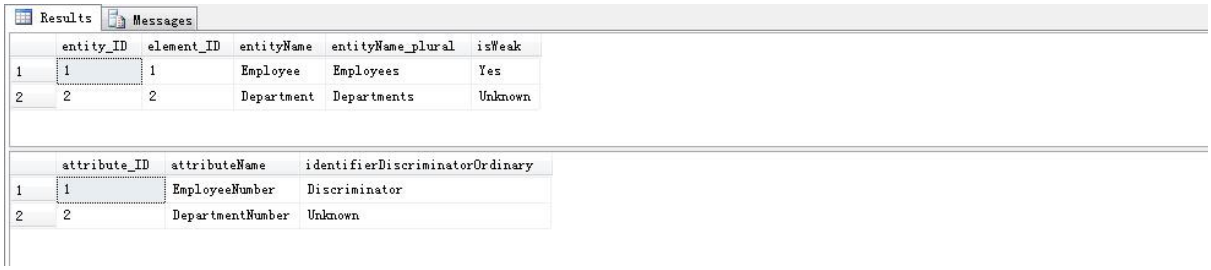


Figure 35 tblEntity class and tblAttribute class

As we described in Chapter 5, if an entity class is eventually classified as a weak one, one of the attributes of this entity will act as a "Discriminator". If, however, the entity class is classified as a strong one, then one of the attributes can be classified as an "Identifier". So when we update the value of "isWeak", the "identifierDiscriminatorOrdinary" will

automatically change. Here we use the trigger named “isWeak_Update” to achieve the automating process. This trigger will be put in Appendix d.

To test this trigger, we use a script to update the value of “isWeak”. After we ran it, we got these two tables. As it shown, the value of “isWeak” set as “Yes”, and then the value of “identifierDiscriminatorOrdinary”, as we want, automatically changes to Discriminator.


Results Messages

	entity_ID	element_ID	entityName	entityName_plural	isWeak
1	1	1	Employee	Employees	Yes
2	2	2	Department	Departments	Unknown

	attribute_ID	attributeName	identifierDiscriminatorOrdinary
1	1	EmployeeNumber	Discriminator
2	2	DepartmentNumber	Unknown

Figure 36 Update the value of “isWeak” to “Yes”

Similarly, if we set the value of “isWeak” to No, we get an Identifier value.


Results Messages

	entity_ID	element_ID	entityName	entityName_plural	isWeak
1	1	1	Employee	Employees	No
2	2	2	Department	Departments	Unknown

	attribute_ID	attributeName	identifierDiscriminatorOrdinary
1	1	EmployeeNumber	Identifier
2	2	DepartmentNumber	Unknown

Figure 37 Update the value of “isWeak” to “No”

7. Discussion

The EERMM database is a promising approach to creating and managing the database, especially when there is insufficient information for a complete database design. As shown in Figure 38, the approach includes 7 steps.

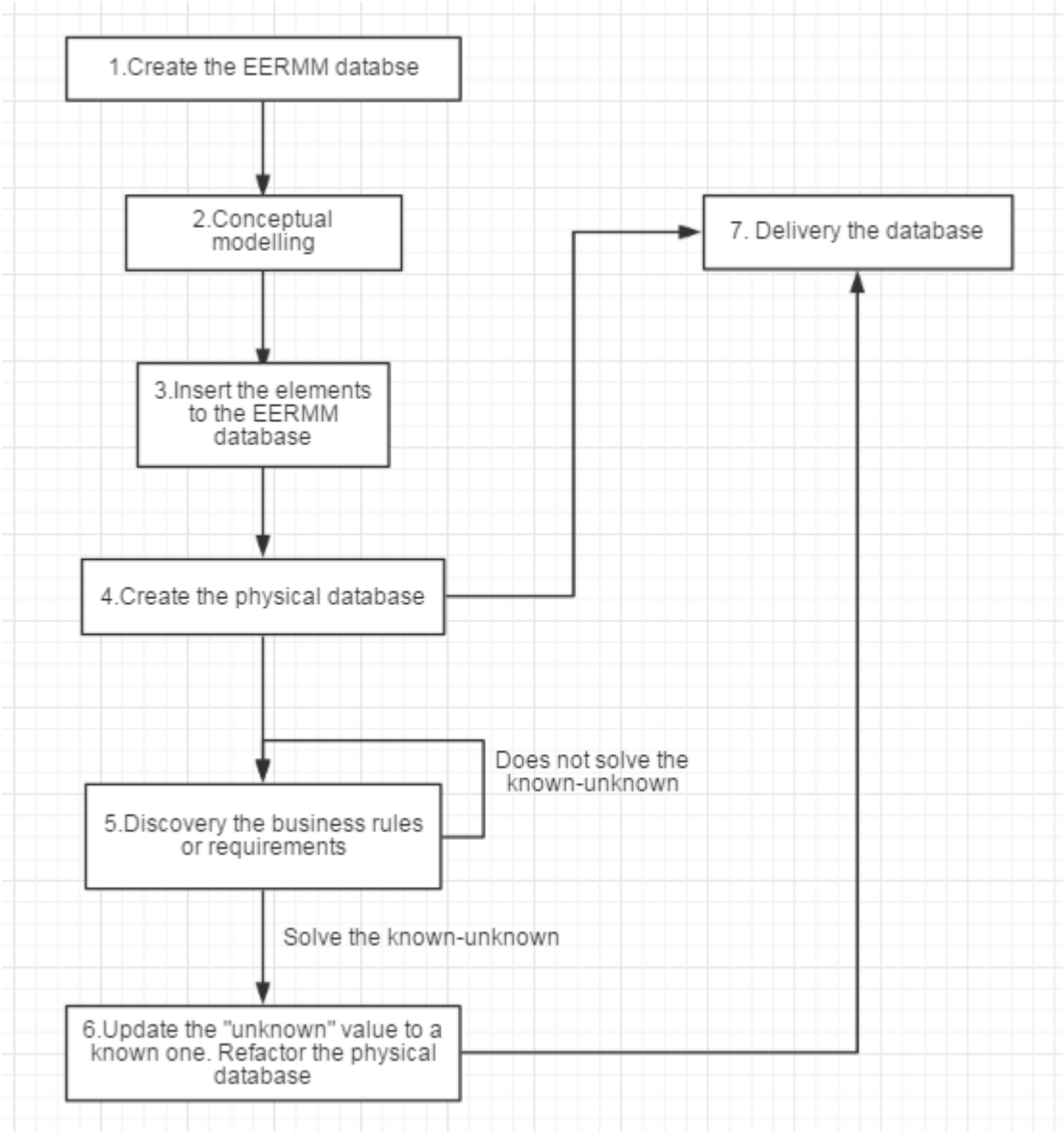


Figure 38 The process to use EERMM database

1. Create the EERMM database. The database developer can easily and automatically create this database by the create script.
2. Conceptual modelling. Requirements analyst and database modeler coordinate, according to the results of requirement analysis, define the entities, relationships and attributes. Draw the ER model of those elements. Mark the known-unknown information.

3. Insert the elements into the EERMM database. According to the conceptual model, the database developer needs to insert those elements into the EERMM database, values the known-unknowns as “Unknown”.

4. Create physical database. According to steps 2 and 3, the database developer creates a usable physical database and inserts its name to the tblSchema class of EERMM database, then delivers it.

5. Discover new business rules. When the discovery of a new business rule allows a known-unknown to be partially or fully resolved, the process goes to step 6. If not, repeat this step.

6. The following steps are initiated.

a. In the ER model, the construct is marked as resolved, or partially resolved in this particular way.
b. The data is checked for violations of the business rule and a warning message is generated if any such violations are detected.
c. Deprecation notices should be issued concerning the use of any structures which might disappear or be altered as a consequence of this known-unknown resolution.
   All knock-on effects of this known-unknown resolution must be detected and the associated design constructs altered accordingly. (a) The conceptual model is updated, (b) the physical database design is changed and (c) The necessary changes to the data in the database are made. All of this happens completely automatically.

7. Deliver the new version of the physical database.

There are many benefits to develop the database with our approach. Firstly, the abstraction of the meta-model is one level higher than normal models, so the database developer has more comprehensive and higher vision to manage the database, and easily generalizes the problem in the modeling process. Secondly, the traditional database design approaches need the complete information before the database implementation. However, the EERMM database allows the developer to develop the physical database with incompleteness. It is more suitable for the incremental software development environment such as agile software development methods. The developer can leave the incompleteness to other iterations, and keep delivering the usable database to satisfy the value of stakeholders. Lastly, our approach simplifies the database developer’s work. The EERMM database provides many triggers and stored procedures to automate the database development process. The user only needs to update the

unknown value to a known one after the new requirements has been discovered, and then all the changes to the conceptual model and physical database can be made automatically.

However, the method of EERMM database with “Unknown” values is on the initial stage. There are still a number of aspects which should be improved. Firstly, the definition of “Unknown” values for the meta-attributes is not complete. In this thesis, we only take part of the meta-attributes into account to explain our approach. We expect to analyse and specify all the meta-attributes which can have “Unknown” value, and how to change the conceptual model and refactor the physical database when the unknowns change to unknown values in the future. Secondly, our database only supports the binary relationship. Even though the binary relationship can generalize most relationships in the real world, we still need to solve this limitation, and to support the ternary relationships, even n-ary relationships in the future. At last, we need to implement more triggers and stored procedures to automate the database development process.

8. Conclusions

In this research, a database solution named ER meta-model database with unknown values is proposed for the implementation and refactoring of the database with incomplete information. This database is based on the Fidalgo's meta-model. By using this database, the developer can easily implement the database with incomplete requirements in an incremental development team such as agile team. The developer can develop an incomplete but usable database in the early iterations, and marks the known-unknowns as "unknown" to the database. In the after iterations, when the known-unknowns transform to known-knowns, the developer can simply update the "unknown" values to known values, and the system will automatically do the corresponding changes to the conceptual model and physical database.

My contribution in this research includes two parts. On the one hand, I generalized the possible meta-attributes which can contain "Unknown" value. And I also discovered what the relationship is between different meta-attributes, and between meta-attribute and the physical database structure. Then I discovered when the values of a meta-attribute change from "Unknown" to known values, how these changes impact the other values in the conceptual model and the physical database. On the other hand, I implemented several triggers and stored procedures to support the automatic process.

According to the discussion in Chapter 7, there are still some limitations in this research. The compiled meta-model database is applicable. However, it is required to implement all the possible unknown values to the database, which in my research only implemented some, this needs to be resolved in the future. Moreover, I still need to compile more triggers and stored procedures to support the automatic process.

References

[Agile Alliance, 2015] Agile Alliance. What is Agile software development. Available at: http://www.agilealliance.org/the-alliance/what-is-agile/, 2015

[Abrahamsson, 2003] Abrahamsson, Pekka, et al. New directions on agile methods: a comparative analysis. Software Engineering, 2003. Proceedings. 25th International Conference on. Ieee, 2003.

[Ambler, 2005] Scott J Ambler. The Process of Database Refactoring: Strategies for Improving Database Quality. Available at: http://agiledata.org/essays/databaseRefactoring.html, 2005

[Ambler et al., 2006] Scott J Ambler, Pramod J. Sadalage. Refactoring Databases: Evolutionary Database Design. Addison-Wesley Professional, 2006

[Beck, 1999] Beck, Kent. Embracing Change with Extreme Programming. Computer 32 (10): 70 – 77. 1999.

[Beck and Fowler, 2000] Kent Beck, Martin Fowler. Planning Extreme Programming. Addison-Wesley Professional. 2000

[Chen, 1976] Peter Chen. The Entity Relationship Model — Toward a Unified View of Data. Springer Berlin Heidelberg. 1976.

[Coad et al, 1999] Peter Coad, Eric Lefebvre and Jeff De Luca. Java Modeling in Color with UML. Prentice Hall PTR. 1999.

[Cheng and Atlee, 2007] Cheng B. H. C., Atlee, J. M.: Research Directions in Requirements Engineering. Future of Software Engineering (FOSE '07), IEEE. 2007

[Cohn, 2004] Mike Cohn. User Stories Applied: For Agile Software Development. Addison-Wesley Professional; 1 edition. 2004

[Cohn, 2005] Mike Cohn. Agile Estimating and Planning. Prentice Hall PTR. 2005

[CODD, 1970] CODD, E.F. A relational model of data for large shared data banks. Comm. ACM 13, 6. 1970

[Elmasri and Navathe, 2010] Elmasri, R., Navathe, S.: Fundamentals of Database Systems, 6th edn.. p. 1200. Addison Wesley. 2010

[Fowler et al, 1999] Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999

[Fidalgo et al., 2012] Fidalgo, R. do N., Maranh~ao, E. de S., Espa~na, S., de Castro, J.B., Pastor, O. EERMM: A Metamodel for the Enhanced Entity-Relationship Model. Conceptual Modeling - 31st International Conference ER 2012, LNCS 7532, pp. 515-524. 2012

[Fidalgo et al., 2013] Fidalgo, R. N., Alves, E., España, S., Castro, J., and Pastor, O. Metamodeling the Enhanced Entity-Relationship Model. Journal of Information and Data Management. 2013

[Gomory, 1995] Gomory, R.: The Known, the Unknown and the Unknowable. Scientific American, June, pp. 120. 1995

[Hernandez, 2003] Michael J. Hernandez. Database Design for Mere Mortals™: A Hands-On Guide to Relational Database Design, Second Edition. Addison Wesley. 2003

[Highsmith, 2002] Jim Highsmith. Agile Software Development Ecosystems. Addison-Wesley Professional; 1 edition. 2002

[Hirotaka and Ikujiro, 1986] Hirotaka Takeuchi, Ikujiro Nonaka. The new new product development game. Harvard Business Review 86116:137–146. 1986.

