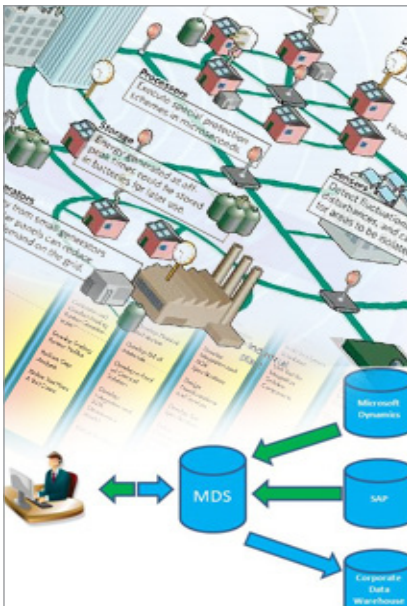


B E Y O N D

Think Beyond. Think Mphasis.

WHITE PAPER



Some Novel Ideas Around Business Entity Modeling

Bert Hooyman
Chief Architect, Mphasis Europe

February, 2011

Contents

Introduction	2
1.Business Entity Modeling	3
2.Business Entity Associations	9
3.Access Control: Role Modeling	14
4.Database: Schema Modeling	18
5.Concluding Remarks	22
6.References	26

List of Figures

Figure 1 Entity Subset Modeling	5
Figure 2 Entity Inheritance Modeling	6
Figure 3 Entity Services Modeling	7
Figure 4 Property Modeling	7
Figure 5 Projection Modeling	8
Figure 6 Documenting entity and property models	9
Figure 7 Association Modeling	10
Figure 8 Association Modeling	12
Figure 9 Association Subsets	13
Figure 10 Association Documentation	13
Figure 11 Role Modeling	14
Figure 12 Data-Driven Access Control using Subsets ..	15
Figure 13 Extended Entity Modeling	15
Figure 14 Property Access Model	16
Figure 15 Association Access Model	16
Figure 16 Service Access Model	16
Figure 17 Role-specific Projection Modeling	17
Figure 18 Access Control Documentation	18
Figure 19 Modeling tables and columns	20
Figure 20 Modeling Relationships	20
Figure 21 Modeling Subtypes and Supertypes	21
Figure 22 Modeling Change Tracking	21
Figure 23 Full Business Entity Meta-model	23
Figure 24 Full Access Control Meta-model	24
Figure 25 Full Schema Meta-model	24

List of Tables

Table 1 Entity lifecycle	4
Table 2 Aspects of an entity subset	5
Table 3 Property lifecycle modes	7
Table 4 Association lifecycle support	12
Table 5 Entity Access Control Model	14
Table 6 Property-level Access Control Modeling	15
Table 7 Association-level Access Control Modeling ..	16
Table 8 Modeling table purpose	19
Table 9 Modeling column purpose	19
Table 10 Modeling subtypes	21
Table 11 Change Tracking	21
Table 12 Documentation at the Database Schema Level .	22

Introduction

A lot has been written already on the subject of data modeling, object modeling, and business entity modeling. Why another paper?

In this report, business entity modeling will be considered with a somewhat broad perspective; this approach considers business entity modeling

- for the purpose of defining an object/relational mapping,
- for the purpose of defining a data management user interface,
- and finally for the purpose of defining an access control model against the business information being modeled.

The focus is on the business stakeholder's role as a provider of modeling input. There is little emphasis on the design of a physical data schema, although the ideas presented here extend naturally to that area as well.

It is not intended to redefine *everything* related to data modeling. Instead, this report addresses a number of less conventional aspects of business entity modeling. The main focus areas addressed in this report are:

- Modeling business entities and their properties;
- Modeling associations (relationships) between entities;
- Modeling (role-based as well as value-based) access control to business data;
- Logical data schema modeling.

In this paper, we address only the initial stages of data modeling, i.e. the stage where there is direct interaction with the business stakeholders. For this reason, we cover the logical data schema modeling only lightly, and we ignore the physical data schema design completely.

A Note on Model Diagrams

In this report, we do not propose a new way to create model diagrams – not because it would be impossible, but because we believe that there are so many diagramming techniques already, one would be hard pressed to believe yet another notation would make any inroads.

For now, we consider the modeling approach presented here as a textual modeling notation – we have an XSD schema that covers all elements of the modeling approach discussed in this report. A suitable XML editor, such as XMLSpy, can be used with this schema to help the creation of valid data models. As a next step, we plan to develop an interactive dialogue tool to assist business analysts in their modeling efforts. This tool would be used during business stakeholder interviews and provide valid XML output upon completion.

Intended Audience

The subject of this report is business entity modeling. Both business users and business analysts are stakeholders in a modeling exercise. However, the subject of this report is the modeling language itself, so the business analysts are the end

users and we're analyzing the needs of the analyst. The intended audience for this report is the architecture and design community, where tooling must be provided to let business analysts do their work. Ideally, the resulting business domain models can be used for a technical design as well (of the physical data schema as well as the data access code). Again this should be of interest to the architecture and design community.

Structure of this Paper

Chapters 1 through 4 discuss the details of some novel modeling concepts. In chapter 1, we address the business entity model and its properties. We discuss entity associations (relationships) in chapter 2. Chapter 3 focuses on role modeling and role-based access control. Chapter 4 is a light coverage of logical database schema modeling; it introduces some auxiliary modeling concepts in that space. Concluding remarks, including future work, are presented in chapter 5; references are provided in chapter 6.

In chapter 1 through 4, every concept is discussed in a separate section, e.g. section 1.1 is about "Indicative data volumes". Every such section provides a description of the issue at hand followed by a modeling proposal. All modeling proposals are captured in metadata diagrams throughout the text. In total, 43 modeling concepts are discussed.

1. Business Entity Modeling

Chapter Contents

1.1	Indicative data volumes	3
1.2	Common entity name and its plural form	3
1.3	Natural object ordering	3
1.4	Entity lifecycle	3
1.5	Business entity subsets	4
1.6	Entity inheritance	5
1.7	Version history & change management requirements	6
1.8	Entity services	6
1.9	Property names and textual on-screen reference	7
1.10	Property validation	7
1.11	Property lifecycle	7
1.12	Entity projections	7
1.13	Document all entities and entity properties	8
1.14	Summary of entity modeling	9

We begin our modeling exercise with business entities; these are the fundamental business objects that are the foundation of the business domain model¹. We focus on data modeling, leaving the behavioral aspects of the business domain mostly out of scope.

1.1 Indicative data volumes

In traditional business entity modeling, expected data volumes are rarely considered. However, such information is relevant for the data management user interface, specifically. In addition, expected data volumes are useful in the final stages of physical database schema design. There is no need to understand the exact data volumes, but what is good to know is the distinction between:

- Entities of which there will be many instances, so that list pagination, scrolling and search & find are necessary elements of the user interface
- Entities of which there will be only a few instances, ever. Examples are value lists that carry only a few instances so that all of them fit on a single screen when displayed in list format
- Entities of which only one single instance will ever exist, such as system configuration objects.

During physical database schema design, not only the data volume is important, but also the expected volume growth rate and the data volatility (i.e. the rate of change). This applies to the first of the three options above, only, of course. Volumes and volume growth rates drive database storage sizing and capacity calculation.

Modeling Proposal

The modeling approach suggested to indicate data volumes is to introduce a metadata attribute named *extent* with possible values of *numerous*, *handful* and *singleton*.

1.2 Common entity name and its plural form

This is a simple enhancement of the normal modeling process – let the business determine not only the common business entity name, but also the plural form which can either be a linguistic plural or an alternative term used for a collection of entities.

The plural form of Account is Accounts, but the plural form of Log Entry is Audit Log. In logistics, the plural form of Shipment can very well be Manifest. The benefit of capturing plural names is that plural names are very useful for system-generated user interfaces and documentation.

1.3 Natural object ordering

In a relational database, there is no implicit ordering of data as it is returned by a query. For a consistent user experience, it is useful to define a default ordering on all business entity types. Such a default ordering need not be on a unique property, in which case it is useful to allow ordering on multiple properties. Moreover, it is desirable to allow ordering on non-resident properties, i.e. on properties of associated objects.

Modeling Proposal

The suggested modeling approach is to introduce a modeling attribute named “ordering” which is a comma-separated sequence of object properties (a.k.a. a *projection*, as described later in this report). The first property mentioned will be the major sort property, and in case of identical values on the major property, subsequent properties will be used for sub-sorting the query results. Every property in itself is either a simple entity property, or the property of an associated entity in which case the object property name must include the association path to the associated entity, as explained under “entity projections” on page 7.

1.4 Entity lifecycle

We define the entity lifecycle as *the allowed set of operations* on any instance of that entity type. Understanding the desired entity lifecycle is not very important for a relational data model, however it is relevant in the O/RM space and it is crucial for the automated generation of a data management user interface. We believe that a proper capture of the entity lifecycle helps the modeling exercise in that it exposes an otherwise implicit behavior of the business entities.

An entity lifecycle provides specific information about:

- the ability to create new entity instances;
- the ability to modify entity instances once they have been created;
- the ability to delete entity instances;
- the ability to copy entity instances rather than having to construct them afresh;
- the ability to expose entity instances for viewing purposes only.

¹ We use the terms business object and business entity for the same concept. A domain model brings together a set of related business entities.

Modeling Proposal

Clearly, the entity lifecycle is closely related to the well-known CRUD operations. With four CRUD operations, the number of different combinations is sixteen. If we consider “copy” as a fifth operation, then we have 32 combinations already. Luckily, most of these combinations are useless or impossible. The combinations listed in Table 1 are actually useful and sufficient.

In summary, the default lifecycle mode should be *maintained*, with or without the copy and expert options. The *disposable* lifecycle is recommended for simple entity types that have only a few properties. The *permanent* lifecycle is intended for business transaction data which is normally never updated once created (only compensated in case of errors). The *visible* lifecycle is intended for read-only entity types. *Evolving* lifecycles apply to data sets that cannot grow or shrink but must still be modifiable².