[Irani et al., 2001] Irani, P., Tingley, M., and Ware, C. Using Perceptual Syntax to Enhance Semantic Content in Diagrams. IEEE. Computer Graphics and Applications 21 (5): 76–85. 2001.

[Layton, 2012] Mark C. Layton. Agile Project Management For Dummies. JOHN WILEY & SONS SINGAPORE PTE LTD. 2012

[Martin, 2003] Martin, Robert C. Agile software development: principles, patterns, and practices. Pearson Education, Inc. 2003.

[Richards, 2007] Keith Richards. Agile project management: running PRINCE2 projects with DSDM Atern. OGC - Office of Government Commerce. The Stationery Office. 2007.

[Rybinski, 1987] Rybinski, H.. On First-Order-Logic Databases. ACM Transactions on Database Systems 12 (3): 325–349. 1987

[Salay et al., 2013] Rick Salay, Marsha Chechik, Jennifer Horkoff, Alessio Di Sandro. Managing requirements uncertainty with partial models. Requirements Engineering. Springer-Verlag. 2013

[Schwaber, 1996] Ken Schwaber. Controlled Chaos: Living on the Edge. American Programmer. April 1996.

[Schwaber, 2004] Ken Schwaber. Agile Project Management with Scrum (Developer Best Practices). Microsoft Press; 1 edition. 2004

[Sunderic and Woodhead, 2002] Dejan Sunderic and Tom Woodhead. Stored Procedure Programming. The McGraw-Hill Companies. 2002

[Teorey et al. 1986] Teorey, Toby J., Dongqing Yang, and James P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. ACM Computing Surveys, 1986.

[Teorey et al, 2011] Toby J. Teorey, Sam S. Lightstone, Tom Nadeau, H.V. Jagadish. Database Modeling and Design. Morgan Kaufmann; 5 edition. 2011

[Thalheim, 2011] Bernhard Thalheim. The Theory of Conceptual Models, the Theory of Conceptual Modelling and Foundations of Conceptual Modelling. Handbook of Conceptual Modeling, pp 543-577 .Springer Berlin Heidelberg. 2011

[Thalheim, 2012] Bernhard Thalheim. The Science and Art of Conceptual Modeling. Transactions on Large-Scale Data- and Knowledge-Centered Systems VI, pp 76-105. Springer-Verlag Berlin Heidelberg. 2012

[Ware and Bobrow, 2004] Ware, C. and Bobrow, R. Motion to Support Rapid Interactive Queries on Node–Link Diagrams. ACM Transactions on Applied Perception 1 (1): 3–18. 2004.

[Wiegers, 2003] Wiegers, K. E.: *Software Requirements, 2nd ed*. Microsoft Press. 2003

[Wan-Kadir and Loucopoulos, 2004] Wan-Kadir, Wan M. N., Loucopoulos, P.: Relating evolving business rules to software design. Journal of Systems Architecture 50(7): 367-382. 2004

[Zhang et al., 2014] Zheying Zhang, Peter Thanisch, Jyrki Nummenmaa, Jing Ma. Detecting Missing Requirements in Conceptual Models. Information and Software Technologies Communications in Computer and Information Science, Volume 465, pp 248-259. 2014

Appendix. Database Software

This Appendix contains a representative sample of the code developed for this project. The three code fragments are (a) EERMM database create script (b) a stored procedure used for inserting a new element of the concrete ER model into the meta-model database, (c) a stored procedure used to update a meta-attribute value from "Unknown" to some known value and (d) The triggers which detects when a meta-attribute value has changed and handle the "knock-on" effects of that update, (e) the stored procedures to support the updating, (f) the script to create EmptsAndDepts database.

(a) EERMM database create script

```
USE [master]
GO
/****** Object:  Database [EERMM]    Script Date: 15/06/2014 18:45:53 ******/
CREATE DATABASE [EERMM]
 CONTAINMENT = NONE
 ON PRIMARY
( NAME = N'ERMM', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\ERMM.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'ERMM_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.MSSQLSERVER\MSSQL\DATA\ERMM_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
ALTER DATABASE [EERMM] SET COMPATIBILITY_LEVEL = 110
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [EERMM].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [EERMM] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [EERMM] SET ANSI_NULLS OFF
GO
ALTER DATABASE [EERMM] SET ANSI_PADDING OFF
GO
ALTER DATABASE [EERMM] SET ANSI_WARNINGS OFF
```

```
GO
ALTER DATABASE [EERMM] SET ARITHABORT OFF
GO
ALTER DATABASE [EERMM] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [EERMM] SET AUTO_CREATE_STATISTICS ON
GO
ALTER DATABASE [EERMM] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [EERMM] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [EERMM] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [EERMM] SET CURSOR_DEFAULT GLOBAL
GO
ALTER DATABASE [EERMM] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [EERMM] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [EERMM] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [EERMM] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [EERMM] SET DISABLE_BROKER
GO
ALTER DATABASE [EERMM] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [EERMM] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [EERMM] SET TRUSTWORTHY OFF
GO
ALTER DATABASE [EERMM] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [EERMM] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [EERMM] SET READ_COMMITTED_SNAPSHOT OFF
GO
ALTER DATABASE [EERMM] SET HONOR_BROKER_PRIORITY OFF
```

```
GO
ALTER DATABASE [EERMM] SET RECOVERY FULL
GO
ALTER DATABASE [EERMM] SET MULTI_USER
GO
ALTER DATABASE [EERMM] SET PAGE_VERIFY CHECKSUM
GO
ALTER DATABASE [EERMM] SET DB_CHAINING OFF
GO
ALTER DATABASE [EERMM] SET FILESTREAM( NON_TRANSACTED_ACCESS = OFF )
GO
ALTER DATABASE [EERMM] SET TARGET_RECOVERY_TIME = 0 SECONDS
GO
EXEC sys.sp_db_vardecimal_storage_format N'EERMM', N'ON'
GO
USE [EERMM]
GO
USE [EERMM]
GO
/****** Object: Sequence [dbo].[seq_attribute_ID] Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_attribute_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
 CYCLE
 CACHE
GO
USE [EERMM]
GO
/****** Object: Sequence [dbo].[seq_attributeLink_ID] Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_attributeLink_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
```

```
 CYCLE
 CACHE
GO
USE [EERMM]
GO
/****** Object:  Sequence [dbo].[seq_element_ID]    Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_element_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
 CYCLE
 CACHE
GO
USE [EERMM]
GO
/****** Object:  Sequence [dbo].[seq_entity_ID]    Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_entity_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
 CYCLE
 CACHE
GO
USE [EERMM]
GO
/****** Object:  Sequence [dbo].[seq_link_ID]    Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_link_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
 CYCLE
 CACHE
```

```
GO
USE [EERMM]
GO
/****** Object:  Sequence [dbo].[seq_node_ID]    Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_node_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
 CYCLE
 CACHE
GO
USE [EERMM]
GO
/****** Object:  Sequence [dbo].[seq_relationship_ID]    Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_relationship_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
 CYCLE
 CACHE
GO
USE [EERMM]
GO
/****** Object:  Sequence [dbo].[seq_relationshipLink_ID]    Script Date: 15/06/2014 18:45:54 ******/
CREATE SEQUENCE [dbo].[seq_relationshipLink_ID]
 AS [int]
 START WITH 1
 INCREMENT BY 1
 MINVALUE 1
 MAXVALUE 2147483647
 CYCLE
 CACHE
GO
```

```
/****** Object:  StoredProcedure [dbo].[sp_AddAttributeToEntity]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_11_14
-- Description:    Add a new attribute to an existing element
-- Change date  2014_01_28
--          (1) There was a mistake in this stored procedure.
--          Instead of establishing the attribute as an
--          element, this stored procedure incorrectly
--          used the elementID of the owning entity.
--          The stored procedure now creates a new
--          element for this attribute.
--          (2) There are some other minor changes.
--          Some of the fields were not getting initialized
--          to unknown. I have now corrected that.
--          (3) There was an error in the arguments passed
--          to the stored procedure sp_createAttributeLink.
--          The correct arguments are now passed.
--          (4) In the earlier version, one of the
--          arguments was the entity_id of the owning
--          entity. This has now been changed so that the
--          attribute name is passed instead.
--          (5) I have improved the error checking. The
--          return codes from stored procedures are checked.
-- =============================================
CREATE PROCEDURE [dbo].[sp_AddAttributeToEntity]
   @entityName SYSNAME,
        @attributeName SYSNAME
AS
BEGIN
   SET NOCOUNT ON;
   DECLARE @entity_ID INT,
              @element_ID INT,        -- This one is the element ID of the attribute
```

```
                @owningElement_ID INT,   -- This one is the element ID of the entity which
                                -- owns the attribute
                @attribute_ID INT,
                @attributeLink_ID INT,
            @identifierDiscriminatorOrdinary  VARCHAR(13)  = 'Unknown',
        @simpleOrComposite VARCHAR(9)  = 'Unknown',
        @singleOrMultivalued  VARCHAR(10)  = 'Unknown',
        @storedOrDerived VARCHAR(7)  = 'Unknown',
                @attributeType VARCHAR(20)  = 'Unknown',
        @dataType VARCHAR(20)  = 'Unknown',
                @size INT = NULL,
                @isNull CHAR(7) = 'Unknown',
                @defaultValue  VARCHAR(MAX)  = NULL,
        @comment VARCHAR(MAX)  = NULL,
                @cardinalityStatus CHAR(7) = 'Unknown',
                @cardinality INT = NULL,
                @RC INT = 0;

    SELECT @entity_ID = [entity_ID] FROM tblEntity WHERE [entityName] = @entityName;

    -- Set up the attribute link.
    SELECT @owningElement_ID = element_ID FROM tblElement WHERE [elementName] =
@entityName;
  EXECUTE @RC = [dbo].[sp_createAttributeLink] @attribute_ID,
                                @owningElement_ID,

@attributeLink_ID OUTPUT;
  IF @RC < 0
        RETURN -1;
    ELSE
    BEGIN
      -- In the metamodel, each attribute is an element.
          -- We have to register the attribute as a new element.
      EXECUTE @RC = sp_createElement @attributeName, 'Attribute', @element_ID OUTPUT;
          IF @RC < 1
            RETURN -1;
          ELSE
          BEGIN
```

```
            -- Get the value for the attributes primary key field.
        EXECUTE @attribute_ID = sp_getSeqNumberAttribute;

            INSERT INTO tblAttribute(
                        [attribute_ID], [element_ID], [attributeLink_ID], [attributeName],
                        [identifierDiscriminatorOrdinary],
                        [simpleOrComposite],
                        [singleOrMultivalued],
                        [storedOrDerived],
                        [attributeType],
                        [dataType],[size],[Nullable],[defaultValue],
                        [comment], [cardinalityStatus], [cardinality])

                VALUES(@attribute_ID, @element_ID, @attributeLink_ID, @attributeName,
                        @identifierDiscriminatorOrdinary,
                        @simpleOrComposite,
                        @singleOrMultivalued,
                        @storedOrDerived,
                        @attributeType,
                        @dataType, @size, @isNull, @defaultValue,
                        @comment, @cardinalityStatus, @cardinality);
    END
  END
END

GO
/****** Object:  StoredProcedure [dbo].[sp_AddAttributeToRelationship]    Script Date: 15/06/2014
18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_11_14
-- Description:    Add a new attribute to an existing element
-- =============================================
CREATE PROCEDURE [dbo].[sp_AddAttributeToRelationship]
```

```
   @relationship_ID INT,
         @attributeName SYSNAME
AS
BEGIN
   DECLARE @relationshipName SYSNAME,
            @element_ID INT,
                     @attribute_ID INT,
                     @attributeLink_ID INT,
                     @attributeType VARCHAR(20)  = 'Unknown',
            @dataType VARCHAR(20)  = 'Unknown',
                     @size INT = NULL,
                     @isNull CHAR(7) = 'Unknown',
                     @defaultValue  VARCHAR(MAX)  = NULL,
            @comment VARCHAR(MAX)  = NULL,
                     @cardinalityStatus CHAR(7) = 'Unknown',
                     @cardinality INT = NULL,
                     @RC INT = 0;

         SELECT @relationshipName = relationshipName FROM tblRelationship
                                    WHERE [relationship_ID] = @relationship_ID;
         SELECT @element_ID = element_ID FROM tblElement WHERE [elementName] =
@relationshipName;

         EXECUTE @attribute_ID = sp_getSeqNumberAttribute;
         INSERT INTO tblAttribute(
               [attribute_ID], [element_ID], [attributeLink_ID], [attributeName],
                        [attributeType],
         [dataType],[size],[isNull],[defaultValue],
                        [comment], [cardinalityStatus], [cardinality])

            VALUES(@attribute_ID,  @element_ID, @attributeLink_ID, @attributeName,
                     @attributeType,
               @dataType, @size, @isNull, @defaultValue,
               @comment, @cardinalityStatus, @cardinality);

   EXECUTE @RC = [dbo].[sp_createAttributeLink] @attribute_ID, @attributeName,
                                    @element_ID, @relationshipName
   IF @RC < 0
```

```
            RETURN -1;
          ELSE
            RETURN 1;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_addFunctionalDependencies]    Script Date: 15/06/2014
18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2014_01_23
-- Description:    generate the set of FDs
--            associated with a ternary
--            relationship
-- =============================================
CREATE PROCEDURE [dbo].[sp_addFunctionalDependencies]
   @relationshipName SYSNAME,
          @entityName_1 SYSNAME,
          @entityName_2 SYSNAME,
          @entityName_3 SYSNAME
AS
BEGIN
          SET NOCOUNT ON;
   INSERT INTO [dbo].[tblFunctionalDependencies]
        ([relationshipName]
        ,[left1_entityName]
        ,[left2_entityName]
        ,[right_entityName]
        ,[status])
    VALUES
        (@relationshipName, @entityName_1, @entityName_2, @entityName_3, 'Unknown'),
                    (@relationshipName, @entityName_1, @entityName_3, @entityName_2, 'Unknown'),
        (@relationshipName, @entityName_2, @entityName_3, @entityName_1, 'Unknown'),
        (@relationshipName, @entityName_1, NULL, @entityName_2, 'Unknown'),
```

```
        (@relationshipName, @entityName_1, NULL, @entityName_3, 'Unknown'),
        (@relationshipName, @entityName_2, NULL, @entityName_1, 'Unknown'),
        (@relationshipName, @entityName_2, NULL, @entityName_3, 'Unknown'),
        (@relationshipName, @entityName_3, NULL, @entityName_1, 'Unknown'),
        (@relationshipName, @entityName_3, NULL, @entityName_2, 'Unknown')

END

GO
/****** Object:  StoredProcedure [dbo].[sp_createAttribute]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                  Peter
-- Create date: 2013_11_11
-- Description:    Create NEW Attribute  in the design
--          Then associate the attribute with a
--          new attribute link.  The new attribute
--          link is in turn associated with the
--          entity or relationship  which owns
--          the attribute.
--  Modified
-- 2014_01_20   sp_createAttibuteLink  returns the
--          attributeLink_ID value. This is
--          now inserted into tblAttribute.
-- 2014_01_29   (1)There was an error in the call to
--          sp_createElement. This has been
--          corrected.
--          (2) The stored procedure sp_createAttributeLink
--          now takes @attributeName as an argument.
-- =============================================
CREATE  PROCEDURE  [dbo].[sp_createAttribute]

          @attributeName [sysname],
          @owningElementName  SYSNAME,
          @identifierDiscriminatorOrdinary  VARCHAR(13)   = 'Unknown',
```

```
        @simpleOrComposite VARCHAR(9)  = 'Unknown',
    @singleOrMultivalued  VARCHAR(11)  = 'Unknown',
    @storedOrDerived VARCHAR(7)  = 'Unknown',
        @attributeType [varchar](50)  = 'Unknown',
        @dataType [varchar](50)  = 'Unknown',
        @size [int] = NULL,
        @Nullable  [char](7)  = 'Unknown',
        @defaultValue  [varchar](max) = NULL,
        @comment [varchar](max) = NULL,
        @cardinalityStatus CHAR(7)  = 'Unknown',
        @cardinality [int] = NULL
AS
BEGIN
    SET NOCOUNT ON
    DECLARE  @RC INT = 0,
            @element_ID INT,
                    @owningElement_ID INT,
                    @attribute_ID [int],
                    @attributeLink_ID [int];

    EXECUTE  @attribute_ID = sp_getSeqNumberAttribute;
        -- An attribute is a kind of Element, so create an Element object for
        -- the new attribute
        EXECUTE  dbo.sp_createElement @attributeName, 'Attribute', @element_ID OUTPUT;

    SET @owningElement_ID = dbo.fn_getElement_ID(@owningElementName);
        -- Notice that we have to pass both the ID and the name of the attribute to this stored
        -- procedure.
        EXECUTE  @RC  =  [dbo].[sp_createAttributeLink]  @attribute_ID,  @attributeName,
@owningElement_ID, @attributeLink_ID OUTPUT;
    IF @RC < 0
        RETURN -1;
        ELSE
        BEGIN
        INSERT INTO tblAttribute(
            [attribute_ID], [element_ID], [attributeLink_ID], [attributeName], [attributeType],
        [dataType],[size],[Nullable],[defaultValue],
                    [comment], [cardinalityStatus], [cardinality])
```

```
            VALUES(@attribute_ID,  @element_ID, @attributeLink_ID, @attributeName,

                        @attributeType,
                  @dataType, @size, @Nullable,  @defaultValue,
                  @comment, @cardinalityStatus, @cardinality);
      SET @owningElement_ID = [dbo].[fn_getElement_ID](@owningElementName);
            RETURN @attribute_ID;
         END
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createAttributeLink]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_11_11
-- Description:    Create a new AttributeLink
-- Modified
-- 2014_01_20: It now returns the attributeLink_ID
-- 2014_01_29: The name of the attribute is now
--           passed as an argument to the
--           procedure.
-- =============================================
CREATE PROCEDURE [dbo].[sp_createAttributeLink]
   @attribute_ID INT,
         @attributeName SYSNAME,
   @owningElement_ID INT,
         @attributeLink_ID INT OUTPUT
AS
BEGIN
         DECLARE @attributeLinkName SYSNAME;
         DECLARE @link_ID INT;
         DECLARE @elementName SYSNAME;
```

```
        SET NOCOUNT ON;
        EXECUTE @attributeLink_ID = sp_getSeqNumberAttributeLink;
        SET @elementName = dbo.fn_getElementName( @owningElement_ID );
        SET @attributeLinkName = @elementName + '_' + @attributeName;

        -- Create the Link object and get back its ID:
        EXEC sp_createLink @attributeLinkName, @link_ID;

        SET @attributeLink_ID = NEXT VALUE FOR [seq_attributeLink_ID];
        INSERT INTO tblAttributeLink([attributeLink_ID],
                        [attribute_ID],
                        [link_ID],
                                                element_ID,
                        [attributeLinkName])
                          VALUES( @attributeLink_ID,
                                                @attribute_ID,
                                                @link_ID,
                                                        @owningElement_ID,
                                                @attributeLinkName );
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createBinaryRelationship]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_11_11
-- Description:    Create NEW Relationship in the design
-- CHANGES
--          2013_11_13 This SP has been changed because
--                  of changes to tblElement.
--                  When creating an element, we now
--                  have to specify what kind of element
```

```
--                  it is.
-- ============================================
CREATE PROCEDURE [dbo].[sp_createBinaryRelationship]
        @relationshipName SYSNAME,
        @entityName_1 SYSNAME,
        @entityName_2 SYSNAME,
        @relationship_ID INT OUTPUT,
        @isIdentifier CHAR(7) = 'Unknown',
        @role_1_sglr SYSNAME = NULL,
        @role_1_plrl SYSNAME = NULL,
        @role_2_sglr SYSNAME = NULL,
        @role_2_plrl SYSNAME = NULL
AS
BEGIN
  DECLARE @element_ID INT,
            @elementType CHAR(12) = 'Relationship';

        SET NOCOUNT ON;

        -- STEP 1: create the Generic Relationship object
  EXEC sp_createRelationship @relationshipName, @relationship_ID OUTPUT, @isIdentifier,
                        'binary';

  -- STEP 2: Create the first Relationship Link and use it to associate the first Entity
        DECLARE @link_ID_1 INT;
        EXEC sp_createRelationshipLink @relationshipName, @entityName_1, @link_ID_1 OUTPUT,
                        @role_1_sglr, @role_1_plrl;

        -- STEP 3: Create the second Relationship Link and use it to associate the second Entity
        DECLARE @link_ID_2 INT;
        EXEC sp_createRelationshipLink @relationshipName, @entityName_2, @link_ID_2 OUTPUT,
                        @role_2_sglr, @role_2_plrl;
        RETURN 1;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createElement]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
```

```
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:                Peter
-- Create date: 2013_11_13
-- Description:    CHANGE:  the new attributes elementType
--                 and owningElementOfAttributeID
--                 are inserted.
-- ============================================
CREATE PROCEDURE [dbo].[sp_createElement]
        @elementName SYSNAME,
        @elementType CHAR(12),
        @element_id INT OUTPUT
AS
BEGIN
   DECLARE @node_ID INT;

        EXECUTE dbo.sp_createNode @elementName, @node_ID OUTPUT;

        EXECUTE @element_ID = sp_getSeqNumberElement;
   INSERT INTO tblElement([element_ID], [node_ID], [elementName],
                          [elementType])
                   VALUES(@element_id,  @node_ID, @elementName,
                                          @elementType);
        RETURN 1;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createEntity]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:                Peter
-- Create date: 2013_11_11
-- Description:    Create  NEW Entity in the design
```

```
-- ============================================
CREATE PROCEDURE [dbo].[sp_createEntity]
        @entityName SYSNAME,
        @entity_ID INT OUTPUT,
        @isWeak CHAR(7) = 'Unknown',
        @entityName_plural SYSNAME = NULL
AS
BEGIN
  DECLARE @element_ID INT,
            @elementType CHAR(12) = 'Entity',
                    @owningElementOfAttribute INT = NULL;

        DECLARE @RC INT;
        SET NOCOUNT ON;

        IF EXISTS(SELECT * FROM tblEntity WHERE entityName = @entityName)
                -- There is already an entity with this name in the table.
                RETURN -2;
  ELSE
        BEGIN
    EXECUTE @entity_ID = sp_getSeqNumberEntity;

                EXECUTE @RC = sp_createElement @entityName,
                        @elementType,
                                                        @element_ID OUTPUT;
          IF @RC = 1
                BEGIN
                  IF @entityName_plural IS NULL
                        SET @entityName_plural = @entityName + 's';

       INSERT INTO tblEntity([entity_ID], [element_ID], [entityName], isWeak, entityName_plural)
                        VALUES(@entity_ID, @element_ID, @entityName, @isWeak,
@entityName_plural);
              RETURN 1;
                END
                ELSE
                BEGIN
              RETURN -1;
```

```
            END
        END
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createLink]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_11_08
-- Description:    Add a link node for the schema
-- =============================================
CREATE PROCEDURE [dbo].[sp_createLink]
        @linkName VARCHAR(50),
        @link_ID INT OUTPUT,
        @schema_id INT = 1
AS
BEGIN
        SET NOCOUNT ON;
        EXECUTE @link_ID = sp_getSeqNumberLink;
        INSERT INTO tblLink([link_ID], [schema_ID], [linkName])
        VALUES(@link_ID, @schema_ID, @linkName);
        RETURN 1;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createNode]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                <Author,,Name>
-- Create date: <Create Date,,>
-- Description:    <Description,,>
```

```
-- ============================================
CREATE PROCEDURE [dbo].[sp_createNode]
        @nodeName SYSNAME,
        @node_ID INT OUTPUT,
        @schema_id INT = 1  -- Default value for the schema
AS
BEGIN
        SET NOCOUNT ON;
        EXECUTE @node_ID = sp_getSeqNumberNode;
   INSERT INTO tblNode([node_ID], [schema_ID], [identity])
        VALUES(@node_ID,  @schema_ID, @nodeName);
        RETURN 1;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createRelationship]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:                Peter
-- Create date: 2013_11_11
-- Description:   Create  NEW Relationship  in the design
-- CHANGES
--          2013_11_13 This SP has been changed because
--                of changes to tblElement.
--                When creating an element, we now
--                have to specify what kind of element
--                it is.
-- ============================================
CREATE PROCEDURE [dbo].[sp_createRelationship]
        @relationshipName SYSNAME,
        @relationship_ID INT OUTPUT,
        @isIdentifier CHAR(7) = 'No',
        @degree CHAR(7),
        @decomposable CHAR(7) = 'Unknown'
```

```
AS
BEGIN
   SET NOCOUNT ON
   DECLARE @element_ID INT,
               @elementType CHAR(12) = 'Relationship';

         EXECUTE sp_createElement @relationshipName, @elementType, @element_ID OUTPUT;
         EXECUTE @relationship_ID = sp_getSeqNumberRelationship;

   INSERT INTO tblRelationship([relationship_ID], [element_ID], [relationshipName],
                              isIdentifier, degree, decomposable)
                        VALUES(@relationship_ID,  @element_ID, @relationshipName,
                                                   @isIdentifier, @degree, @decomposable);
         RETURN 1;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createRelationshipLink]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 16/January/2014
-- Description:   Create a RelationshipLink object,
--          pointing at the Entity object and
--          the Relationship object which
--          are passed as argument
-- =============================================
CREATE PROCEDURE [dbo].[sp_createRelationshipLink]
         @relationshipName SYSNAME,
         @entityName SYSNAME,
         @link_ID INT OUTPUT,
         @role_sglr SYSNAME,
         @role_plrl SYSNAME
```

```
AS
BEGIN
   SET NOCOUNT ON
   DECLARE @relationshipLinkName SYSNAME;
         DECLARE @relationshipLink_ID INT;
         SET @relationshipLinkName = @relationshipName + '_' + @entityName + '_' + @role_sglr;
   -- Create a generic link object
   EXEC sp_createLink @relationshipLinkName, @link_ID OUTPUT;

   EXEC @relationshipLink_ID = [sp_getSeqNumberRelationshipLink];

   INSERT INTO [dbo].[tblRelationshipLink]
        ([relationshipLink_ID]
                    ,[relationshipLinkName]
        ,[link_ID]
        ,[relationship_ID]
        ,[entityName]
        ,[participation]
        ,[cardinality]
        ,[role_sglr]
                   ,[role_plrl]
        ,[isIdentifier])
    VALUES ( @relationshipLink_ID
                           ,@relationshipLinkName
                           ,@link_ID
         ,dbo.fn_getRelationship_ID(@relationshipName)
         ,@entityName
         ,'Unknown'
         ,'Unknown'
         ,@role_sglr
                           ,@role_plrl
         ,'Unknown')

END

GO
/****** Object:  StoredProcedure [dbo].[sp_createSchema]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
```

```
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                  Peter
-- Create date: 2013_11_08
-- Description:
-- =============================================
CREATE PROCEDURE [dbo].[sp_createSchema](
@schemaName VARCHAR(MAX)
)
AS
BEGIN
   DECLARE @schema_ID INT;

        SET NOCOUNT ON;
        IF NOT EXISTS(SELECT * FROM sys.objects
                WHERE object_id = OBJECT_ID(N'[dbo].[seq_schema_ID]')
                                AND type = 'SO')
        BEGIN
     CREATE SEQUENCE [dbo].[seq_schema_ID]
       AS [int]
      START WITH 1
      INCREMENT BY 1
      MINVALUE 1
      MAXVALUE 2147483647
      CYCLE
      CACHE;
   END;
        SET @schema_ID = NEXT VALUE FOR [seq_schema_ID];
   INSERT INTO tblSchema ([schema_ID], schemaName)
        VALUES(@schema_ID, @schemaName);
        RETURN 1;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_createTernaryRelationship]  Script Date: 15/06/2014
18:45:54 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:                Peter
-- Create date: 2014_01_23
-- Description:    Create NEW Ternary Relationship in the design

-- ============================================
CREATE PROCEDURE [dbo].[sp_createTernaryRelationship]
        @relationshipName SYSNAME,
        @entityName_1 SYSNAME,
        @entityName_2 SYSNAME,
        @entityName_3 SYSNAME,
        @relationship_ID INT OUTPUT,
        @isIdentifier CHAR(7) = 'No',
        @role_1_sglr SYSNAME = NULL,
        @role_1_plrl SYSNAME = NULL,
        @role_2_sglr SYSNAME = NULL,
        @role_2_plrl SYSNAME = NULL,
        @role_3_sglr SYSNAME = NULL,
        @role_3_plrl SYSNAME = NULL
AS
BEGIN
   DECLARE @element_ID INT,
                @RC INT,
                        @degree CHAR(7),
                        @decomposable CHAR(7);

        SET NOCOUNT ON;
        SET @degree = 'ternary';
        SET @decomposable = 'Unknown';
        -- STEP 1: create the Generic Relationship object
   EXEC @RC = [dbo].[sp_createRelationship]
                @relationshipName,
                                @relationship_ID,
                                @isIdentifier,
```

```
                                        @degree,
                                        @decomposable;
        IF @RC = 1
        BEGIN
    -- STEP 2: Create the first Relationship Link and use it to associate the first Entity
          DECLARE @link_ID_1 INT;
          EXEC sp_createRelationshipLink @relationshipName, @entityName_1, @link_ID_1
OUTPUT,
                        @role_1_sglr, @role_1_plrl;

          -- STEP 3: Create the second Relationship Link and use it to associate the second Entity
          DECLARE @link_ID_2 INT;
          EXEC sp_createRelationshipLink @relationshipName, @entityName_2, @link_ID_2
OUTPUT,
                        @role_2_sglr, @role_2_plrl;
          -- STEP 4: Create the third Relationship Link and use it to associate the second Entity
          DECLARE @link_ID_3 INT;
          EXEC sp_createRelationshipLink @relationshipName, @entityName_3, @link_ID_3
OUTPUT,
                        @role_3_sglr, @role_3_plrl;

              EXEC sp_addFunctionalDependencies @relationshipName, @entityName_1,
@entityName_2, @entityName_3
          RETURN 1;
        END
END

GO
/****** Object: StoredProcedure [dbo].[sp_DDL_table] Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 28/January/2014
-- Description:   Generate the SQL data definition
--          language statement to create a table
```

```
--          for the entity specified in the
--          argument
-- ========================================
CREATE PROCEDURE [dbo].[sp_DDL_table]
   @entityName SYSNAME,
         @ddlCreateTable NVARCHAR(4000) OUTPUT
AS
BEGIN
   -- We declare a vaiable for each of the fields in a
         -- record of tblAttribute. The data type of each of
         -- our variables is identical to the data type of the
         -- corresponding field.
   DECLARE
         @attribute_ID [int],
         @element_ID [int],
         @attributeLink_ID [int],
         @attributeName [sysname],
         @identifierDiscriminatorOrdinary [varchar](13),
         @simpleOrComposite [varchar](9),
         @singleOrMultivalued [varchar](10),
         @storedOrDerived [varchar](7),
         @attributeType [varchar](50),
         @dataType [varchar](50),
         @size [int],
         @Nullable [char](7),
         @defaultValue [varchar](max),
         @comment [varchar](max),
         @cardinalityStatus [char](7),
         @cardinality [int],
         @firstInTheList CHAR(5) = 'True';
         SET NOCOUNT ON;
   DECLARE @entity_ID INT;
         SET @entity_ID = dbo.fn_getENtity_ID(@entityName);
         SET @ddlCreateTable = 'CREATE TABLE ' + @entityName + '('
         -- get the list of attributes for this entity and
         -- add them to the CREATE TABLE statement one at a time
         DECLARE attribute_cursor CURSOR FOR
      SELECT DISTINCT [attributeName],
```

```
                    [identifierDiscriminatorOrdinary],
            [simpleOrComposite],
            [singleOrMultivalued],
            [storedOrDerived],
                                        [attributeType],
                                        [dataType],
                                        [size],
                                        [Nullable],
                                        [defaultValue],
                                        [cardinalityStatus],
                                        [cardinality]