Table 1 Entity lifecycle

Lifecycle	Create	Retrieve	Update	Delete	Copy
Full	y	y	y	y	n
<i>When the lifecycle support is “full”, then instances of this entity type can be created, modified, and deleted.</i>					
Maintained	y	y	y	n	n
<i>In “maintained” mode, entity instances can never be deleted; at best they can have a “soft delete” applied to them making them invisible. This is probably the most appropriate lifecycle form for most business entity types, and it should be considered the default.</i>					
Evolving	n	y	y	n	n
<i>In “evolving” mode, the collection of entity instances remains fixed in that no new items can be added, nor can any item be deleted. But the details of an instance can be updated.</i>					
Permanent	y	y	n	n	n
<i>In “permanent” mode, entity data cannot be modified once created. It applies to all properties of all entity instances and results in a rather constrained data type, especially since no instances can be removed. Consider a permanent lifecycle for capturing business transactions which, by definition, should never be modified once captured (instead, business transactions are compensated in case of errors).</i>					
Disposable	y	y	n	y	n
<i>The “disposable” lifecycle is the natural improvement of the “permanent” lifecycle, as it allows for removal of unwanted entity instances.</i>					
Visible	n	y	n	n	n
<i>“Visible” entity types are truly read-only. This is useful in scenarios where business entity data is replicated into a business domain from somewhere else, or in case multiple applications are sitting on the same data source. The original source would normally be updateable, but the copy that is in scope for the business domain under consideration cannot be modified in any way. A visible domain would be exposed in a user interface through the normal list/details/search screens, without an edit form of course.</i>					
Invisible	n	n	n	n	n
<i>This appears to be a rather useless lifecycle mode, however it is still relevant as it allows for referring to a business entity without exposing its full details. Such business entities exist at the boundaries of a business domain; they are managed and owned by some other business domain. An invisible domain would not be exposed in the user interface independently – there would not be a list screen or a details screen or a search form, but references to the entity could appear in selection lists (as references) and/or in membership operators.</i>					
Maintained (or full) with copy	y	y	y	(y)	y
<i>This mode extends the “maintained” and the “full” mode with a copy operator; in the user interface this results in an additional command and some extra business logic. There is no impact on the object/relational modeling.</i>					
Maintained (or full) by expert	y	y	y	(y)	(y)
<i>This mode reduces the user interface of the “maintained” and the “full” mode. In expert mode, there is no longer a distinction between a “view” screen and an “edit” form. Instead, when a user navigates from a list screen to a details screen, the details are displayed in an editable form immediately (or using Web 2.0, the data is displayed in viewing mode initially but morphed into an editable form without leaving the screen). Expert mode and copy mode are not mutually exclusive – they can be combined.</i>					

1.5 Business entity subsets

In business requirements, it is often the case that for a given business entity, there are well understood subsets of instances that play a relevant role in the requirements. Often, this situation is typified by the presence of a state or status property, and the Subsets focus on a single value for the status.

An example would be your Outlook mailbox, which distinguishes unread mail, read mail and deleted mail. All of these mail messages are essentially identical, but they are treated differently based on the status of the message.

Another example is the notion of “My Business”, for example “my orders”, “my pending reimbursements” etc. These are examples of subsets that filter dynamically on an end user’s identity.

² One example in the domain of role-based access control would be the collection of entitlements. A role is defined as a subset of the known entitlements, and new roles can be created at will. However, the introduction of new entitlements is pointless unless the underlying access control code is also modified. As this normally requires a software release cycle, there is no point in provisioning a “maintained” lifecycle for entitlements. They can still be “evolving rather than visible” to allow for live updates of entitlement descriptions, help texts etc.

The real value of capturing business entity subsets is when it comes to access control, as discussed in more detail in chapter 3 of this report. With entity subsets, it is feasible to model *value-based access control*, i.e. the access to an instance of a given entity type is (in part) controlled by some of the instance's property values. As an example, consider a scenario where a sales person can work on his own sales orders, but is allowed to only see the sales orders of colleagues in his geography, with no access at all to sales orders in other regions. This would require two subsets: "My Sales Orders" defined as all sales orders where the given user is identified as the current sales representative, and "Regional Sales Orders" defined as all sales orders matching the geographic region of the current sales representative. Using

role-based access control, it is now possible to define *maintained* access to "My Sales Orders" and *visible* access to "Regional Sales Orders", complemented by *invisible* for "Sales Orders".

A Note on Using Entity Subsets

Entity subsets are useful predominantly for data retrieval; it is to a large extent a static selection of a subset of entity instances based on some criteria. During insert, the expected behavior would be that the relevant entity properties are initialized with the appropriate values. However, when this is not the case, or when a matching instance is updated in a way that it no longer meets the subset criteria, then upon completion of a "save", the data would no longer be present in the entity subset, but only through the entity itself.

Modeling Proposal

We refer to subsets through the notion of an *entity subset*. It is highly useful to identify such entity subsets during the business entity modeling exercise; to understand the selection criteria and to identify how the subset differs from its source. Hence, the following aspects of an entity subset should be modeled:

Table 2 Aspects of an entity subset

Aspect	Rationale
qualifier	The selection criteria that distinguish the subset from the source collection
name	The name of the subset (e.g. "Unread Mail Messages")
plural name	The plural name for the subset
extent	The extent of the subset. This can be less than the extent of the source.
ordering	The ordering of the entity subset
default access rights	Access control rules for a subset can be different from the source (see the chapter on role modeling later in this document)

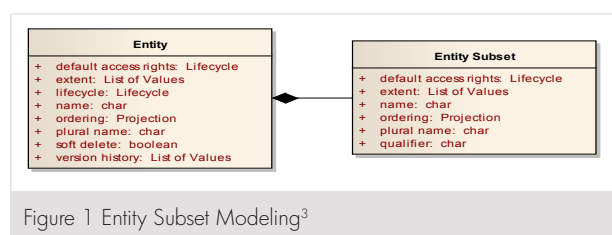


Figure 1 Entity Subset Modeling³

Note that an entity subset is not equivalent to an entity subtype, i.e. it is not implemented through an inheritance scheme. The source and the subset entities do not differ in any way; it is only that some data values are specific to the subset.

1.6 Entity inheritance

Modeling inheritance is common practice in object-oriented modeling and in entity relationship modeling as well, ever since it was introduced in Enhanced Entity Relationship modeling (EER).

What is important is to ensure that the inheritance is modeled in sufficient detail so that code generation and database schema generation strategies are optimally informed.

Modeling Proposal

There are two relevant aspects to inheritance modeling. We refer to these as the *coverage* and the *subtyping* aspects.

1.6.1 Inheritance Coverage

Coverage is concerned with whether every instance of the base class is always a member of at least one subclass or not. When coverage is *incomplete*, some base class instances are not of any subclass hence a full user interface for managing the base class is required. Conversely, when the coverage is *complete* then there are no instances of the base class entity that are not also an instance of a subclass. Base class instances may (but need not) be managed independently. Ultimately, the coverage can be *abstract*, which is to say that the implementation uses an abstract base class which is by definition never exposed in the user interface.

1.6.2 Inheritance Subtyping

Subtyping is concerned with the instances of the subclasses. When any base class instance is an instance of *at most one* subclass, subtyping is said to be *exclusive*. Conversely, when a base class instance can be an instance of *more than one* subtype at the same time, then subtyping is said to be *inclusive*.

Inclusive subtyping occurs rather frequently in the business; the classic example is the famous Person base class in a university setting. A person can be a student or a professor or both. Unfortunately, implementing inclusive subtyping is not straightforward, and building an intuitive user interface is not very straightforward, either. For a professor, the data entry screen should expose the professor-specific attributes, whereas for a student, it should expose the student-specific attributes. Would we need two distinct data entry screens for people who act in both roles? Or would the data entry screen combine the attributes of both roles? How about inclusive subtyping over more than two subtypes?

³The diagrams in this report do not represent a proposed modeling notation. Instead, these diagrams capture the metamodel, i.e. they document the modeling, not the resulting models. Concepts covered in this reported are marked with a plus sign, whereas concepts that are not discussed in the text are marked with a minus sign.

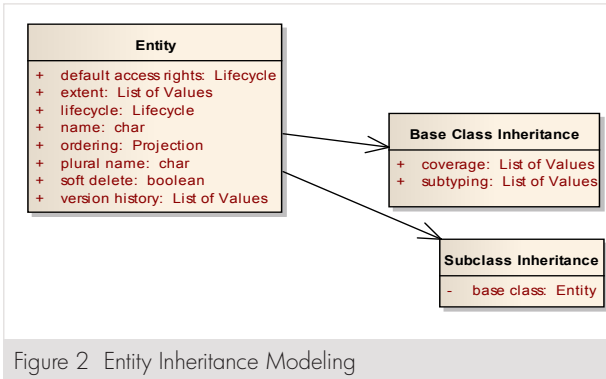


Figure 2 Entity Inheritance Modeling

1.6.3 Inheritance versus Subsets

We discussed entity subsets in an earlier section of this report. Is a subset any different from inheritance? We think it is. A subset of an entity has exactly the same properties as the source entity, the only difference is in its qualification as being a member of the subset or not – some entity properties meet specific criteria. Change the property and the instance may or may not be a member of the subset any longer.

Using inheritance, we introduce the option of having different entity properties for selective subtypes, and augmented behavior as well. This is not possible with subsets. It is possible to use inheritance as an alternative for subsets only when static selection criteria are used, e.g. status fields or date fields that determine whether an instance is part of the subset or not. Dynamic subsets (“my open requests”) are not possible. At any rate, using subsets would be preferred as the implementation of subsets is much simpler than inheritance.

1.7 Version history & change management requirements

Considering the maintenance of a version history usually comes as an afterthought to entity modeling. It is often thought (incorrectly) to be solved through an audit log, which, in the worst case scenario, is a text file representation of data modifications that happened in the past⁴.

Such a version history is not meant for consumption by end users – it is a system administration support utility at best. A proper version history is a record of how things were set up in the past. Consider a person entity and an associated address entity. The association tells us the current address of the person. What if there is a business requirement to track past addresses of the person? That is a version history of the person-to-address association, quite likely with start and end dates.

Modeling Proposal

At the entity level, change tracking can be one of:

- *none*, which is to say that only the current values are maintained (default),
- *auditLog*, which is to say that all changes to entity data are captured in an audit log (this is appropriate for all business transaction data because transactions are typically never

- amended, only compensated). Consider an audit log for all entities that have a lifecycle mode of “permanent” or “disposable” (the concept of a lifecycle is addressed in section 1.4 on page 3 and 4 of this document)
- *versionHistory*, which is to say that the full history of property changes to an entity is held. With every update to an entity instance, a record is added to its version history.
- *unitOfWork* which is an elaboration of versionHistory. Here, the scope is not just one single entity type but rather a collection of entities (an entity hierarchy) that is modified in one single unit of work. The unit of work groups all the changes to the data (typically through a common identifier that is added to all version histories and or audit log entries).

As an alternative to the above proposal, one might consider tracking version histories at the property level, rather than at the entity level. This is certainly possible, albeit complicated and potentially confusing to the software development community. It is a possible solution in cases where the business entity models are very rich, i.e. they contain a large number of simple properties. Rather than storing copies of all those (hundreds or thousands of) properties, one might consider storing only versions of the ones that changed.

Orthogonal to version histories, but still relevant in the context of tracking changes over time, is the concept of *soft delete*, which effectively means “whenever an object is to be deleted, mark it as such but do not remove the information from physical storage”. This is also known as a *logical delete*. It is a separate entity attribute that must be captured as such.

1.8 Entity services

Where *object* modeling is concerned with information modeling as well as behavioral modeling, classic *data* modeling is not in any way concerned about entity behavior.

We recommend to include service modeling in the business entity modeling exercise. Services are user-visible behavioral aspects of business entities. Think of them as user tasks that are applicable to entities. Common examples are:

- print a summary list
- export details of an entity
- upload instance data

Within any business domain, there will be many services specific to the entity type being modeled. In the email domain, consider:

- mark as read/unread
- move to folder
- recall
- resend
- view source
- add sender to blocked senders list
- ...

⁴ See the earlier white paper “Understanding Application Audit Requirements”, available at http://www.mphasis.com/knowledgeBank/kbank_whitepapers_all.html#2008.