              FROM tblAttribute  AS A
              INNER JOIN tblAttributeLink  AS L
              ON A.attributeLink_ID  = L.attributeLink_ID
              INNER JOiN tblElement  AS E
              ON L.element_ID = E.element_ID
              WHERE E.elementName  = @entityName
OPEN attribute_cursor;
      FETCH NEXT FROM attribute_cursor INTO
        @attributeName,
        @identifierDiscriminatorOrdinary,
        @simpleOrComposite,
        @singleOrMultivalued,
        @storedOrDerived,
        @attributeType,
        @dataType,
        @size,
        @Nullable,
        @defaultValue,
        @cardinalityStatus,
        @cardinality

WHILE @@FETCH_STATUS = 0
BEGIN
        IF @firstInTheList = 'True'
              -- For the first attribute in the list, we just
                    -- have to change the value of the variable
              SET @firstInTheList = 'False'
```

```
                ELSE
                        -- This attribute is not the first in the list,
                        -- so put a comma into the string which represents
                        -- the DDL statement.
                    SET @ddlCreateTable = @ddlCreateTable + ', ';

                SET @ddlCreateTable = @ddlCreateTable + @attributeName + ' '
                            + @dataType;



            -- Get the next attribute associated with this
                    -- with this entity
        FETCH NEXT FROM attribute_cursor INTO
                                    @attributeName,
                                    @identifierDiscriminatorOrdinary,
                                    @simpleOrComposite,
                                    @singleOrMultivalued,
                                    @storedOrDerived,
                                    @attributeType,
                                    @dataType,
                                    @size,
                                    @Nullable,
                                    @defaultValue,
                                    @cardinalityStatus,
                                    @cardinality
    END
    CLOSE attribute_cursor
    DEALLOCATE  attribute_cursor
    SET @ddlCreateTable = @ddlCreateTable + ');'
END

GO
/****** Object:  StoredProcedure [dbo].[sp_genQuest_BinaryRelLinkCardinality]    Script Date:
15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
-- =============================================
-- Author:                Peter
-- Create date: 2014_01_20
-- Description:    For each binary relationship,  check
--            if either of the relationship links
--            has Unknown cardinality. If so,
--            print out a question which asks
--            about the cardinality.
-- =============================================

CREATE PROCEDURE [dbo].[sp_genQuest_BinaryRelLinkCardinality]
   @theQuestions VARCHAR(MAX)  OUTPUT
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;
   DECLARE @relationship_ID INT;
        DECLARE @entity_1 SYSNAME
        DECLARE @entity_2 SYSNAME
        DECLARE @entityName_1 SYSNAME
        DECLARE @entityName_2 SYSNAME
        DECLARE @link_1 INT
        DECLARE @link_2 INT
        DECLARE @role_1_sglr SYSNAME
        DECLARE @role_1_plrl SYSNAME;

        SET @theQuestions = '';

        DECLARE db_cursor CURSOR FOR
     SELECT DISTINCT L.relationship_ID,
                        L.relationshipLink_ID,
                        L.[entityName],
                                              L.role_sglr, L.role_plrl
                FROM
                (  SELECT * FROM tblRelationship
                        WHERE [degree] = 'binary'
```

```
                ) AS R
                INNER JOIN
                (   SELECT * FROM tblRelationshipLink
                        WHERE [cardinality] = 'Unknown'
                ) AS L
                ON R.relationship_ID = L.relationship_ID;


    OPEN db_cursor
    FETCH NEXT FROM db_cursor INTO @relationship_ID, @link_1, @entityName_1 ,
                            @role_1_sglr, @role_1_plrl
    WHILE @@FETCH_STATUS = 0
    BEGIN
                SELECT @link_2 = relationshipLink_ID,
                    @entityName_2 = [entityName]
                FROM [dbo].[tblRelationshipLink]
                WHERE relationship_ID = @relationship_ID
                AND   relationshipLink_ID <> @link_1;

                SET @theQuestions = @theQuestions +
                            'Can ' + dbo.fn_addIndefiniteArticle( @entityName_1 )
                            + ' ' + @role_1_plrl
                                                + ' more than one ' + @entityName_2
                                                + '?'
                                                + CHAR(13);   -- This is to get each question on a
separate line.

        FETCH NEXT FROM db_cursor INTO @relationship_ID, @link_1, @entityName_1 ,
                                @role_1_sglr, @role_1_plrl
    END
    CLOSE db_cursor
    DEALLOCATE db_cursor
END
GO
/****** Object:  StoredProcedure [dbo].[sp_genQuest_BinaryRelLinkParticipation]        Script Date:
15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:                Peter
-- Create date: 2014_01_18
-- Description:    For each binary relationship, check
--          if either of the relationship links
--          has Unknown participation. If so,
--          print out a question which asks
--          about the cardinality.
-- Changes
-- 2014_02_05   The procedure now works with entity
--          names rather than entity ID numbers.
--          This is to make the application
--          software easier to use.
-- ============================================
CREATE PROCEDURE [dbo].[sp_genQuest_BinaryRelLinkParticipation]
   @theQuestions VARCHAR(MAX)  OUTPUT
AS
BEGIN
        -- SET NOCOUNT ON added to prevent extra result sets from
        -- interfering with SELECT statements.
        SET NOCOUNT ON;
   DECLARE @relationship_ID INT;
        DECLARE @entityName_1 SYSNAME
        DECLARE @entity_ID_1 INT
        DECLARE @entityName_2 SYSNAME
        DECLARE @entity_ID_2 INT
        DECLARE @link_1 INT
        DECLARE @link_2 INT
        DECLARE @role_1_sglr SYSNAME
        DECLARE @role_1_plrl SYSNAME;

        DECLARE db_cursor CURSOR FOR
      SELECT DISTINCT L.relationship_ID,
                        L.relationshipLink_ID,
                        L.[entityName],
                                          L.role_sglr, L.role_plrl
```

```
            FROM
            (   SELECT * FROM tblRelationship
                    WHERE [degree] = 'binary'
            ) AS R
            INNER JOIN
            (    SELECT * FROM tblRelationshipLink
                     WHERE [participation] = 'Unknown'
            ) AS L
            ON R.relationship_ID = L.relationship_ID;

    SET @theQuestions = '';
    OPEN db_cursor
    FETCH NEXT FROM db_cursor INTO @relationship_ID, @link_1, @entityName_1 ,
                                   @role_1_sglr, @role_1_plrl
    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Get the details of the other entity associated with the
            -- other relationship link in this relationship
            SELECT @link_2 = relationshipLink_ID,
                 @entityName_2 = [entityName]
            FROM [dbo].[tblRelationshipLink]
            WHERE relationship_ID = @relationship_ID
            AND   relationshipLink_ID <> @link_1;

            SET @entity_ID_1 = dbo.[fn_getEntity_ID]( @entityName_1 )
            SET @entity_ID_2 = dbo.[fn_getEntity_ID]( @entityName_2 )

            -- The question needs to be of the form
        -- "Is it permissible that a/an <entity1_name> does not <role_plrl>
            -- any <entity2_name> ?"

            SET @theQuestions = @theQuestions +
                    'Is it permissible that ' + dbo.fn_addIndefiniteArticle( @entityName_1 )
                + ' does not ' + @role_1_plrl
                                    + ' any of the ' +
dbo.fn_getEntityName_plural( @entityName_2 )
                                    + CHAR(13);
```

```
      FETCH NEXT FROM db_cursor INTO @relationship_ID, @link_1, @entityName_1 ,
                                    @role_1_sglr, @role_1_plrl
   END
   CLOSE db_cursor
   DEALLOCATE db_cursor
END
GO
/****** Object:  StoredProcedure [dbo].[sp_genQuest_TernaryRel_FDs]    Script Date: 15/06/2014
18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                  Peter
-- Create date: 2014_01_23
-- Description:    Generate questions about functional
--            dependencies associated with
--            ternary relationships
-- =============================================
CREATE PROCEDURE [dbo].[sp_genQuest_TernaryRel_FDs]
   @theQuestions VARCHAR(MAX)  OUTPUT
AS
BEGIN
   DECLARE @relationshipName SYSNAME,
               @left1_entityName SYSNAME,
               @left2_entityName SYSNAME,
               @right_entityName SYSNAME;
         DECLARE @localQuestions VARCHAR(MAX) = '';
         SET NOCOUNT ON;
         DECLARE db_cursor CURSOR FOR
   SELECT DISTINCT F.relationshipName,
                           F.left1_entityName,
                           F.left2_entityName,
                                       F.right_entityName
         FROM
         (  SELECT * FROM tblRelationship
                  WHERE [degree] = 'ternary'
```

```
            AND decomposable = 'Unknown'
    ) AS R
    INNER JOIN
    (   SELECT * FROM tblFunctionalDependencies
                WHERE [status] = 'Unknown'
    ) AS F
    ON R.relationshipName = F.relationshipName
    ORDER BY relationshipName, left1_entityName;

SET @localQuestions = ' ';
OPEN db_cursor
FETCH NEXT FROM db_cursor INTO @relationshipName,
                                                    @left1_entityName,
                                                    @left2_entityName,
                                                    @right_entityName;
WHILE @@FETCH_STATUS = 0
BEGIN
  IF   @left1_entityName IS NOT NULL
            AND @right_entityName IS NOT NULL
          BEGIN
            IF    @left2_entityName IS NOT NULL
                  BEGIN
            SET @localQuestions = @localQuestions +
                    'Can more than one ' + @right_entityName
                + ' be associated with a given combination of a '
                                    + @left1_entityName
                                    + ' and a ' + @left2_entityName
                                    + '?'
                                    + CHAR(13);
    END
            ELSE
            BEGIN
                    -- There is one attribute on the lhs of the functional dependency
              SET @localQuestions = @localQuestions +
                    'Can more than one ' + @right_entityName
                + ' be associated with a given '
                                    + @left1_entityName
                                    + '?'
```

```
                                                + CHAR(13);
                        END
                    END
        FETCH NEXT FROM db_cursor INTO  @relationshipName,
                                                                    @left1_entityName,
                                                                    @left2_entityName,
                                                                    @right_entityName;
    END
    CLOSE db_cursor
    DEALLOCATE  db_cursor
          SET @theQuestions = @localQuestions;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_getSeqNumberAttribute]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_11_13
-- Description:    Get the next sequence number for tblAttribute.
--             If no sequence exists yet, create one.
-- =============================================
CREATE PROCEDURE [dbo].[sp_getSeqNumberAttribute]
AS
BEGIN
    DECLARE @attribute_ID INT;
          IF NOT EXISTS(SELECT * FROM sys.objects
                    WHERE object_id = OBJECT_ID(N'[dbo].[seq_attribute_ID]')
                                    AND type = 'SO')
          BEGIN
       CREATE SEQUENCE [dbo].[seq_attribute_ID]
         AS [int]
        START WITH 1
        INCREMENT BY 1
```

```
        MINVALUE  1
        MAXVALUE  2147483647
        CYCLE
        CACHE;
    END;
    SET @attribute_ID = NEXT VALUE  FOR [seq_attribute_ID];
          RETURN  @attribute_ID;
END

GO
/******  Object:    StoredProcedure  [dbo].[sp_getSeqNumberAttributeLink]       Script  Date:  15/06/2014
18:45:54 ******/
SET ANSI_NULLS  ON
GO
SET QUOTED_IDENTIFIER  ON
GO
-- =============================================
-- Author:                    Peter
-- Create date: 2013_11_14
-- Description:    Get the next sequence number for an
--             AttributeLink
-- =============================================
CREATE  PROCEDURE  [dbo].[sp_getSeqNumberAttributeLink]
AS
BEGIN
    DECLARE  @attributeLink_ID INT;
    IF NOT EXISTS(SELECT * FROM sys.objects
                    WHERE  object_id = OBJECT_ID(N'[dbo].[seq_attributeLink_ID]')
                                        AND type = 'SO')
          BEGIN
       CREATE  SEQUENCE  [dbo].[seq_attributeLink_ID]
         AS [int]
        START WITH 1
        INCREMENT  BY  1
        MINVALUE  1
        MAXVALUE  2147483647
        CYCLE
        CACHE;
```

```
   END;
         SET @attributeLink_ID = NEXT VALUE  FOR [seq_attributeLink_ID];
         RETURN  @attributeLink_ID;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_getSeqNumberElement]     Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS  ON
GO
SET QUOTED_IDENTIFIER  ON
GO
-- =============================================
-- Author:                  Peter
-- Create date: 13/11/13
-- Description:    Get the next sequence number for tblElement's
--             primary key column. If no sequence has been
--             created, then create one.
-- =============================================
CREATE  PROCEDURE  [dbo].[sp_getSeqNumberElement]
AS
BEGIN
   DECLARE  @element_id  INT;
         SET NOCOUNT  ON;
         IF NOT EXISTS(SELECT * FROM sys.objects
                  WHERE  object_id = OBJECT_ID(N'[dbo].[seq_element_ID]')
                                    AND type = 'SO')
         BEGIN
      CREATE  SEQUENCE  [dbo].[seq_element_ID]
        AS [int]
       START WITH 1
       INCREMENT  BY  1
       MINVALUE  1
       MAXVALUE  2147483647
       CYCLE
       CACHE;
   END;
         SET @element_ID = NEXT VALUE  FOR [seq_element_ID];
```

```
        RETURN @element_ID;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_getSeqNumberEntity]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 13/11/13
-- Description:    Get the next sequence number for tblEntity's
--            primary key column. If no sequence has been
--            created, then create one.
-- =============================================
CREATE PROCEDURE [dbo].[sp_getSeqNumberEntity]
AS
BEGIN
   DECLARE @entity_id INT;
        SET NOCOUNT ON;
        IF NOT EXISTS(SELECT * FROM sys.objects
                WHERE object_id = OBJECT_ID(N'[dbo].[seq_entity_ID]')
                                AND type = 'SO')
        BEGIN
     CREATE SEQUENCE [dbo].[seq_entity_ID]
       AS [int]
      START WITH 1
      INCREMENT BY 1
      MINVALUE 1
      MAXVALUE 2147483647
      CYCLE
      CACHE;
   END

        SET @entity_ID = NEXT VALUE FOR [seq_entity_ID];
        RETURN @entity_ID;
```

```
END

GO
/****** Object:  StoredProcedure [dbo].[sp_getSeqNumberLink]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_12_06
-- Description:    Get the next sequence number for a
--            Link
-- =============================================
CREATE PROCEDURE [dbo].[sp_getSeqNumberLink]
AS
BEGIN
   DECLARE @link_ID INT;
   IF NOT EXISTS(SELECT * FROM sys.objects
                  WHERE object_id = OBJECT_ID(N'[dbo].[seq_link_ID]')
                                    AND type = 'SO')
          BEGIN
      CREATE SEQUENCE [dbo].[seq_link_ID]
        AS [int]
       START WITH 1
       INCREMENT BY 1
       MINVALUE 1
       MAXVALUE 2147483647
       CYCLE
       CACHE;
   END;
          SET @link_ID = NEXT VALUE FOR [seq_link_ID];
          RETURN @link_ID;
END

GO
```

```
/****** Object:  StoredProcedure [dbo].[sp_getSeqNumberNode]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 13/11/13
-- Description:    Get the next sequence number for tblNode's
--            primary key column. If no sequence has been
--            created, then create one.
-- =============================================
CREATE PROCEDURE [dbo].[sp_getSeqNumberNode]
AS
BEGIN
   DECLARE @node_id INT;
          SET NOCOUNT ON;
          IF NOT EXISTS(SELECT * FROM sys.objects
                    WHERE object_id = OBJECT_ID(N'[dbo].[seq_node_ID]')
                                   AND type = 'SO')
          BEGIN
      CREATE SEQUENCE [dbo].[seq_node_ID]
        AS [int]
       START WITH 1
       INCREMENT BY 1
       MINVALUE 1
       MAXVALUE 2147483647
       CYCLE
       CACHE;
   END;
          SET @node_ID = NEXT VALUE FOR [seq_node_ID];
          RETURN @node_ID;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_getSeqNumberRelationship]    Script Date: 15/06/2014 18:45:54 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                  <Author,,Name>
-- Create date: <Create Date,,>
-- Description:    <Description,,>
-- =============================================
CREATE PROCEDURE [dbo].[sp_getSeqNumberRelationship]
AS
BEGIN
   DECLARE @relationship_ID INT;
         IF NOT EXISTS(SELECT * FROM sys.objects
                  WHERE object_id = OBJECT_ID(N'[dbo].[seq_relationship_ID]')
                                    AND type = 'SO')
         BEGIN
      CREATE SEQUENCE [dbo].[seq_relationship_ID]
        AS [int]
       START WITH 1
       INCREMENT BY 1
       MINVALUE 1
       MAXVALUE 2147483647
       CYCLE
       CACHE;
   END;
         SET @relationship_ID = NEXT VALUE FOR [seq_relationship_ID];
         RETURN @relationship_ID;
END

GO
/****** Object:  StoredProcedure [dbo].[sp_getSeqNumberRelationshipLink]    Script Date: 15/06/2014
18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
```

```
-- Author:                  Peter
-- Create date: 2014_01_16
-- Description:    Get the next sequence number for an
--            AttributeLink
-- ============================================
CREATE PROCEDURE [dbo].[sp_getSeqNumberRelationshipLink]
AS
BEGIN
   DECLARE @relationshipLink_ID INT;
   IF NOT EXISTS(SELECT * FROM sys.objects
                  WHERE object_id = OBJECT_ID(N'[dbo].[seq_relationshipLink_ID]')
                                    AND type = 'SO')
          BEGIN
      CREATE SEQUENCE [dbo].[seq_relationshipLink_ID]
        AS [int]
       START WITH 1
       INCREMENT BY 1
       MINVALUE 1
       MAXVALUE 2147483647
       CYCLE
       CACHE;
   END;
          SET @relationshipLink_ID = NEXT VALUE FOR [seq_relationshipLink_ID];
          RETURN @relationshipLink_ID;
END

GO
/****** Object: UserDefinedFunction [dbo].[fn_addIndefiniteArticle]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:                  Peter
-- Create date: 2014_01_19
-- Description:    the argument is a noun. The function
--            adds an indefinite article to the
```

```
--          noun
-- ============================================
CREATE FUNCTION [dbo].[fn_addIndefiniteArticle]
(
   @noun VARCHAR(MAX)
)
RETURNS VARCHAR(MAX)
AS
BEGIN
   DECLARE @firstChar CHAR(1) = SUBSTRING(@noun, 1, 1);
         DECLARE @theResult VARCHAR(MAX) =
         CASE
            WHEN @firstChar IN ('a', 'e', 'i', 'o', 'u')
                  THEN 'an ' + @noun
            ELSE 'a ' + @noun
         END
         RETURN @theResult
END

GO
/****** Object:  UserDefinedFunction [dbo].[fn_getAttributeName]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- ============================================
-- Author:                Peter
-- Create date: 2013_12_16
-- Description:    This function is passed as an argument
--          an attribute ID number. The function
--          find the unique record associated with
--          this attribute and returns the name of
--          the attribute.
-- ============================================
CREATE FUNCTION [dbo].[fn_getAttributeName]
(
```

```
   @attribute_ID INT
)
RETURNS SYSNAME
AS
BEGIN
        DECLARE @attributeName SYSNAME;
        SELECT @attributeName = attributeName FROM tblAttribute
                        WHERE [attribute_ID] = @attribute_ID;
        RETURN @attributeName;
END



GO
/****** Object:  UserDefinedFunction [dbo].[fn_getElement_ID]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2013_12_16
-- Description:    This function is passed the name
--            of an element. It returns the
--            element's ID.
-- =============================================
CREATE FUNCTION [dbo].[fn_getElement_ID]
(
   @elementName SYSNAME
)
RETURNS INT
AS
BEGIN
        DECLARE @element_ID INT;
        SELECT @element_ID = Element_ID FROM tblElement
                        WHERE [elementName] = @elementName;
        RETURN @element_ID;
END
```

```
GO
/****** Object:  UserDefinedFunction [dbo].[fn_getElementidOfEntity]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:		Peter
-- Create date: 2013_11_14
-- Description:
-- =============================================
CREATE FUNCTION [dbo].[fn_getElementidOfEntity]
(
   @entity_ID INT
)
RETURNS INT
AS
BEGIN
	DECLARE @element_ID INT;
	SELECT @element_ID = Element_ID FROM tblEntity
				WHERE [entity_ID] = @entity_ID;
	RETURN @element_ID;
END

GO
/****** Object:  UserDefinedFunction [dbo].[fn_getElementName]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:		Peter
-- Create date: 2013_11_14
-- Description:	This finctoin is passed as an argument
```

```
--          an element ID number. The function
--          find the unique record associated with
--          this element and returns the name of
--          the element.
-- ============================================
CREATE FUNCTION [dbo].[fn_getElementName]
(
   @element_ID INT
)
RETURNS SYSNAME
AS
BEGIN
          DECLARE @elementName SYSNAME;
          SELECT @elementName = elementName FROM tblElement
                              WHERE [element_ID] = @element_ID;
          RETURN @elementName;
END


GO
/****** Object:  UserDefinedFunction [dbo].[fn_getEntity_ID]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- ============================================
-- Author:                Peter
-- Create date: 2014_01_14
-- Description:    Find the entity ID for the
--          entity name given as an argument
-- ============================================
CREATE FUNCTION [dbo].[fn_getEntity_ID]
(
   @entityName SYSNAME
)
RETURNS INT
AS
```

```
BEGIN
	DECLARE @entity_ID INT;

	SELECT @entity_ID = [entity_ID]
	FROM tblEntity
	WHERE entityName = @entityName;

	-- Return the result of the function
	RETURN @entity_ID

END

GO
/****** Object: UserDefinedFunction [dbo].[fn_getEntityName] Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:		Peter
-- Create date: 2013_11_14
-- Description:	This finctoin is passed as an argument
--		an element ID number. The function
--		find the unique record associated with
--		this element and returns the name of
--		the element.
-- =============================================
CREATE FUNCTION [dbo].[fn_getEntityName]
(
  @entity_ID INT
)
RETURNS SYSNAME
AS
BEGIN
	DECLARE @entityName SYSNAME;
	SELECT @entityName = entityName
```

```
	FROM tblEntity
	WHERE [entity_ID] = @entity_ID;
	RETURN @entityName;
END


GO
/****** Object:  UserDefinedFunction [dbo].[fn_getEntityName_plural]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- =============================================
-- Author:		Peter
-- Create date: 2013_11_14
-- Description:	This function is passed as an argument
--		an element ID number. The function
--		find the unique record associated with
--		this element and returns the plural
--		version of the name of
--		the element.
-- =============================================
CREATE FUNCTION [dbo].[fn_getEntityName_plural]
(
  @entityName SYSNAME
)
RETURNS SYSNAME
AS
BEGIN
	DECLARE @entityName_plural SYSNAME;
	SELECT @entityName_plural = entityName_plural
	FROM tblEntity
	WHERE [entityName] = @entityName;
	RETURN @entityName_plural;
END
```

```
GO
/****** Object:  UserDefinedFunction [dbo].[fn_getRelationship_ID]    Script Date: 15/06/2014 18:45:54
******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Peter
-- Create date: 2014_01_16
-- Description:    Get relationship_ID from
--            tblRelationship using the
--            relationshipName as the way
--            of identifying the record.
-- =============================================
CREATE FUNCTION [dbo].[fn_getRelationship_ID]
(
        @relationshipName SYSNAME
)
RETURNS INT
AS
BEGIN
        DECLARE @relationship_ID INT;
   SELECT @relationship_ID = relationship_ID
        FROM tblRelationship
        WHERE relationshipName = @relationshipName;

        RETURN @relationship_ID;
END

GO
/****** Object:    UserDefinedFunction [dbo].[fn_getRelationshipName]        Script Date: 15/06/2014
18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
-- ============================================
-- Author:                Peter
-- Create date: 2014_01_16
-- Description:    Get relationshipName from
--            tblRelationship using the
--            relationship_ID as the way
--            of identifying the record.
-- ============================================
CREATE FUNCTION [dbo].[fn_getRelationshipName]
(
        @relationship_ID INT
)
RETURNS SYSNAME
AS
BEGIN
        DECLARE @relationshipName SYSNAME;
   SELECT @relationshipName = relationshipName
        FROM tblRelationship
        WHERE relationship_ID = @relationship_ID;

        RETURN @relationshipName;
END

GO
/****** Object:  Table [dbo].[tblAssociativeEntity]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblAssociativeEntity](
        [associativeEntity_ID] [int] NOT NULL,
        [entity_ID] [int] NULL,
        [relationship_ID] [int] NULL,
PRIMARY KEY CLUSTERED
(
        [associativeEntity_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
```

```
) ON [PRIMARY]

GO
/****** Object:  Table [dbo].[tblAttribute]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblAttribute](
	[attribute_ID] [int] NOT NULL,
	[element_ID] [int] NULL,
	[attributeLink_ID] [int] NULL,
	[attributeName] [sysname] NOT NULL,
	[identifierDiscriminatorOrdinary] [varchar](13) NULL,
	[simpleOrComposite] [varchar](9) NULL,
	[singleOrMultivalued] [varchar](10) NULL,
	[storedOrDerived] [varchar](7) NULL,
	[attributeType] [varchar](50) NULL,
	[dataType] [varchar](50) NULL,
	[size] [int] NULL,
	[Nullable] [char](7) NULL,
	[defaultValue] [varchar](max) NULL,
	[comment] [varchar](max) NULL,
	[cardinalityStatus] [char](7) NULL,
	[cardinality] [int] NULL,
 CONSTRAINT [PK__tblAttri__9093DDA3A673BD19] PRIMARY KEY CLUSTERED
(
	[attribute_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblAttributeLink]    Script Date: 15/06/2014 18:45:54 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING OFF
GO
CREATE TABLE [dbo].[tblAttributeLink](
	[attributeLink_ID] [int] NOT NULL,
	[attribute_ID] [int] NULL,
	[link_ID] [int] NULL,
	[element_ID] [int] NULL,
	[attributeLinkName] [varchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[attributeLink_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object: Table [dbo].[tblCategory] Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblCategory](
	[category_ID] [int] NOT NULL,
	[node_ID] [int] NULL,
	[categorySL_ID] [int] NULL,
	[label] [varchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[category_ID] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblCategoryGL]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblCategoryGL](
	[categoryGL_ID] [int] NOT NULL,
	[category_ID] [int] NULL,
	[entity_ID] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[categoryGL_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/****** Object:  Table [dbo].[tblCategorySL]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblCategorySL](
	[categorySL_ID] [int] NOT NULL,
	[category_ID] [int] NULL,
	[entity_ID] [int] NULL,
	[specialisationLink_ID] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[categorySL_ID] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/****** Object:  Table [dbo].[tblDirectInheritanceLink]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblDirectInheritanceLink](
	[directInheritanceLink_ID] [int] NOT NULL,
	[link_ID] [int] NULL,
	[entity_ID] [int] NULL,
	[role] [varchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[directInheritanceLink_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblElement]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblElement](
	[element_ID] [int] NOT NULL,
	[node_ID] [int] NULL,
	[elementName] [sysname] NOT NULL,
```

```
	[elementType] [char](12) NOT NULL,
 CONSTRAINT [PK__tblEleme__38BB8593A407E185]  PRIMARY  KEY CLUSTERED
(
	[element_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING  OFF
GO
/****** Object:  Table [dbo].[tblEntity]    Script Date: 15/06/2014 18:45:54  ******/
SET ANSI_NULLS  ON
GO
SET QUOTED_IDENTIFIER  ON
GO
SET ANSI_PADDING  ON
GO
CREATE  TABLE  [dbo].[tblEntity](
	[entity_ID] [int] NOT NULL,
	[element_ID] [int] NULL,
	[entityName] [sysname] NOT NULL,
	[entityName_plural]  [sysname] NULL,
	[isWeak] [char](7) NULL,
 CONSTRAINT [PK__tblEntit__AF9891AFB00C3F0A]  PRIMARY  KEY CLUSTERED
(
	[entity_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING  OFF
GO
/****** Object:  Table [dbo].[tblFunctionalDependencies]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS  ON
GO
SET QUOTED_IDENTIFIER  ON
```

```
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblFunctionalDependencies](
	[FD_ID] [int] IDENTITY(1,1) NOT NULL,
	[relationshipName] [sysname] NOT NULL,
	[left1_entityName] [sysname] NOT NULL,
	[left2_entityName] [sysname] NULL,
	[right_entityName] [sysname] NOT NULL,
	[status] [char](9) NOT NULL,
 CONSTRAINT [PK_tbl_functionalDepdendencies] PRIMARY KEY CLUSTERED
(
	[FD_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object: Table [dbo].[tblGeneralizationLink] Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblGeneralizationLink](
	[generalizationLink_ID] [int] NOT NULL,
	[link_ID] [int] NULL,
	[role] [varchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[generalizationLink_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

```
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblInheritance]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblInheritance](
	[inheritance_ID] [int] NOT NULL,
	[node_ID] [int] NULL,
	[inheritanceGL_ID] [int] NULL,
	[label] [varchar](max) NULL,
	[disjointness] [char](8) NULL,
PRIMARY KEY CLUSTERED
(
	[inheritance_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblInheritanceGL]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblInheritanceGL](
	[inheritanceGL_ID] [int] NOT NULL,
	[inheritance_ID] [int] NULL,
	[generalizationLink_ID] [int] NULL,
	[entity_ID] [int] NULL,
```

```
	[completeness] [char](7) NULL,
PRIMARY KEY CLUSTERED
(
	[inheritanceGL_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblInheritanceSL]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblInheritanceSL](
	[inheritanceSL_ID] [int] NOT NULL,
	[inheritance_ID] [int] NULL,
	[specializaitonLink_ID] [int] NULL,
	[entity_ID] [int] NULL,
	[completeness] [char](7) NULL,
PRIMARY KEY CLUSTERED
(
	[inheritanceSL_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblLink]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblLink](
	[link_ID] [int] NOT NULL,
	[schema_ID] [int] NULL,
	[linkName] [varchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[link_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object: Table [dbo].[tblNode] Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblNode](
	[node_ID] [int] NOT NULL,
	[schema_ID] [int] NULL,
	[identity] [varchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[node_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
```