Modeling Proposal

For every identified service, consider modeling the service *name*; the screen *label* to be used on screen, and the *scope* of the service, i.e. whether it applies to a single instance, to all instances in a summary list or to a selected subset of instances.

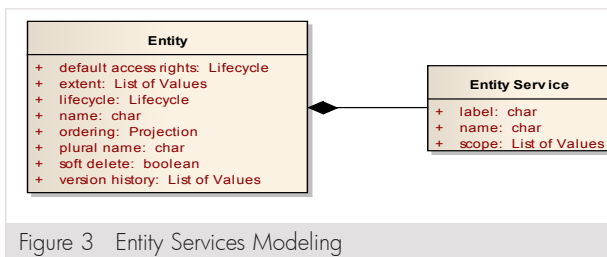


Figure 3 Entity Services Modeling

1.9 Property names and textual on-screen reference

Having discussed entity modeling so far, we now turn to the modeling of entity properties or attributes.

Entity properties should have clear and well-chosen names.

Typically, property names appear in the user interface in either a long or a short form, e.g. as field labels and column headers. There may be cases where field labels should be different from the property name, for example when the user audience is not comfortable with the names chosen by the subject matter experts.

Modeling Proposal

For this reason, a business entity model should allow for the specification of a *long label* and a *short label* in addition to the property name.

1.10 Property validation

For text-type properties, free-form data entry often leads to invalid or incomplete data. Newly entered data should be checked immediately upon entry. To support validation, the property model may be extended with a field mask, an allowed values pattern or a simple cross-attribute validation (e.g. `dateOfJoining ≥ dateOfBirth + 16 years`).

Modeling Proposal

No specific modeling is proposed here, as the full breadth and depth of necessary validation is highly dependent on the business problem at hand. At a future stage, we might be able to identify some best practices and patterns around this. We are currently looking into a simple *constraint language* as an extension to the modeling concepts introduced here.

1.11 Property lifecycle

Entity lifecycles were introduced earlier in this report; there is a similar concept of property lifecycles that is in fact much simpler. Valid lifecycle modes for properties are a subset of the entity lifecycle modes:

Table 3 Property lifecycle modes

Entity Lifecycle	Property Lifecycle	Description
Full	not supported	Not supported as the concept of "delete" does not apply to a property.
Maintained	✓ supported	Values can be set, modified and cleared as needed
Evolving	not supported	Not supported as there is no practical difference with "maintained".
Permanent	✓ supported	Values can be set at most once – once set the value cannot be modified or cleared (typically for natural keys)
Disposable	not supported	Not supported as the concept of "delete" does not apply to a property.
Visible	✓ supported	Values can be viewed but not modified in any way (typically for calculated properties)
Invisible	not supported	Not supported as invisible properties might as well not be modeled at all.

Modeling Proposal

With the above property lifecycle modes of *maintained*, *permanent* and *visible*, the lifecycle is almost completely defined. The one attribute that is not covered is whether a property value can be null yes or no, which is of interest in the context of *maintained* and *permanent*.

A permanent property that is *nullable* can initially be null, but once it is set to a specific value it cannot be modified. A maintained nullable property, on the other hand, can be cleared to null at any point in time. A permanent non-null property is useful in many scenarios, in particular when using natural keys. Use a *visible* lifecycle for properties that are calculated by the underlying database.

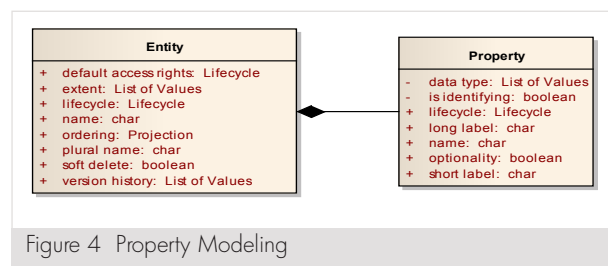


Figure 4 Property Modeling

1.12 Entity projections

In requirements engineering, one of the important steps in the process is to capture the relevant data elements that participate in a use case; in many cases this translates into the fields that are exposed on a data viewing or data entry screen. Every screen or window, and indeed every report, displays a well-specified set of entity properties. This is what we propose as the *projection*.

We define an entity projection as a set of entity properties considered for inclusion to accomplish a certain user task.

When the user task is about selecting an entity instance from a list of instance summary data, then the projection is the specification of the properties that appear in the summary (i.e. the columns in a tabular display). For a search & find task, the projection specifies the entity properties considered for inclusion on the search form.

The whole concept of entity projections is irrelevant for the entity modeling exercise where it relates to the database design or the object-relational modeling. It is relevant only for the client-side interaction with a business entity model. As such it is an important part of any business requirements analysis.

The benefit of introducing projections is that they encapsulate the most volatile part of any system requirements specification: the data elements being considered. With projections, these changes are now isolated in one place. It is possible (in fact desirable) to consider client API designs that use projections as a means to encapsulate change⁵.

A projection is a set of properties; these are either simple entity properties or they are properties of associated entities. These in turn can be singular or collections (these terms are introduced further down, when we discuss entity association modeling). To specify a projection, we recommend using a textual representation. A projection is thus a comma separated list of property names. Each property name is either a simple entity property name or it refers to a property of an associated entity (a.k.a. a non-resident property). To expose the product description for every line item on a sales order, the projection could for example include "Order.lineItem.product.description", where "lineItem" is the named association from order to order line item, and "product" is the named association from line item to product. Entity associations are discussed in chapter 2 of this document.

Modeling Proposal

Every property that is included in a projection already has a property name, and in addition a long label and a short label may be defined for every property as well. In the context of a projection, it may be desirable to associate yet another label with a property, which is the label being applied wherever the projection is used. This is especially useful when including associated properties that have a natural name in the context of their owning entity, but when used via an association that name may no longer be a natural choice.

With every entity type, there will be multiple projections, typically one each for every type of task (list, details, edit, search...). For each property that is included in a projection, there may be a *projected label* that overrides the long and short labels given for the property itself. Furthermore, there may be restrictions on the *access* to a property in a given projection, for example not all properties may be editable even though they appear on an edit form.

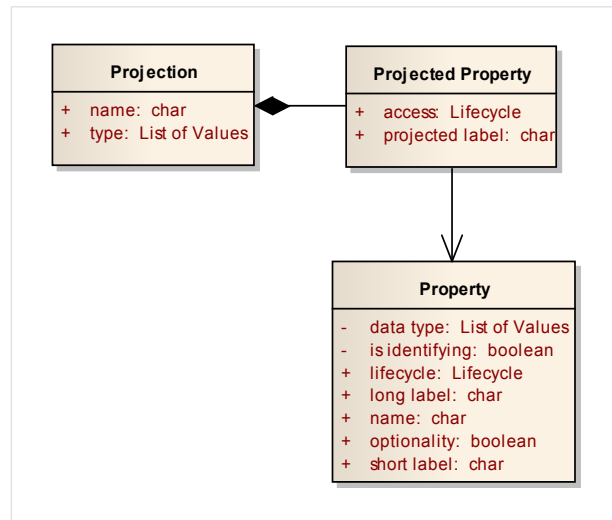


Figure 5 Projection Modeling

1.13 Document all entities and entity properties

The notion of documentation in itself may not be novel, at least not to most of us. However, current modeling techniques that are based on a diagramming notation often fail to include documentation integral to the model. We suggest ensuring that a business entity model is always fully documented.

In particular, we suggest to document:

- entity and property definitions; these form an extremely relevant *glossary of business terms*, when done properly.
- design rationale (why is this modeled in this way, what alternatives were considered and why were they rejected)
- business requirements satisfied by introducing an entity or a property. Ideally, such requirements are classified as being data, policy or process requirements – these have varying levels of stability
- mapping of entities and properties to database tables and columns, especially when these mappings are not obvious

Beyond the above documentation elements, consider another level of documentation that goes across all business entities. This top-level documentation addresses the business tasks that the final system is supposed to support. At this level, documentation is in the form of *storyboards* (architecturally significant use cases).

⁵ See the earlier white paper "Repositories for Data Management – A Data Access Pattern Language", available at http://www.mphasis.com/knowledgeBank/kbank_whitepapers_all.html#2005.

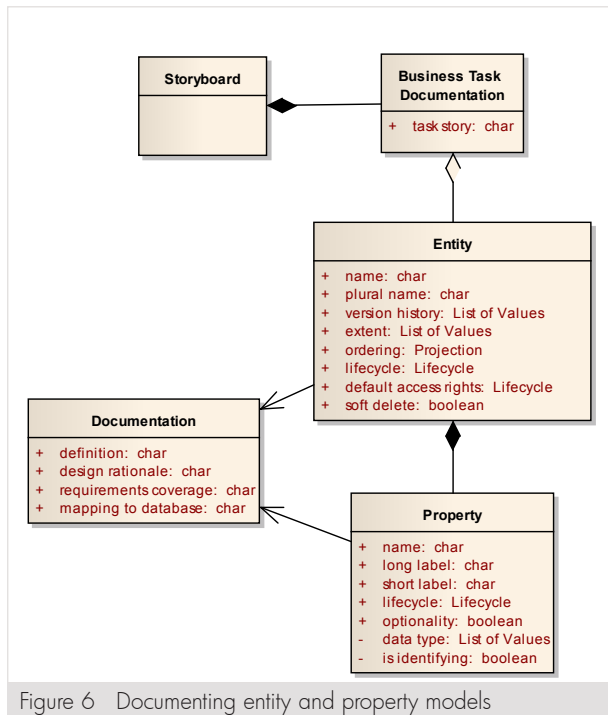


Figure 6 Documenting entity and property models

1.14 Summary of entity modeling

In this chapter, we have introduced basic entity modeling, leaving entity associations (relations) to be covered in the next chapter. We have focused on the entity data model, rather than the behavior of business objects.

We chose to ignore the more common aspects of data modeling and look at alternative or incremental modeling concepts to improve business entity modeling.

Entity Naming and Ordering

All entity types must have a well-defined name that the business can agree upon. We recommend defining a plural name as well, or more precisely a name that describes a collection of instances (*staff* describes the collection of all employees but is not the plural form of employee).

Entity collections should have a natural ordering, and this ordering should be modeled explicitly.

Entity Data Volumes

Indicative data volume modeling helps during requirements analysis in understanding the need for elaborate data management user interfaces with search, list pagination etc.

Entity Lifecycle

The entity lifecycle models the allowed set of operations, irrespective of a subsequent access control model. With the lifecycle being part of an entity model, it is feasible to consider automated generation of appropriate user interfaces that support the intended lifecycle.

Entity Subset

Entity subsets are introduced to provide support for value-based access control. On their own, entity subsets can be used to provide an entry point to controlled subsets of entity instances. This is also possible using association subsets.

Entity Inheritance

We propose to extend the inheritance model for base classes and subclasses with coverage and subtype exclusivity. Incompletely covered base classes require a data management user interface for the base class, whereas complete and abstract base classes do not require this.

Inclusive subtypes are more complex than exclusive subtypes. In the latter case, a data instance is of at most one subtype, so its data management user interface is straightforward and data access is simple. In the case of inclusive subtypes, a data instance can be of more than one subtype at the same time. Data can now be spread over multiple tables, and the GUI must support data entry across all subtypes in parallel.