```
/****** Object:  Table [dbo].[tblRelationship]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblRelationship](
	[relationship_ID] [int] NOT NULL,
	[element_ID] [int] NULL,
	[relationshipName] [sysname] NOT NULL,
	[isIdentifier] [char](7) NULL,
	[degree] [char](7) NULL,
	[decomposable] [char](7) NULL,
 CONSTRAINT [PK__tblRelat__C0CED15C45EAE21E] PRIMARY KEY CLUSTERED
(
	[relationship_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[tblRelationshipLink]    Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblRelationshipLink](
	[relationshipLink_ID] [int] NOT NULL,
	[relationshipLinkName] [sysname] NOT NULL,
	[link_ID] [int] NULL,
	[relationship_ID] [int] NULL,
	[entityName] [sysname] NULL,
	[participation] [char](7) NULL,
```

```
	[cardinality] [char](7) NULL,
	[role_sglr] [sysname] NULL,
	[role_plrl] [sysname] NULL,
	[isIdentifier] [char](7) NULL,
PRIMARY KEY CLUSTERED
(
	[relationshipLink_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object: Table [dbo].[tblSchema] Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[tblSchema](
	[schema_ID] [int] NOT NULL,
	[schemaName] [sysname] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[schema_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/****** Object: Table [dbo].[tblSpecializationLink] Script Date: 15/06/2014 18:45:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[tblSpecializationLink](
```

```
	[specializationLink_ID]  [int] NOT NULL,
	[link_ID] [int] NULL,
	[role] [varchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[specializationLink_ID]  ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]  TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  View [dbo].[vw_unknown_Relationship_Link_cardinality]    Script Date: 15/06/2014
18:45:54 ******/
SET ANSI_NULLS  ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[vw_unknown_Relationship_Link_cardinality]
AS
SELECT E.entityName, E.entityName_plural,
U.relationshipLinkName,  U.role_sglr, U.role_plrl
FROM tblEntity  AS E
INNER JOIN
(
  SELECT R.relationshipName, RLU.*
  FROM
	( SELECT * FROM tblRelationship
	  WHERE [degree] = 'binary'
	) AS R
  INNER JOIN
  ( SELECT * FROM tblRelationshipLink
    WHERE cardinality = 'Unknown'
	 ) AS RLU
	 ON R.relationship_ID = RLU.relationship_ID
) AS U
```

```
ON E.[entityName] = U.[entityName]


GO
ALTER TABLE [dbo].[tblAttributeLink] WITH NOCHECK ADD CONSTRAINT [FK_tblAttributeLink_tblLink] FOREIGN KEY([link_ID])
REFERENCES [dbo].[tblLink] ([link_ID])
GO
ALTER TABLE [dbo].[tblAttributeLink] NOCHECK CONSTRAINT [FK_tblAttributeLink_tblLink]
GO
ALTER TABLE [dbo].[tblElement] WITH NOCHECK ADD CONSTRAINT [FK_tblElement_tblNode] FOREIGN KEY([node_ID])
REFERENCES [dbo].[tblNode] ([node_ID])
GO
ALTER TABLE [dbo].[tblElement] NOCHECK CONSTRAINT [FK_tblElement_tblNode]
GO
ALTER TABLE [dbo].[tblLink] WITH NOCHECK ADD CONSTRAINT [FK_tblLink_tblSchema] FOREIGN KEY([schema_ID])
REFERENCES [dbo].[tblSchema] ([schema_ID])
GO
ALTER TABLE [dbo].[tblLink] NOCHECK CONSTRAINT [FK_tblLink_tblSchema]
GO
ALTER TABLE [dbo].[tblNode] WITH NOCHECK ADD CONSTRAINT [FK_tblNode_tblSchema] FOREIGN KEY([schema_ID])
REFERENCES [dbo].[tblSchema] ([schema_ID])
GO
ALTER TABLE [dbo].[tblNode] NOCHECK CONSTRAINT [FK_tblNode_tblSchema]
GO
ALTER TABLE [dbo].[tblRelationshipLink] WITH NOCHECK ADD CONSTRAINT [FK_tblRelationshipLink_tblLink] FOREIGN KEY([link_ID])
REFERENCES [dbo].[tblLink] ([link_ID])
GO
ALTER TABLE [dbo].[tblRelationshipLink] NOCHECK CONSTRAINT [FK_tblRelationshipLink_tblLink]
GO
USE [master]
GO
```

```
ALTER DATABASE [EERMM] SET READ_WRITE
GO
```

(b) Inserting a new element script

```
USE [EERMM]
DELETE tblAttribute;
DELETE tblAttributeLink
DELETE tblElement;
DELETE tblLink;
DELETE tblNode;
DELETE tblEntity;
DELETE tblRelationship;
DELETE tblRelationshipLink;

IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_attribute_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_attribute_ID];
IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_attributeLink_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_attributeLink_ID];
IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_element_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_element_ID];
IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_entity_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_entity_ID];
IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_link_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_link_ID];
IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_node_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_node_ID];
IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_relationship_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_relationship_ID];
IF EXISTS( SELECT * FROM sys.objects
        WHERE object_id = OBJECT_ID(N"[seq_relationshipLink_ID]") AND type = 'sO")
   DROP SEQUENCE [seq_relationshipLink_ID];
```

```
GO

DECLARE @RC int
DECLARE @entityName_Emp sysname = “Employee”
DECLARE @entityName_Dept sysname = “Department”
DECLARE @entityName_Dependent SYSNAME = “Dependent”
DECLARE @entity_ID_Emp int
DECLARE @entity_ID_Dept int
DECLARE @isWeak char(7) = “No”;

EXECUTE @RC = [dbo].[sp_createEntity]
  @entityName_Emp
 ,@entity_ID_Emp OUTPUT
 ,@isWeak;

EXECUTE @RC = [dbo].[sp_createEntity]
  @entityName_Dependent
 ,@entity_ID_Emp OUTPUT
 ,@isWeak = “Unknown”;

EXECUTE @RC = [dbo].[sp_createEntity]
  @entityName_Dept
 ,@entity_ID_Dept OUTPUT
 ,@isWeak = “No”;


EXECUTE @RC = [dbo].[sp_createAttribute] “EmployeeNumber”, @entityName_Emp;

EXECUTE @RC = [dbo].[sp_createAttribute] “DepartmentNumber”, @entityName_Dept;

DECLARE @relationshipName sysname
DECLARE @role_1_sglr sysname
DECLARE @role_1_plrl sysname
DECLARE @role_2_sglr sysname
DECLARE @role_2_plrl sysname
DECLARE @relationship_ID int
DECLARE @isIdentifier char(7)
```

```
SET @relationshipName = “Employment”;
SET @role_1_sglr = “works in”
SET @role_1_plrl = “work in”
SET @role_2_sglr = “employs”
SET @role_2_plrl = “employ”
SET @isIdentifier = “No”;

EXECUTE @RC = [dbo].[sp_createBinaryRelationship]
   @relationshipName
  ,@entityName_Emp
  ,@entityName_Dept
  ,@relationship_ID OUTPUT
  ,@isIdentifier
  ,@role_1_sglr ,@role_1_plrl
  ,@role_2_sglr ,@role_2_plrl

SET @relationshipName = “Dependency”;
SET @role_1_sglr = “has”
SET @role_1_plrl = “have”
SET @role_2_sglr = “belongs to”
SET @role_2_plrl = “belong to”

SET @isIdentifier = “Unknown”;

EXECUTE @RC = [dbo].[sp_createBinaryRelationship]
   @relationshipName
  ,@entityName_Emp
  ,@entityName_Dependent
  ,@relationship_ID OUTPUT
  ,@isIdentifier
  ,@role_1_sglr ,@role_1_plrl
  ,@role_2_sglr ,@role_2_plrl
GO

DECLARE @theQuestions VARCHAR(MAX);
EXEC sp_genQuest_BinaryRelLinkParticipation  @theQuestions OUTPUT;
PRINT @theQuestions;
EXEC [dbo].[sp_genQuest_BinaryRelLinkCardinality]  @theQuestions OUTPUT
```

```
PRINT @theQuestions;
```

(c) Update value scripts

Update the value of "Cardinality" to "One"

```
USE EERMM
DECLARE @RC int
DECLARE @ cardinality CHAR(7),
      @entityName SYSNAME,
      @relationshipName SYSNAME,
      @relationshipLinkName SYSNAME,
      @role_sglr SYSNAME,
      @role_plrl SYSNAME;

EXECUTE @RC= [dbo].[sp_updateRelationshipLink_ cardinality]
 @ cardinality = 'total',
 @entityName = 'Employee',
 @relationshipName = 'Employment',
 @relationshipLinkName = 'Employment_Employee_works in',
 @role_sglr = 'works in',
 @role_plrl = 'work in';
```

Update the value of "Participation" to "Total"

```
USE EERMM
DECLARE @RC int
DECLARE @participation CHAR(7),
      @entityName SYSNAME,
      @relationshipName SYSNAME,
      @relationshipLinkName SYSNAME,
      @role_sglr SYSNAME,
      @role_plrl SYSNAME;

EXECUTE @RC= [dbo].[sp_updateRelationshipLink_participation]
 @participation = 'total',
 @entityName = 'Employee',
 @relationshipName = 'Employment',
 @relationshipLinkName = 'Employment_Employee_works in',
 @role_sglr = 'works in',
```

```
 @role_plrl = 'work in';

USE EmpsAndDepts
UPDATE Employment_Bridge
SET KeyDepartment = NULL
WHERE KeyEmployee = 11

SELECT relationshipLinkName,relationship_ID,entityName,participation FROM
EERMM.dbo.tblRelationshipLink
WHERE relationshipLinkName = “Employment_Employee_works in”
SELECT * FROM dbo.Employment_Bridge
```

Update the value of “isWeak”

```
USE EERMM
UPDATE [dbo].[tblEntity]
SET [isWeak] = “Yes”
WHERE entityName = “Employee”

SELECT * FROM tblEntity
SELECT attribute_ID, attributeName, identifierDiscriminatorOrdinary
FROM tblAttribute

USE EERMM
UPDATE [dbo].[tblEntity]
SET [isWeak] = “No”
WHERE entityName = “Employee”

SELECT * FROM tblEntity
SELECT attribute_ID, attributeName, identifierDiscriminatorOrdinary
FROM tblAttribute
```

(d) Triggers

Trigger tr_Cardinality_Update

```
USE [EERMM]
GO
/****** Object:  Trigger [dbo].[tr_Cardinality_Update]    Script Date: 2015/5/16 21:44:42 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
  ALTER TRIGGER [dbo].[tr_Cardinality_Update]
  ON [dbo].[tblRelationshipLink]
  AFTER UPDATE
AS
BEGIN
   DECLARE @databaseName SYSNAME;
         DECLARE @schema_ID INT;
         DECLARE @link_ID INT;
   DECLARE @oldCardVal VARCHAR(7);
         DECLARE @newCardVal VARCHAR(7);
         DECLARE @relationship_ID INT;
         DECLARE @relationshipLink_ID INT;
         DECLARE @degree VARCHAR(7);
         DECLARE @entityName SYSNAME;
         DECLARE @other_entityName SYSNAME;
         DECLARE @otherLink_cardinality VARCHAR(7);
         DECLARE @sql NVARCHAR(MAX);
         DECLARE @relationshipName SYSNAME;
         DECLARE @bridgeTable SYSNAME;
         DECLARE @foreignKeyColumn SYSNAME;
         DECLARE @tblNew AS TABLE(
           [cardinality] VARCHAR(7),
           [relationship_ID] INT,
           [relationshipLink_ID] INT,
                [entityName] SYSNAME,
                [link_ID] INT
         );
         DECLARE @tblOld AS TABLE(
```

```
    [cardinality] VARCHAR(7),
    [relationship_ID] INT,
    [relationshipLink_ID] INT,
        [entityName] SYSNAME,
        [link_ID] INT
);
DECLARE @newline AS NVARCHAR(2);
SET @newline = NCHAR(13) + NCHAR(10);

IF @@ROWCOUNT = 0 RETURN;
SET NOCOUNT ON;
IF NOT UPDATE([cardinality]) RETURN;

INSERT INTO @tblNew( [cardinality], [relationship_ID], [relationshipLink_ID],
        [entityName], [link_ID])
SELECT [cardinality], [relationship_ID], [relationshipLink_ID],
        [entityName], [link_ID]
FROM inserted;

INSERT INTO @tblOld( [cardinality], [relationship_ID], [relationshipLink_ID],
        [entityName], [link_ID])
SELECT [cardinality], [relationship_ID], [relationshipLink_ID],
        [entityName], [link_ID]
FROM deleted;

--The "inserted" table contains the new value for the row which has been updated.
SELECT @newCardVal = [cardinality],
    @relationship_ID = [relationship_ID],
    @relationshipLink_ID = [relationshipLink_ID],
        @entityName = [entityName],
        @link_ID = [link_ID]
FROM @tblNew;

-- The "deleted" table contains the old value for the row which has been updated.
SELECT @oldCardVal = [cardinality]
FROM @tblOld;

-- For our present purposes, we are only interested in binary relationships.
```

```
    -- Hence we need to find the degree of the relationship from the corresponding
    -- record in tblRelationship.

    SELECT @degree = degree, @relationshipName = relationshipName
    FROM tblRelationship
    WHERE relationship_ID = @relationship_ID;

    -- Exit if this is not a binary relationship
    IF @degree <> "binary" RETURN;

    -- Find the other relationship link associated with this relationship link's
    -- relationship.

    SELECT @otherLink_cardinality = cardinality,
        @other_entityName = entityName
    FROM tblRelationshipLink
    WHERE relationship_ID = @relationship_ID
    AND relationshipLink_ID <> @relationshipLink_ID;

    IF @oldCardVal = "Unknown" AND @newCardVal = "One"
    BEGIN
-- Check if it is possible to change the relationship from many-to-many
            -- to one-to-many
            IF @otherLink_cardinality = "Unknown" OR @otherLink_cardinality = "Many"
            BEGIN
               -- So the entity associated with this relationship link is the "One" end
                    -- of a One-to-Many, or One-to-Unknown, relationship.
                    -- We add a foreign key column to the other table which references
                    -- the primary key column of this table.

                    SET @foreignKeyColumn = " [key" + @other_entityName + "] "

                    -- Get the name of the target database from tblSchema
                    SELECT @databaseName = tblSchema.schemaName
                    FROM tblSchema INNER JOIN tblLink
                    ON tblSchema.[schema_ID] = tblLink.[schema_ID]
                    WHERE tblLink.link_ID = @link_ID;
```

```
                    SET @sql = "USE " + @databaseName + ";" + @newline;
                    SET @sql = @sql + "ALTER TABLE [" + @entityName + "] "
                    SET @sql = @sql + " ADD " + @foreignKeyColumn + " INT NULL;";
                    EXEC sp_executesql @sql;
        PRINT @databaseName;
                -- Next, we add data to the new column from the bridge table

                    SET @bridgeTable = "[" + @relationshipName + "_Bridge]";

                    SET @sql =      "USE [" + @databaseName + "];" + @newline;
                    SET @sql = @sql + "UPDATE [" + @entityName + "] " + @newline;
                    SET @sql = @sql + 'sET " + @foreignKeyColumn + " = ";
                    SET @sql = @sql + @bridgeTable + "." + @foreignKeyColumn + @newline;
                    SET @sql = @sql + " FROM [" + @entityName + "]" + @newline;
                    SET @sql = @sql + " INNER JOIN " + @bridgeTable + @newline;
                    SET @sql = @sql + " ON " + @entityName + ".[key" + @entityName;
                    SET @sql = @sql + "] = " + @bridgeTable + ".[key" + @entityName + "];"+
@newline;
                    EXEC sp_executesql @sql;

            END
        END
END
```

Trigger "tr_Participation_Update"

```
USE [EERMM]
GO
/****** Object:  Trigger [dbo].[tr_Participation_Update]    Script Date: 2015/5/16 21:45:29 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER TRIGGER [dbo].[tr_Participation_Update]
  ON [dbo].[tblRelationshipLink]
  AFTER UPDATE
AS
BEGIN
 DECLARE @databaseName SYSNAME;
```

```
 DECLARE @schema_IN INT;
 DECLARE @link_ID INT;
 DECLARE @oldPartVal VARCHAR(7);
 DECLARE @newPartVal VARCHAR(7);
 DECLARE @relationship_ID INT;
 DECLARE @relationshipLink_ID INT;
 DECLARE @degree VARCHAR(7);
 DECLARE @entityName SYSNAME;
 DECLARE @other_entityName SYSNAME;
 DECLARE @otherLink_participation VARCHAR(7);
 DECLARE @sql NVARCHAR(MAX);
 DECLARE @relationshipName SYSNAME;
 DECLARE @bridgeTable SYSNAME;
 DECLARE @foreignKeyColumn SYSNAME;
 DECLARE @tblNew AS TABLE(
        [participation] VARCHAR(7),
        [relationship_ID] INT,
        [relationshipLink_ID] INT,
       [entityName] SYSNAME,
                  [link_ID] INT
     );
DECLARE @tblOld AS TABLE(
        [participation] VARCHAR(7),
        [relationship_ID] INT,
        [relationshipLink_ID] INT,
          [entityName] SYSNAME,
                                [link_ID] INT
     );
DECLARE @newline AS NVARCHAR(2);
SET @newline = NCHAR(13) + NCHAR(10);

IF @@ROWCOUNT = 0 RETURN;
  SET NOCOUNT ON;
  IF NOT UPDATE([participation]) RETURN;

  INSERT INTO @tblNew( [participation], [relationship_ID],
               [relationshipLink_ID], [entityName], [link_ID])
  SELECT [participation], [relationship_ID], [relationshipLink_ID], [entityName], [link_ID]
```

```
FROM inserted;