Entity History

We propose to model the business requirements towards the history of changes to entity data as an integral part of the entity model. The requirement is modeled as a single attribute that can take one of four distinct values, modeling what the requirements are, not how the requirements are to be realized.

Entity Properties, Projections and Services

All entities have properties and associations with other entities. We address entity associations in a subsequent chapter. Entity properties are relatively simple; for these we propose to model on-screen data labels (long and short) and allowed operations (as a lifecycle).

We introduce projections as collections of properties, with projection-specific on-screen label and access rights. We introduce service modeling as an extension point for the basic CRUD services of a data management user interface.

2. Business Entity Associations

Chapter Contents

2.1 Entity Association:Undirectional	10
2.2 Entity Association: Verb or Noun	10
2.3 Association Integrity	10
2.4 Association Kind	10
2.5 Association Conductivity	11
2.6 Association Cardinality : Handful or Numerous	11
2.7 Natural Ordering of associated objects	12
2.8 On - Screen naming of associations	12
2.9 Association lifecycle	12
2.10 Association Subsets	13
2.11 Document all associations	13
2.12 Summary of association modeling	13

In chapter 1, we have elaborated the modeling of business entity types and their properties. We now turn to entity associations or relationships. In general, association modeling has much more of a technical connotation compared to entity modeling. Especially in entity-relationship modeling, the concepts of 1:N, N:M etcetera are difficult for business users to understand.

The modeling concepts proposed in this chapter are addressing this concern, introducing some novel ideas that make it easier for business users to articulate their needs. A basic assumption we make regarding associations is that we consider them to be part of an entity, very much like how properties are part of an entity. This is in accordance with the *Composite* pattern⁶.

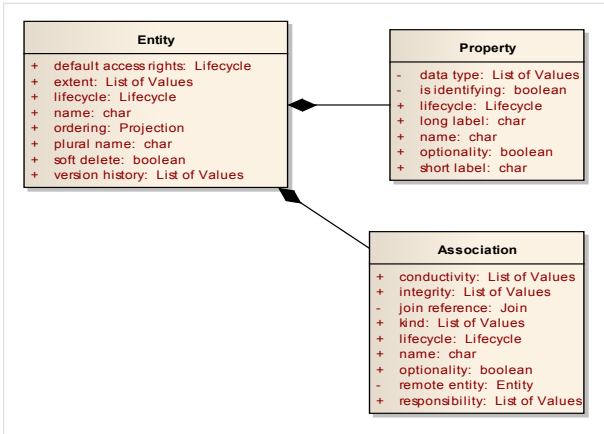


Figure 7 Association Modeling

2.1 Entity association: unidirectional

By definition, an association links two entity types, and the link can be considered from either side. However, the attributes of the association may be different, depending on the direction of the association.

Modeling Proposal

For this reason, we recommend considering *unidirectional* associations, in other words we treat the association from A to B separate from the one that links B to A. There is much more to an association than just its name, and we want to be able to accurately describe the desired behavior of an association for each of its directions. As an aside, with unidirectional associations it is much easier to consider each association as being part of an entity definition. Also, we now have the option to not model a reverse association at all.

2.2 Entity association: verb or noun

Associations can be named and in fact should be named. Existing modeling techniques use verbs (Information Engineering, IDEFIX), nouns (Chen, E-R Modeling) or prepositions (Barker, Oracle CASE).

Modeling Proposal

We recommend using nouns, as we consider the associated entities as properties (composite) of an entity. In this way, we think of a sales order’s line items as Order.lineltems, this being a collection of Lineltem instances. And because we recommend unidirectional associations, there is no need to think of the “reverse” name. There is of course a complementary association, from Lineltem to Order, which is part of the Lineltem entity model. That association may be named “order”, so that its details can be referred to using Lineltem.order.

2.3 Association integrity

In database schema design, data integrity is accomplished using constraints on the foreign keys residing in related data. In business entity modeling, there is no awareness of foreign keys or key constraints – the business is usually only interested in understanding the impact on associated entities in case of changes, especially when one end of an association is deleted. Referential integrity conflicts that arise due to a change in data values are of no concern to the business – such conflicts are a result of the technical implementation of referential integrity, only.

As we propose to model associations as being unidirectional, we have an opportunity to define the integrity constraints from two ends separately.

Modeling Proposal

One possible value for an association’s integrity is *delegated* which means that this end is not concerned about integrity, leaving integrity maintenance to the other side. The other possible values should at least include *cascade*, *restrict* and *null*, plus *ignore* which implies “I know I am responsible, but there is no business reason to maintain the integrity”.

We propose to consider an additional aspect of managing referential integrity, which is to define the ownership (responsibility) for maintaining the integrity. There are multiple candidate parties to take ownership; one is the underlying RDBMS, another one is the client code and a third one is the intermediate data access object (DAO) or data access framework (DAF) being used. As a matter of fact, the value of *ignore*, when presented to an RDBMS as part of a database schema, means that integrity is maintained somewhere else, and *cascade*, *restrict* and *null* all indicate that the RDBMS is responsible.⁷

2.4 Association kind

We introduce the concept of association kind to distinguish various classes of associations that differ in how business users interact with them. This concept combines ideas of relation cardinality and (in)dependency. The ideas introduced here are closer to the business users as they describe the desired behavior, rather than the technical consequences.

Modeling Proposal

A *dependent* association is one where the associated entity instance cannot exist without the associating entity. Think of the lineltems association for Order entities. The line items cannot exist without the order, so the lineltems association is a dependent association.

A *reference* association is one where the associating entity instance refers to (zero or) one associated entity instance, technically using a foreign key. In business terms, this behaves as a regular entity property that has a lookup value (or reference). As a matter of fact, in many cases the business will not even recognize this as an association;

⁶ Composite, pp. 163-173 in “Design Patterns”, Gamma, E., R.Helm, R. Johnson, J. Vlissides, 1995, ISBN 0201633612

⁷ Note: This concept of *responsibility* really is a first step towards implementation and need not be part of the initial business requirements gathering or domain modeling exercise.

they simply see it as a property with a list of allowed values⁸.

A *passive* association is one where there is never any active involvement from the associating entity's side. As a contrived example, consider a list of values that represents currencies. Within the Order entity model, there will be a reference to a given currency, through mention of the currency value from the list of values (a reference lookup). From the perspective of the Currency entity, there may be a complementary association towards Order which represents "orders placed in this currency". That association is itself a passive association, as the actual data associations are created and maintained from the Order end, not from the Currency end. Clearly, when considering a pair of associations between two entity types, it cannot be the case that both of them are passive, as in that situation there would never be a way to create such associations.

Now consider the Orders and Lineltems example. From the Lineltem perspective, the order association is passive, even though technically the foreign key resides at the Lineltem end, suggesting it is a reference. However it is not a reference association, as that would imply that one would be able to freely create an independent line item and subsequently tie it to an existing order.

As a consequence, for every dependent association from A to B, there is a complementary passive association from B to A. In practical terms, when adding line items to an order there is no active user involvement in establishing the association with the order. Instead, the line items are created in the context of one specific order, exclusively, and the association is implied from the start. It is not always necessary to model these passive associations as there may not always be a need to access the information in that way.

A *membership* association is one where the associated entities can exist without any associating entities being present (i.e. they are independent). From the associating entity end, an association is made by choosing one or more of the existing candidate entities. These associations couple two independent entity types, exactly the same as how a reference association would.

An example would be the association from a Book entity to one or more keywords. The keywords already exist, and the associations couple a book to zero or more of such keywords. The membership association is technically equivalent to an n:m relation in an E/R schema. Note that the complementary association from Keyword to Book need not be of membership type, in fact in most cases it will be of the passive kind.

2.5 Association conductivity

This is another concept that is novel to business domain modeling. Conductivity is related to the ability to track changes over time as mentioned in section 1.7 on page 6.

⁸ To us, data modelers, the difference between a reference and a property with a predefined list of allowed values is that through a reference, the allowed values are treated as *data*, whereas through a property with a list of allowed values, they are treated as *metadata*. This has consequences for data ownership, change management, etc.

What is modeled through conductivity is to what extent changes in an associated entity should be considered as changes in the associating entity (upstream).

When tracking changes over time, the underlying database management system would keep a copy of historic data (a version history). Using a *conductive* association, a change in an associated (downstream) data instance introduces a change in the associating (upstream) instance as well, leading to a version history snapshot being recorded for the associating record, even though none of its own properties changed.

Modeling Proposal

As it turns out, the ability to conduct changes depends on the association kind. Passive associations and reference associations are *non-conductive*. Changes in the associated entities cannot introduce a change in the associating instance.

For membership data, the conductivity at best covers addition and removal, but as associated data in this case is independent, changes to associated data should not lead to a change on the associating instance. So membership associations are either *partially conductive* or *non-conductive*.

Dependent associations can be *fully conductive*, i.e. any change to the associated data leads to a change in the associating entity instance. This includes addition, removal and updates to associated data. Of course dependent associations can also be *partially* or *non-conductive*; this is a modeling decision.

In extreme cases, it might be useful to consider the option of defining conductivity not at the association level, but rather for each property of the associated entity separately. In other words, between orders and line items, a change to the VAT tariff on a line item may be marked as non-conductive, whereas a change to the item price might be considered as conductive.

We do not recommend this approach as it introduces an overwhelming amount of additional information to be modeled with little incremental value.

2.6 Association cardinality: handful or numerous?

We introduced the concept of association *kind* in section 2.4 on page 10. The *dependent*, *membership* and *passive* associations are all collections, i.e. there can be more than one associated entity instance for every associating instance. Only the *reference* kind is always a singleton association.

Some modeling techniques allow specifying the *cardinality* of these associations exactly, as in "between 2 and 7". That is probably ideal, but in many cases the exact boundaries are not known up front. As an alternative, we consider to introduce a more vague specification, which has immediate impact on how data is handled interactively.

Modeling Proposal

Our recommendation is to distinguish between *handful*, *numerous* and *multitude*. When the cardinality is *handful*, it is understood that there are at best a few associated entities; say 5, or 10, or 12 maybe.

The implication is that there would never be a need to search amongst those, or a need to have paginated displays of associated data. The natural way of interacting with *handful* associations is that all associated data fits comfortably on a single screen.

By contrast, *numerous* or *multitude* associations have a larger number of associated instances, and this implies a much richer data management user interface.

Finally, *singleton* associations have at most one associated entity instance for every associating instance. All associations of *reference* kind are singleton associations, so the two terms describe the same thing.

Note that we introduced the concept of *extent* for entities earlier, in section 1.1 on page 3. Entity extent and association cardinality are very much the same thing – they record data volumes. It is perfectly OK for an association to have cardinality *handful*, whilst the associated entity itself has extent *numerous*. Of course the other way around is impossible.

We recommend modeling the *optionality* of associated data separate from its cardinality. An optional association can be, as the name suggests, non-present. In E-R terms, optionality is the difference between 1:N and 0..1:N.

2.7 Natural ordering of associated objects

For all collection-type associations (dependent, membership and passive), there may be a natural ordering of the associated data. In section 1.3 we introduced entity ordering; association ordering works in exactly the same way.