INSERT INTO @tblOld( [participation], [relationship_ID],
            [relationshipLink_ID], [entityName], [link_ID])
SELECT [participation], [relationship_ID], [relationshipLink_ID], [entityName],[link_ID]
FROM deleted;

SELECT @newPartVal = [participation], @relationship_ID = [relationship_ID],
    @relationshipLink_ID = [relationshipLink_ID],
    @entityName = [entityName],
                @link_ID = [link_ID]
FROM @tblNew;


SELECT @oldPartVal = [participation]
FROM @tblOld;

SELECT @relationshipName = relationshipName
FROM tblRelationship
WHERE relationship_ID = @relationship_ID;

SELECT @other_entityName = entityName
FROM tblRelationshipLink
WHERE relationship_ID = @relationship_ID
AND relationshipLink_ID <> @relationshipLink_ID;

IF @oldPartVal = "Unknown" AND @newPartVal = "Total"
BEGIN


SET @foreignKeyColumn = "key" + @other_entityName

                    SELECT @databaseName = tblSchema.schemaName
                    FROM tblSchema INNER JOIN tblLink
                    ON tblSchema.[schema_ID] = tblLink.[schema_ID]
                    WHERE tblLink.link_ID = @link_ID;

                SET @bridgeTable = @relationshipName + "_Bridge";
```

```
        SET @sql = "USE [" + @databaseName + "];" + @newline;
        SET @sql = @sql + "ALTER TABLE " + @bridgeTable + @newline;
        SET @sql = @sql + "ALTER COLUMN " + @foreignKeyColumn + " INT NOT NULL;";

        EXEC sp_executesql @sql;

    END
  END
```

Trigger "tr_isWeak_Update"

```
USE [EERMM]
GO
/****** Object:  Trigger [dbo].[tr_isWeak_Update]    Script Date: 2015/5/16 21:46:27 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER TRIGGER [dbo].[tr_isWeak_Update]
  ON [dbo].[tblEntity]
  AFTER UPDATE
AS
BEGIN
  DECLARE @entity_ID INT;
  DECLARE @element_ID INT;
  DECLARE @attribute_ID INT;
  DECLARE @IsWeak VARCHAR(7);
  DECLARE @oldIsWeak VARCHAR(7);
  DECLARE @newIsWeak VARCHAR(7);
  DECLARE @entityName SYSNAME;
  DECLARE @identifierDiscriminatorOrdinary VARCHAR(13);
  DECLARE @tblNew AS TABLE (
        [isWeak] VARCHAR (7),
                [element_ID] INT,
                [entityName] SYSNAME
                );
  DECLARE @tblOld AS TABLE (
```

```
    [isWeak] VARCHAR  (7),
                [element_ID] INT,
                [entityName] SYSNAME
                );

INSERT INTO @tblNew( [isWeak], [element_ID],[entityName])
SELECT [isWeak], [element_ID]          ,[entityName]
FROM inserted;

INSERT INTO @tblOld( [isWeak], [element_ID],[entityName])
SELECT [isWeak], [element_ID],[entityName]
FROM deleted;

 --The "inserted" table contains the new value for the row which has been updated.
SELECT @newIsWeak = [isWeak], @element_ID = [element_ID], @entityName = [entityName]
FROM @tblNew;

SELECT @oldIsWeak = [isWeak]
FROM @tblOld;


SELECT @attribute_ID = tblAttribute.attribute_ID
FROM tblEntity
INNER JOIN
tblAttributeLink
ON
tblAttributeLink.element_ID  = tblEntity.element_ID
INNER JOIN
tblAttribute
ON
tblAttribute.attributeLink_ID  = tblAttributeLink.attributeLink_ID
WHERE tblEntity.entityName  = @entityName
AND tblAttribute.identifierDiscriminatorOrdinary  = “Unknown”;

PRINT @attribute_ID;
IF @oldIsWeak = “Unknown” AND @newIsWeak = “Yes”
BEGIN
UPDATE [dbo].[tblAttribute]
```

```
 SET [identifierDiscriminatorOrdinary]  = “Discriminator”
 WHERE attribute_ID = @attribute_ID;
 END

 IF @oldIsWeak = “Unknown” AND @newIsWeak = “No”
 BEGIN
 UPDATE [dbo].[tblAttribute]
 SET [identifierDiscriminatorOrdinary]  = “Identifier”
 WHERE attribute_ID = @attribute_ID;
 END
END
```

(e) **Stored procedures**

sp_updataRelationshipLink_ cardinality

```
USE [EERMM]
GO
/****** Object:  StoredProcedure [dbo].[sp_updateRelationshipLink_cardinality]    Script Date: 2015/6/9
2:24:43 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                Liwei and Peter
-- Create date: 16/July/2014
-- Description:    update
-- =============================================
ALTER PROCEDURE [dbo].[sp_updateRelationshipLink_cardinality]
      @entityName SYSNAME,
      @relationshipName SYSNAME,
                  @relationshipLinkName SYSNAME,
      @role_sglr SYSNAME,
      @role_plrl SYSNAME,
                  @cardinality CHAR(7)
AS
BEGIN
          SET NOCOUNT ON;
```

```
DECLARE @oldCardinality AS CHAR(7);
DECLARE @count AS INTEGER;
DECLARE @relationship_ID AS INTEGER;

-- Get the relationship_ID for the link's relationship
SELECT @relationship_ID = relationship_ID
FROM tblRelationship
WHERE relationshipName = @relationshipName;

-- Check that tblRelationshipLink contains a record
-- with these values.

SELECT @count = COUNT(*)
FROM tblRelationshipLink
WHERE relationshipLinkName = @relationshipLinkName
AND relationship_ID = @relationship_ID
AND entityName = @entityName
AND role_sglr = @role_sglr
AND role_plrl = @role_plrl;

IF @count = 0
BEGIN
   PRINT 'ERROR: record not found in tblRelationshipLink'
END
ELSE
BEGIN
   -- Check that the current value of the cardinality attribute
   -- for this record is 'Unknown'.
   SELECT @oldCardinality = cardinality
   FROM tblRelationshipLink
   WHERE relationshipLinkName = @relationshipLinkName
   AND entityName = @entityName
         AND role_sglr = @role_sglr
   AND role_plrl = @role_plrl;

         IF @oldCardinality <> 'Unknown'
         BEGIN
            PRINT 'ERROR: the value of meta-attribute cardinality is currently not Unknown'
```

```
                END
                ELSE
                BEGIN
                  UPDATE tblRelationshipLink
                        SET cardinality = @cardinality
                        WHERE relationshipLinkName = @relationshipLinkName
            AND entityName = @entityName
                        AND role_sglr = @role_sglr
            AND role_plrl = @role_plrl;
                END
        END
END
```

sp_updateRelationshipLink_participation

```
USE [EERMM]
GO
/****** Object: StoredProcedure [dbo].[sp_updateRelationshipLink_participation] Script Date:
2015/6/9 8:03:44 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[sp_updateRelationshipLink_participation]
    -- Add the parameters for the stored procedure here
    @entityName SYSNAME,
    @relationshipName SYSNAME,
    @relationshipLinkName SYSNAME,
    @role_sglr SYSNAME,
    @role_plrl SYSNAME,
    @participation CHAR(7)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    DECLARE @oldParticipation AS CHAR(7);
    DECLARE @count AS INTEGER;
    DECLARE @relationship_ID AS INTEGER;
```

```
-- Get the relationship_ID for the link's relationship
SELECT @relationship_ID = relationship_ID
FROM tblRelationship
WHERE relationshipName = @relationshipName;

-- Check that tblRelationshipLink contains a record
-- with these values.

SELECT @count = COUNT(*)
FROM tblRelationshipLink
WHERE relationshipLinkName = @relationshipLinkName
AND relationship_ID = @relationship_ID
AND entityName = @entityName
AND role_sglr = @role_sglr
AND role_plrl = @role_plrl;

IF @count = 0
BEGIN
   PRINT 'ERROR: record not found in tblRelationshipLink'
END
ELSE
BEGIN
   -- Check that the current value of the participation attribute
   -- for this record is 'Unknown'.
   SELECT @oldParticipation = participation
   FROM tblRelationshipLink
   WHERE relationshipLinkName = @relationshipLinkName
   AND entityName = @entityName
      AND role_sglr = @role_sglr
   AND role_plrl = @role_plrl;

      IF @oldParticipation <> 'Unknown'
      BEGIN
         PRINT 'ERROR: the value of meta-attribute participation is currently not Unknown'
      END
      ELSE
      BEGIN
```

```
            UPDATE tblRelationshipLink
                SET participation = @participation
                WHERE relationshipLinkName = @relationshipLinkName
            AND entityName = @entityName
                AND role_sglr = @role_sglr
            AND role_plrl = @role_plrl;
            END
        END
END
```

(f) EmpsAndDepts create script

```
USE [master]
GO
/****** Object:  Database [EmpsAndDepts]    Script Date: 17/06/2014 15:32:54 ******/
IF EXISTS(SELECT * from sys.databases WHERE name='EmpsAndDepts')
BEGIN
   DROP DATABASE EmpsAndDepts;
END

CREATE DATABASE [EmpsAndDepts]
 CONTAINMENT = NONE
 ON PRIMARY
( NAME = N'EmpsAndDepts', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.SQLEXPRESS\MSSQL\DATA\EmpsAndDepts.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED, FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'EmpsAndDepts_log', FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL11.SQLEXPRESS\MSSQL\DATA\EmpsAndDepts_log.ldf' , SIZE = 1024KB , MAXSIZE = 2048GB , FILEGROWTH = 10%)
GO
ALTER DATABASE [EmpsAndDepts] SET COMPATIBILITY_LEVEL = 110
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [EmpsAndDepts].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
```

```
ALTER DATABASE [EmpsAndDepts] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [EmpsAndDepts] SET ANSI_NULLS OFF
GO
ALTER DATABASE [EmpsAndDepts] SET ANSI_PADDING OFF
GO
ALTER DATABASE [EmpsAndDepts] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [EmpsAndDepts] SET ARITHABORT OFF
GO
ALTER DATABASE [EmpsAndDepts] SET AUTO_CLOSE OFF
GO
ALTER DATABASE [EmpsAndDepts] SET AUTO_CREATE_STATISTICS ON
GO
ALTER DATABASE [EmpsAndDepts] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [EmpsAndDepts] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [EmpsAndDepts] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [EmpsAndDepts] SET CURSOR_DEFAULT GLOBAL
GO
ALTER DATABASE [EmpsAndDepts] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [EmpsAndDepts] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [EmpsAndDepts] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [EmpsAndDepts] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [EmpsAndDepts] SET DISABLE_BROKER
GO
ALTER DATABASE [EmpsAndDepts] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [EmpsAndDepts] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [EmpsAndDepts] SET TRUSTWORTHY OFF
GO
```

```
ALTER DATABASE [EmpsAndDepts] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [EmpsAndDepts] SET PARAMETERIZATION SIMPLE
GO
ALTER DATABASE [EmpsAndDepts] SET READ_COMMITTED_SNAPSHOT OFF
GO
ALTER DATABASE [EmpsAndDepts] SET HONOR_BROKER_PRIORITY OFF
GO
ALTER DATABASE [EmpsAndDepts] SET RECOVERY FULL
GO
ALTER DATABASE [EmpsAndDepts] SET MULTI_USER
GO
ALTER DATABASE [EmpsAndDepts] SET PAGE_VERIFY CHECKSUM
GO
ALTER DATABASE [EmpsAndDepts] SET DB_CHAINING OFF
GO
ALTER DATABASE [EmpsAndDepts] SET FILESTREAM( NON_TRANSACTED_ACCESS = OFF )
GO
ALTER DATABASE [EmpsAndDepts] SET TARGET_RECOVERY_TIME = 0 SECONDS
GO
EXEC sys.sp_db_vardecimal_storage_format N'EmpsAndDepts', N'ON'
GO
USE [EmpsAndDepts]
GO
/****** Object:  Table [dbo].[Department]    Script Date: 17/06/2014 15:32:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Department](
	[keyDepartment] [int] NOT NULL,
	[DeptName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Dept] PRIMARY KEY CLUSTERED
(
	[keyDepartment] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Employee]    Script Date: 17/06/2014 15:32:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
INSERT dbo.Department
   VALUES  (101,'Sales'),
              (102,'Production'),
                    (103,'Marketing')
GO
CREATE TABLE [dbo].[Employee](
          [keyEmployee] [int] NOT NULL,
          [EmpName] [varchar](max) NOT NULL,
 CONSTRAINT [PK_Emp] PRIMARY KEY CLUSTERED
(
          [keyEmployee] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Employment_Bridge]    Script Date: 17/06/2014 15:32:54 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
INSERT dbo.Employee
```

```
   VALUES  (11,'John Smith'),
                (12,'Jane Brown'),
                        (13,'Ann Jones'),
                        (14,'Robert Bruce')
GO
CREATE TABLE [dbo].[Employment_Bridge](
          [keyEmployee] [int] NOT NULL,
          [keyDepartment] [int] NOT NULL
) ON [PRIMARY]

GO
INSERT dbo.Employment_Bridge
   VALUES  (11,101),
                (12,102),
                        (13,103),
                        (14,103)
GO
USE EERMM
DECLARE @RC int

EXECUTE @RC= [dbo].[sp_createSchema] 'EmpsAndDepts'

USE [master]
GO
ALTER DATABASE [EmpsAndDepts] SET READ_WRITE
GO
SELECT * FROM dbo.Department
SELECT * FROM dbo.Employee
SELECT * FROM dbo.Employment_Bridge
```