Modeling Proposal

The suggested modeling approach is to introduce a modeling attribute named “ordering” which is a comma-separated sequence of object properties (a.k.a. a *projection*, as described in section 1.12). The first property mentioned will be the major sort property, and in case of identical values on the major property, subsequent properties will be used for sub-sorting the query results. Every property in itself is either a simple entity property, or the property of an associated entity in which case the object property name must include the association path to the associated entity.

2.8 On-screen naming of associations

All associations have a name, as discussed in section 2.2. This may not always be the preferred on-screen name for the associated information.

Modeling Proposal

There is no need to introduce new names for reference-type associations as these associations effectively add a set of property values to an entity instance, each having a well-defined name plus long and short label already. Where needed, each of the referred properties can be re-labeled as part of a projection, too.

Hence, we consider association naming for collection-style associations, only. The name that we need to consider is a reference to the collection as a whole, not to any individual record, and certainly not to instance properties.

We refer to this name as the association *signpost*, as it will often be used as an on-screen navigation aid to drill down into the associated data.

Consider, as an example, a book library that tracks the individual takeouts of its books. For a given publication, there would be an association named takeouts that points to all takeout transactions of the book. In the publication details screen, there would be a navigational item (e.g. a menu item or a tab on a tabbed GUI), that would need a proper label for the collection of takeouts. That is what the signpost is providing. It could simply be “Takeouts”, or “History of takeouts” of “Takeout history” or “Loan log” or something.

2.9 Association lifecycle

Just as entities have a lifecycle (see section 1.4), entity associations also have a lifecycle. In general, the lifecycle of an association should be a subset of the lifecycle of the entity it is part of. As an example, a *viewable* entity should not have a *maintained* association defined as part of it.

Modeling Proposal

Table 4 Association lifecycle support

Lifecycle	Create	Retrieve	Update	Delete	Copy
Full	y	y	y	y	n
Maintained	y	y	y	n	n
Evolving	n	y	y	n	n
Permanent	y	y	n	n	n
Disposable	y	y	n	y	n
Visible	n	y	n	n	n
Invisible	n	n	n	n	n
Maintained (or full) with copy	y	y	y	(y)	y
Maintained (or full) by expert	y	y	y	(y)	(y)

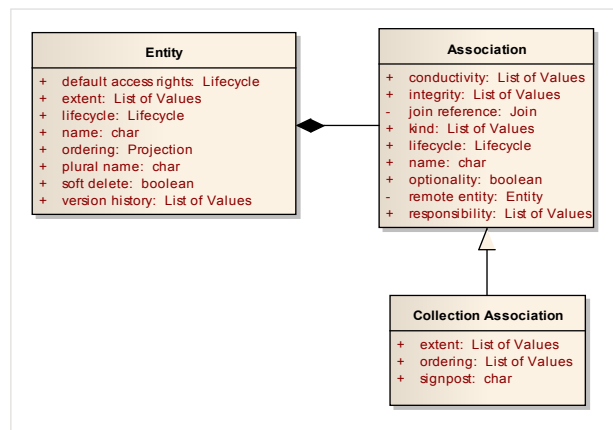


Figure 8 Association Modeling

2.10 Association subsets

All collection-type associations can have one or more subsets, just as entity types can have subsets. The purpose of this is to provide easy access to well-defined subsets of associated data.

Modeling Proposal

For association subsets, we propose to model its qualifier, name, its extent, and ordering.

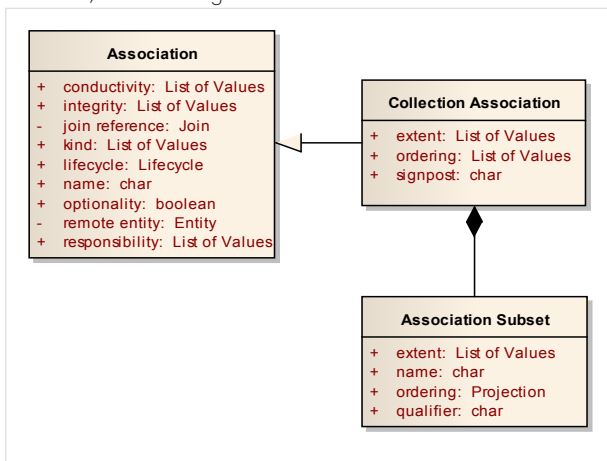


Figure 9 Association Subsets

2.11 Document all associations

We recommend documenting all entity associations, in the same way and for the same reasons we mentioned in section 1.13 on page 8.

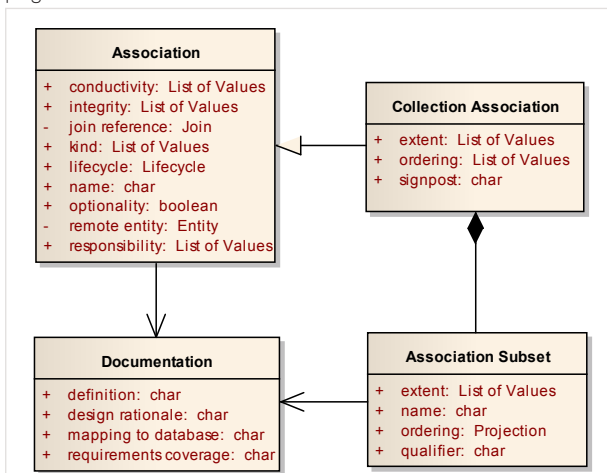


Figure 10 Association Documentation

2.12 Summary of association modeling

In this chapter, we have extended the simple entity modeling of Chapter 1 with entity associations. We proposed a modeling approach that aligns naturally with the business requirements gathering process.

Directionality

Associations are unidirectional so there is always an *associating* entity and an *associated* entity. These entities can also be considered in terms of *local* and *remote*. For association naming, we recommend nouns, rather than verbs.

Cardinality

Regarding cardinality, a unidirectional association always has exactly one instance on the local end, so there are only a few options for qualifying the remote end. Associations with at most one remote instance are defined as *singletons*. Associations with remote collections come in three forms: *handful*, *numerous* or *multitude*. The difference between *handful* and *numerous* is that on-screen interaction is quite different for small collections as compared to large collections.

Association *optionality* is introduced to model nullable versus non-nullable associations.

Kind

Association *kind* models how users interact with associated entities. For associated collections, the possible options are *dependent*, *passive* and *membership*. For associations with a single instance, the options are *passive* and *reference*.

Integrity

We propose to model both the desired *integrity constraints* and the *integrity responsibility*. From the business perspective, only the constraints matter, in other words the business should be concerned about what happens to remote data when a local instance is deleted. For subsequent technical modeling, the integrity responsibility defines whether the desired integrity should be maintained in business logic, as part of a data access framework, or in the RDBMS.

Naming and Ordering

Beyond noun-based association naming, all collection-style associations require a *signpost*, which is the on-screen label that refers to the collection as a whole. Association *ordering* provides a natural ordering to related data.

Association Lifecycle

An association lifecycle describes the allowed set of operations on an association, in analogy to an entity lifecycle.

Association Subset

Association subsets are introduced as an extension of entity subsets. Where an entity subset is predominantly used to provide value-based access control, association subsets are an effective means to organize subsets of associated data, based on some data value such as status, due date, ownership etc.

Association Conductivity

Finally, we introduce *conductivity* to model the effect of data changes in associated data. This is relevant only in the context of version histories and audit logs. The purpose of a conductive association is that a change in the associated data is reflected as a change to the associating instance as well, even when there are no data changes to the associating instance itself.

3. Access Control: Role Modeling

Chapter Contents

- 3.1 Capture business roles during business entity Modeling 14
- 3.2 Define role access rights using lifecycles 14
- 3.3 Use entity subsets for Data-Driven access rights 14
- 3.4 Create new roles using existing roles as baseline 15
- 3.5 Introduce default entity access right 15
- 3.6 Detail property-level access rights 15
- 3.7 Detail association - level access rights 16
- 3.8 Detail Service - level access rights 16
- 3.9 Introduce role - specific projections 17
- 3.10 Document all roles 17
- 3.11 Access control summary 18

Role based access control (RBAC), or any other type of access control for that matter, is not part of the *data* modeling that we elaborated on in the preceding chapters. It is orthogonal to that, defining who (which role) has access to a given entity type, and what the entitlements are for that role. These entitlements are expressed as a *lifecycle*, as defined in section 1.4 on page 3. There, we defined a lifecycle as an *allowed set of operations*, from the perspective of the entity itself, i.e. what operations should the entity type support. Now, in role modeling, we'll use the lifecycle again, this time to define the *role's accessible set of operations*. In other words, we use one of the lifecycle values to indicate how a given role has access to a particular entity. Of course, the accessible set can never exceed the allowed set, so there are constraints to the possible values for an entity's access life cycle.

3.1 Capture business roles during business entity modeling

As part of business requirements capture, we believe it is important to identify both the roles and the entitlements of each role. In other words, what are the roles identified by the business, and for each of those, what are the entitlements against every Entity type?

Modeling Proposal

We propose to model the *roles* separately from the *entities*, i.e. to keep business entities and business roles separate. This allows for a phase-wise implementation, starting without a role-based model first and introducing more and more roles as needed.

To support a *basic* access control model without any role-based control, we propose to introduce *default access rights* for all entities, and we elaborate this in section 3.5 below.

For role-based access control, every role needs to define only the access rights to the extent that they deviate from the default access rights.

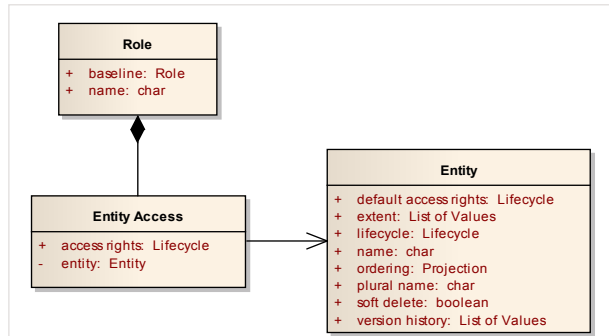


Figure 11 Role Modeling

3.2 Define role access rights using lifecycles

As stated, the concept of a *lifecycle* can be applied to role modeling as it represents the allowed lifecycle for a given role towards a given entity type.

Modeling Proposal

Table 5 Entity Access Control Model

Lifecycle	Description
Full	The role has full data access, including creation, retrieval, modification, removal and copying of entity instances. Only for entities that have lifecycle <i>full</i> .
Maintained	The role may create, retrieve, modify, and copy entity instances. Valid for entity lifecycles <i>full</i> or <i>maintained</i> .
Evolving	The role may retrieve entity instances and modify the details of those. Valid for <i>full</i> , <i>maintained</i> and <i>evolving</i> entity lifecycles.
Permanent	The role can retrieve existing instance data and create new instances, but there are no modification or removal rights. Valid for <i>full</i> , <i>maintained</i> , <i>permanent</i> and <i>disposable</i> entity lifecycles.
Disposable	The role can retrieve existing instance data and create or remove instances, but there are no modification rights. Valid for <i>full</i> , <i>maintained</i> and <i>disposable</i> entity lifecycles.
Visible	The role can retrieve existing instance data, only. Valid for all but the <i>invisible</i> lifecycle.
Invisible	The role has no access to the entity type in any way. Valid for any entity lifecycle.

3.3 Use entity subsets for data-driven access rights

There are many business scenarios where the access rights to business information are dependent on the information content itself. As an example, consider territory sales operations: a given sales representative may be allowed to *see* all opportunities and deals pending in her territory, but she can only *modify* the deals she owns. Deals outside the territory are totally *invisible*. The access rights to the sales information is dependent on two properties: ownership and region.

This is data-driven access control, as access is governed by the values of entity data being accessed. Rather than introducing an elaborate data-driven access control model, we propose to use *entity subsets* as defined in section 1.5 on page 4 of this report. Entity subsets can be used to define *target subsets* of an entity collection, and separate access rights can subsequently be set up to define the entitlements of a given role towards that subset.

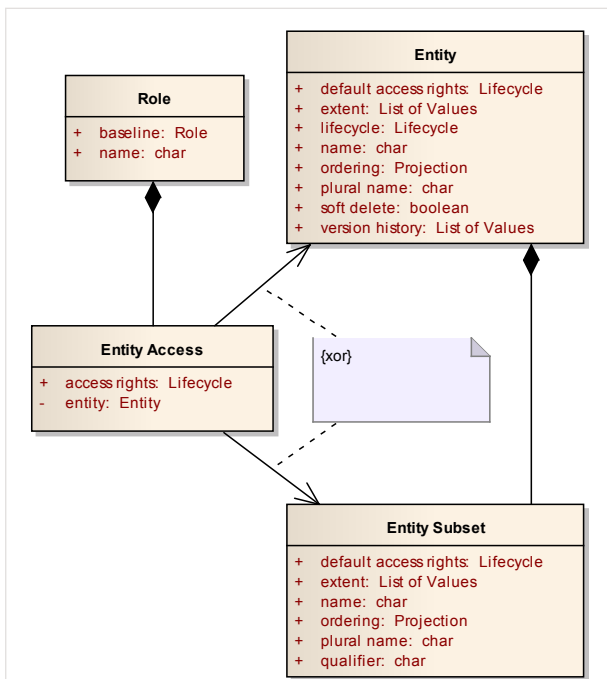


Figure 12 Data-Driven Access Control using Subsets

3.4 Create new roles using existing roles as baseline

In typical business scenarios, roles or user profiles are identified incrementally. There will be the concept of a standard business user role, followed by various managerial roles with more and more entitlements. In addition, the business will identify the need for roles with reduced access rights, for example reduced visibility on certain data sets, or read-only lifecycles, etc.

These follow-on profiles are very often specified in terms of their variation from the standard profile.

Modeling Proposal

We propose to follow this line of thought in formal role modeling, and allow each new role to be defined initially as being baselined from an existing role. All entitlements of the baseline role are passed to the new role, and for the new role all exceptions are modeled specifically (including limitations to the baseline as well as extensions from the baseline). As the baseline role itself can be baselined from a third role, the per-role differences are all considered in a cascading manner. In our modeling approach, the only change needed is the introduction of an optional *baseline* attribute on every role.

3.5 Introduce default entity access rights

With an entity lifecycle in place, there is a clear definition of the *allowed* data operations on an entity type. In reality, most business users will have reduced access rights; for them the lifecycle is a subset of the defined entity lifecycle.

We propose to model the *default entity access rights* to expose the intended lifecycle for the primary audience. These access rights are part of the Entity model, not of the Role model, to be used either in the absence of a role model or to provide a default for entity-level access as part of role modeling.

Modeling Proposal

The default access rights to an entity type define the most common type of access (maintained, permanent, disposable...) that business users will require. The default access rights are modeled again using a lifecycle, with the implied constraint that default access rights can never be more “open” than the underlying entity lifecycle.

It is suggested to also define the default access rights that apply as fallback across all entity types, further reducing the amount of information that needs to be captured. When the overall access rights across all entity types are *managed*, then it would make sense to specify this in one place and to not repeat this information for every entity model. At the entity level, one would only record the exceptions to the overall default.

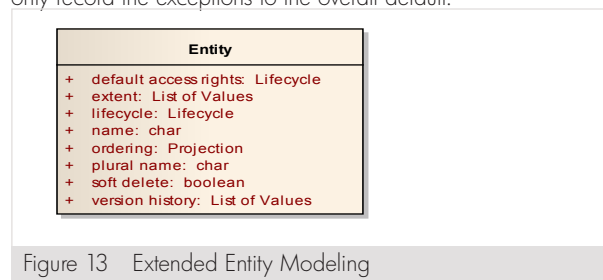


Figure 13 Extended Entity Modeling

3.6 Detail property-level access rights

The entity-level access control model introduced thus far applies to all parts of an entity model – there is no detailed access control at the property, the association or the service level.

In many cases, such detailed access control will be necessary, for example in situations where a given role cannot modify some data properties and a special management-level role is required to do so. This is common for approval status fields and the like.

Modeling Proposal

We propose to introduce property-level access rights by way of specialization: wherever such access rights are not mentioned, the entity-level access rights apply. In this way, we reduce the amount of extra information that is needed during requirements capture.

The possible lifecycle values that apply to property-level access control are those that we introduced earlier for the property lifecycle itself; see section 1.11 on page 7. Again, property-level access rights can never realize entitlements beyond the entity-level access rights.

Table 6 Property-level Access Control Modeling

Entity Lifecycle	Description
Maintained	Values can be set, modified and cleared as needed
Permanent	Values can be set at most once – once set the value cannot be modified or cleared
Visible	Values can be viewed but not modified in any way

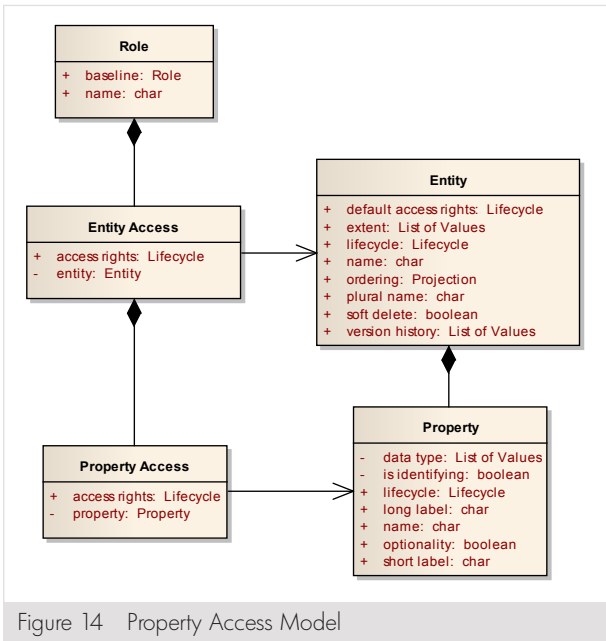


Figure 14 Property Access Model

3.7 Detail association-level access rights

For association-level access control modeling, we apply the same principles as for property-level access control.

Modeling Proposal

The allowed lifecycle values for associations are the same as those for entities:

Table 7 Association-level Access Control Modeling

Lifecycle	Description
Full	The role has full data access, including creation, retrieval, modification, removal and copying of entity associations. Only for associations that have lifecycle <i>full</i> .
Maintained	The role may create, retrieve, modify, and copy entity associations. Valid for association lifecycles <i>full</i> or <i>maintained</i> .
Evolving	The role may retrieve entity associations and modify the details of those. No new association instances or removal of existing associations are allowed. Valid for <i>full</i> , <i>maintained</i> and <i>evolving</i> association lifecycles.
Disposable	The role can retrieve existing association data and create or remove associations, but there are no modification rights. Valid for <i>full</i> , <i>maintained</i> and <i>disposable</i> association lifecycles.
Permanent	The role can retrieve existing association data and create new associations, but there are no modification or removal rights. Valid for <i>full</i> , <i>maintained</i> , <i>permanent</i> and <i>disposable</i> association lifecycles.
Visible	The role can retrieve existing association data, only. Valid for all but the <i>invisible</i> lifecycle.
Invisible	The role has no access to the entity associations in any way. Valid for any association lifecycle.

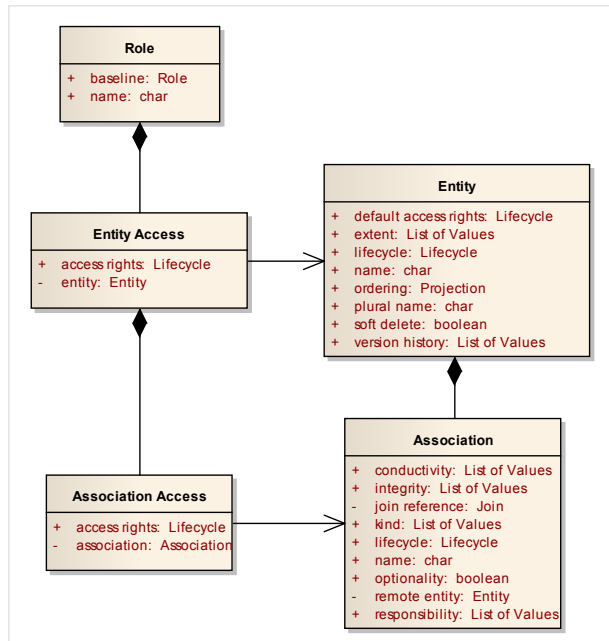


Figure 15 Association Access Model

3.8 Detail service-level access rights

In section 1.8, we introduced the concept of entity services. It should come as no surprise that we can include these services in our access control model. The business value of this is that it is now possible to indicate which additional entity-level tasks, beyond CRUD, are allowed for various roles.

Modeling Proposal

The proposed service model results in an enumeration of services applicable to an entity type (see Figure 3 on page 7). Within a role, we propose to model service access by replacement, i.e. whenever service access is modeled as part of a role, this model now defines the full access to an entity's services. Any service not mentioned is simply not accessible. There can be no services introduced in the role that are not already defined in the entity model.

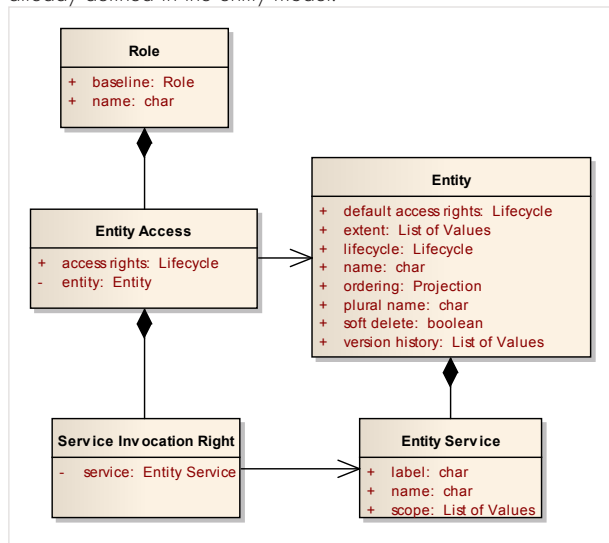


Figure 16 Service Access Model

3.9 Introduce Role-specific Projections

With an access model in place, a final part of modeling relates to refining the entity projections. This provides a means to specify the screen data visibility in a role-specific way. This avoids the situation where data elements are defined to appear on screen whilst the user's role does not allow data visibility.

Modeling Proposal

We propose to model role-specific projections on the basis of replacement: in the absence of a role-specific projection, the corresponding entity projection will be applied. Only when a role-specific projection is modeled will it be considered as a replacement for the original entity projection.

Projection names can be used to link the role-specific projection with its master. A role-specific projection should not carry any data elements not considered in the master.

3.10 Document all roles

We believe role documentation is as important as entity documentation. The type of documentation is different, but only slightly.

Modeling Proposal

For every role, we suggest to document:

- the role definition
- design rationale (why is this role introduced, what alternatives were considered and why were they rejected)

Subsequently, we recommend to document all entity access models as well as all property and association access models as follows:

- design rationale (why is an entitlement introduced, what alternatives were considered and why were they rejected)
- business requirements satisfied by introducing the access model. Ideally, such requirements are classified as being either policy or process requirements – these have varying levels of stability

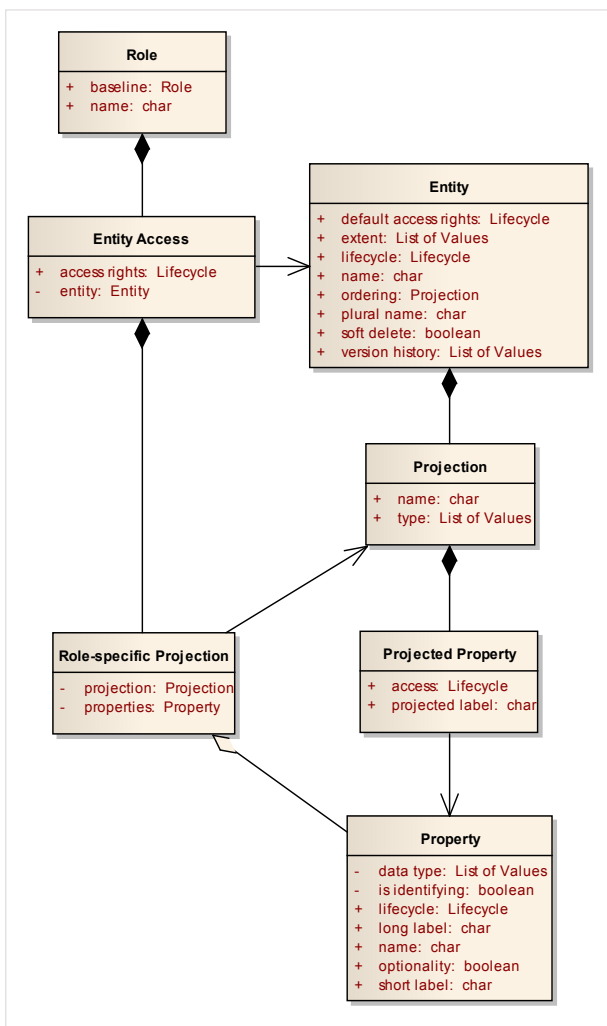


Figure 17 Role-specific Projection Modeling

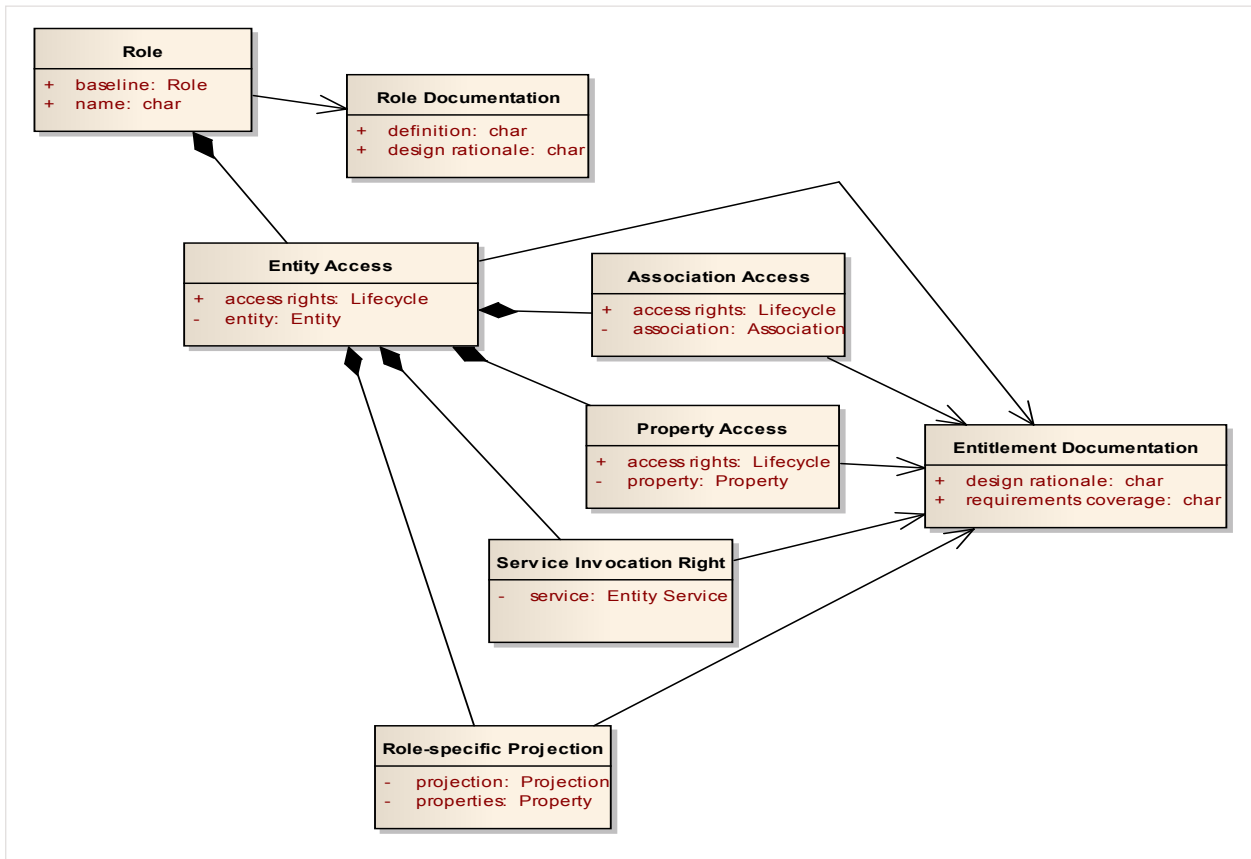


Figure 18 Access Control Documentation

3.11 Access control summary

In this chapter, we have introduced a very detailed access control model, with a very precise modeling approach. Care has been taken to ensure that this approach is still practical, by providing sensible defaults wherever possible.

Default Access Control

When no information is captured on access control in any way, then access to all data defaults to the entity lifecycle as modeled for each entity. There is a recommendation for a default lifecycle which is *maintained*, so even when no lifecycle information is provided, there is at least a baseline access control (one that allows full create/modify/read access to all data).

The minimal access control specification is a domain wide access control level, valid across all entity types. This could be any desired level, i.e. *full*, *maintained*, *evolving*, *permanent*, *disposable* or *visible*.

More detail can be provided at the entity level, through default entity access rights. This would allow different access rights per entity type, but still would not introduce any roles.

Role-Based Access Control

Through the notion of role modeling, it is possible to provide different access levels on the same entity type to different user groups. We do not specify how a user is associated with a role, but assuming that every user has a well-known role, then RBAC provides highly differentiated access control based on that role.

The role-based entitlements are set per entity type and can optionally extend to entity properties, associations, services and role-specific projections.

Value-Based Access Control

We have introduced value-based access control through the concept of entity subsets. For every (combination of) data value(s) that requires specific access control, an entity subset is introduced in the entity model. Any roles may specify entitlements towards these subsets in exactly the same way as how they are specified against the entity types themselves. And subsets have default access rights as well.

4. Database: Schema Modeling

Chapter Contents

4.1 Table purpose	19
4.2 Table column purpose	19
4.3 Column control – Who is Responsible?	19
4.4 Column Encodings	20
4.5 Denormalization	20
4.6 Relationships and the join clauses that define them	20
4.7 Subtyping and how generalization is to be implemented	20
4.8 Change tracking at the table level	21
4.9 Document all tables and columns	21
4.10 Summary of schema modeling enhancements	22

This chapter introduces some novel concepts related to logical data modeling. The field of relational modeling itself is already a mature space, and no breakthrough concepts are to be expected. However, when it comes to object/relational mapping, and in particular mapping to existing database schemas, there is opportunity for improvement. This chapter introduces some of that. The aim is to provide, as part of the schema model, sufficient information for automation support, formal documentation etc.

4.1 Table purpose

Every table that is part of a database scheme serves a particular purpose. Most tables carry business information (no surprise here); other tables are present only as a technical necessity (merge tables for N:M relationships are one example). Beyond these, there may be tables carrying historic data and tables with audit data.

Modeling Proposal

We propose to mark every table's purpose as a required piece of information. Not just for documentation purposes, but also to support automated version management and auditing either at the RDBMS or at the data access level. Typical values needed for a table's purpose are:

Table 8 Modeling table purpose⁹

Table Purpose	Description
table carries current business data	These are the standard tables of the relational database
table carries historic business data	When version history is supported for an entity type, there will be either a separate table with historic values or a table that combines current and historic data
table combines current and historic business data	
table is a merge table	This is technical support data without business meaning
table carries historic merge data	
table combines current and historic merge data	
table is an audit log	This type of table support simple logging of relevant business transactions
table carries unit of work information	This type of table supports user-level undo as an extension of version history

4.2 Table column purpose

Most table columns normally carry business information; however there are many technical reasons for adding non-business data to a table definition. Some of these reasons include:

- need to capture the effective user identity (who changed the information most recently)
- need to capture the valid period (both an as-of date and until date)
- synthetic key
- foreign key

- need to capture the most recent operation (including logical delete)
- type discriminator (for tables carrying a type hierarchy)
- need to capture the record version number (for locking)

All of the above reasons apply to tables that carry business data; for tables carrying historic data, or for audit logs, there are additional technical columns.

Modeling Proposal

Similar to table purpose modeling, column purpose modeling provides both documentation and support for automation of version histories, optimistic offline locking, logical delete and management of type hierarchies (inheritance support).

In addition, automated generation of physical database schemas requires this level of detail to be captured.

Table 9 Modeling column purpose

Column Purpose	Description
asOfDate	Start of valid period for this record
effectiveUser	Identity of the user who manipulated this record last
logDetails	Column captures log details (in an audit log table)
identifier	Column is (part of) a primary key
foreignKey	Column is (part of) a foreign key
mostRecentOperation	Most recent operation on this data row (i.e. insert, update, delete or bubble)
rowKey	Column carries a row identifier (in a unit of work table)
tableName	Column carries a table identifier (in a unit of work table)
typeDiscriminator	Supports type hierarchies
unitOfWork	Row identifier towards the unit of work table (in a version history)
untilDate	End of valid period for this record
unrestrained	Business data
version	Version number for data (locking)
denormalized	Column carries denormalized information

4.3 Column control – who is responsible?

As part of the design exercise, there comes a point when a decision is made as to where certain logic is implemented. Related to data management, content is created either by the end users, by the relational database management system, or by the intermediate data access framework. As an example, denormalized data can be managed through triggers and stored procedures in the RDBMS, and synthetic keys are routinely generated by the RDBMS. Business data, with column purpose marked as unrestrained, is controlled by the end users i.e. both the data access framework and the RDBMS are passive participants.

Modeling Proposal

Capturing the control over every table column is again a valuable piece of documentation. Also, it is indispensable for automation at the data access level, as this information provides instructions to the DAF about what it is responsible for.

⁹ For an in-depth discussion of the various strategies for managing historic values, see the earlier white paper "Understanding Application Audit Requirements", available at http://www.mphasis.com/knowledgeBank/white_papers.asp.

Finally, all columns marked as “RDBMS is responsible” together form a statement of work for the technical database designer (having to create triggers, stored procedures and such).

The control property of every column model can take one of three possible values:

- *controlled* – the data access framework is in charge
- *calculated* – the RDBMS is in charge
- *unrestrained* – no responsibilities at the data access tier or below

4.4 Column encodings

This appears to be a rather mundane and low-level modeling exercise at first sight, however it is indispensable information for any data access component, and it serves an important role in the design documentation of a logical database schema.

Modeling Proposal

Column encodings are required on several types of columns. Referring back to the column purpose, immediate examples are *mostRecentOperation* and *typeDiscriminator*. For these, the modeling exercise must include:

- identification of all possible encodings needed (keys)
- choice of appropriate code values for all encodings (values).

For a *mostRecentOperation* column, there are only a few possible encodings, i.e. *insert*, *update*, *delete* and *bubble* (bubble being the change caused by conductivity (as per section 2.5 on page 11)). For a *typeDiscriminator* column, the possible encodings depend on the subtypes to be supported.

4.5 Denormalization

For reasons of performance optimization, a database schema may be denormalized. Denormalized columns carry either data copies sourced from another table, or aggregates of such data. Technically, columns having calculated data are also denormalizations, at least for calculations that are based on data in other columns. We address calculated data separately in section 4.3 because it’s important to understand where data is generated. But for denormalization we focus on data that is transferred from one table to another, either as a copy or through aggregation.

Modeling Proposal

For a column carrying denormalized data, we propose to model three relevant attributes:

- a *formula* – one of *min*, *max*, *sum*, *avg*, *count* and *copy*. This defines the nature of the denormalization; all except *copy* are aggregate functions
- a *reference to another table* – indicating the source table where the data is to be copied from. Rather than marking the name of the source table, we recommend to refer to a *join specification*, which is defined in section 4.6 below
- a *column reference* – pointing to a source column in the related table

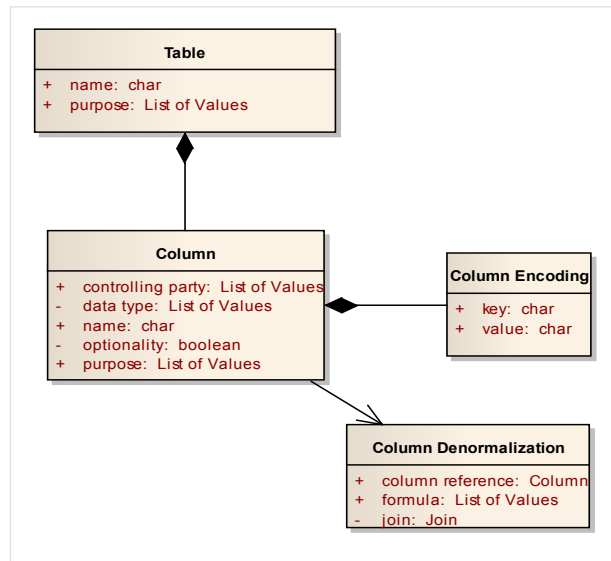


Figure 19 Modeling tables and columns

4.6 Relationships and the join clauses that define them

All business entity associations modeled as per the preceding chapters need to be backed by a relationship between two (sometimes three) tables. In addition, denormalization refers to source tables through an inter-table relationship as well.

In the physical schema, relationships are at best represented by integrity constraints. Given the importance of relationships in the overall context, we believe that relationships should be modeled explicitly, as first-class citizens.

Modeling Proposal

Every relationship has a name and a specification. Through the name, it is possible to refer to a relationship, for example as part of a denormalization. The specification of a relationship is best expressed using SQL concepts. It is the JOIN clause of an SQL statement. Hence, we refer to it as a *join specification*. The join specification includes table and column names (primary and foreign key columns). An example from the Orders and LineItems domain could be: `orders.orderID = lineitems.orderID`

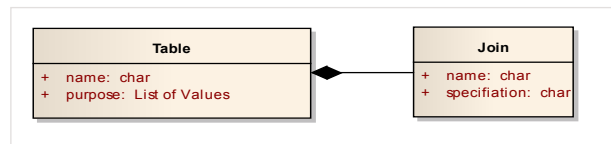


Figure 20 Modeling Relationships

4.7 Subtyping and how generalization is to be implemented

In the object space, inheritance is a fairly straightforward concept, especially when constrained to single inheritance. In the database world, even single inheritance has many different implementation options. Without going into the details, there are essentially two relevant implementation patterns¹⁰:

¹⁰ There is a third pattern, which has supertype and subtype combined in one table, for each table. It’s not a very practical setup, but whenever this is chosen, the constituting tables no longer carry any relationship with each other, and subtyping does not even have to be modeled in any way.

- Data for all subtypes is combined in one table, possibly a table with a large number of columns (such a table carries a *type hierarchy*)
- Data for the *supertype* sits in one table, and for each *subtype* another table is introduced. In this case, the primary key of the subtype tables doubles as a foreign key towards the supertype table (a 1:1 relationship).

For a single table, it is possible to be both a supertype and a subtype table simultaneously, or a subtype and a type hierarchy, or a type hierarchy and a supertype.

Modeling Proposal – Generalization

For generalization (or subtyping), the proposed modeling construct is to mark the desired subtyping involvement of a table as one of the following:

Table 10 Modeling subtypes

Subtyping	Description
none	The table is not part of any subtyping
type hierarchy	The table implements a type hierarchy
subtype plus type hierarchy	The table acts as a subtype (with the supertype in some other table) and simultaneously as a type hierarchy
supertype plus type hierarchy	An unusual combination where most subtypes are implemented using the type hierarchy pattern, but some subtypes have their own private table
subtype	The table implements one single subtype, only
subtype plus supertype	The table is a subtype and is itself further subtyped in other tables
supertype	The table implements a pure supertype

Modeling Proposal – Type Hierarchy

When using the type hierarchy implementation pattern, one single table shares supertype attributes with the union of all subtype attributes. When the inheritance is *exclusive* (see page 11), the table will need a column of purpose *type Discriminator*, and the encodings applied to this table can be freely chosen. If, however, the inheritance is *inclusive*, then any row in the type hierarchy table can represent multiple subtypes simultaneously. Now, care must be taken when choosing encoding values. What is needed is a bitmask, i.e. the appropriate encoding values should be 0x1, 0x2, 0x4, 0x8, 0x10, etc. In this way, encoding values can be logically or-ed together to indicate that a given data row represents multiple subtypes.

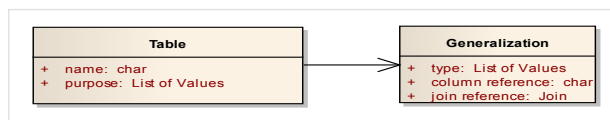


Figure 21 Modeling Subtypes and Supertypes

Modeling Proposal – Supertype

Whenever a table implements a supertype, there are no further modeling requirements. As a recommended option, the table can have a column with purpose *typeDiscriminator*, as per the type hierarchy pattern. Without such a column, it will be difficult to evaluate the subtype or subtypes of a given supertype data row.

The type discriminator will provide this information (but strictly speaking it is redundant, as the subtype tables all have foreign keys back to the supertype).

Modeling Proposal – Subtype

For a table that implements the subtype pattern, the relevant modeling information pertains to the table that holds the supertype attributes. Again, rather than referring to that table by name, the recommended practice is to refer to it through mention of a *join reference*.

4.8 Change tracking at the table level

We introduced change tracking in section 1.7. Ultimately, change tracking is implemented as an integral part of the relational schema of your database.

Modeling Proposal

On every data table that is part of the core data model, we model change tracking as follows:

Table 11 Change Tracking

Attribute	Description
type	Defines what type of change tracking is expected for this table. Possible values are <i>none</i> , <i>audit log</i> , <i>version history</i> and <i>unit of work</i> , in order of increasing complexity.
controlling party	Defines where the version history is maintained. Possible values are <i>controlled</i> , <i>calculated</i> and <i>unconstrained</i> .
soft delete	Not strictly part of tracking history values, this attribute informs about whether a delete operation should result in a physical removal of data or not.
table reference	This identifies the table where changes are held. It may be the same table as the data table itself, for version histories.

With the above specification in place, most of the information for change tracking is in place. The only missing piece is the instructions of how data is copied to the history table, on a column by column basis. This *mapping* defines, for every source column to be tracked, the name of the column in the history table where the data is held.

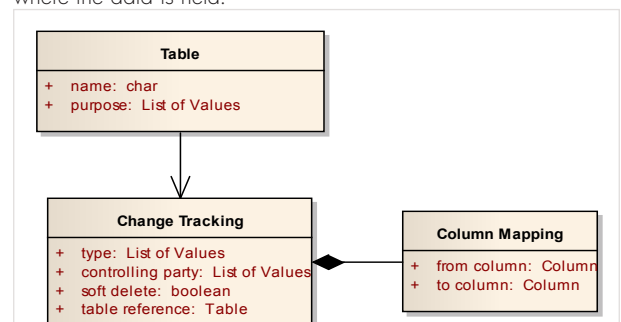


Figure 22 Modeling Change Tracking

4.9 Document all tables and columns

We already indicated that most of the modeling in this chapter is in and by itself an excellent source of documentation. Beyond that, we recommend to capture the following documentation elements:

Table 12 Documentation at the Database Schema Level

Documentation	Description
design rationale	Why is a table or column or other element modeled in this way, and what are the alternatives that are rejected?
null value documentation	For columns only, document why the data can be null. In addition, mark the interpretation of a null value as being one of <ul style="list-style-type: none"> - there is no value - the value is unknown - for the item type, the data element does not apply - in the current state, the data element does not yet apply
redundancy	for any column or relation that is redundant, document why the redundancy is introduced

4.10 Summary of schema modeling enhancements

In this chapter, we have introduced a number of modeling ideas that extend the current (logical) database design space. In particular, we have elaborated the modeling of relations and the management of change over time.

We also introduced the concept of control – which party is responsible for certain behavior. Beyond that we talked about modeling the purpose and intent of every table and all of its columns.

We introduced denormalization as an explicit modeling exercise, and we covered encodings for enumerated data. Finally, we covered the concept of subtyping in its various forms.

5. Concluding Remarks

Chapter Contents

5.1 Complete Meta - Models	22
5.2 Future Work	25

5.1 Complete Meta - Model

We summarize our proposed modeling concepts in three meta-model diagrams. In these diagrams, all features discussed in this report are marked with a “+” sign, and features not discussed are indicated with a “-” sign.

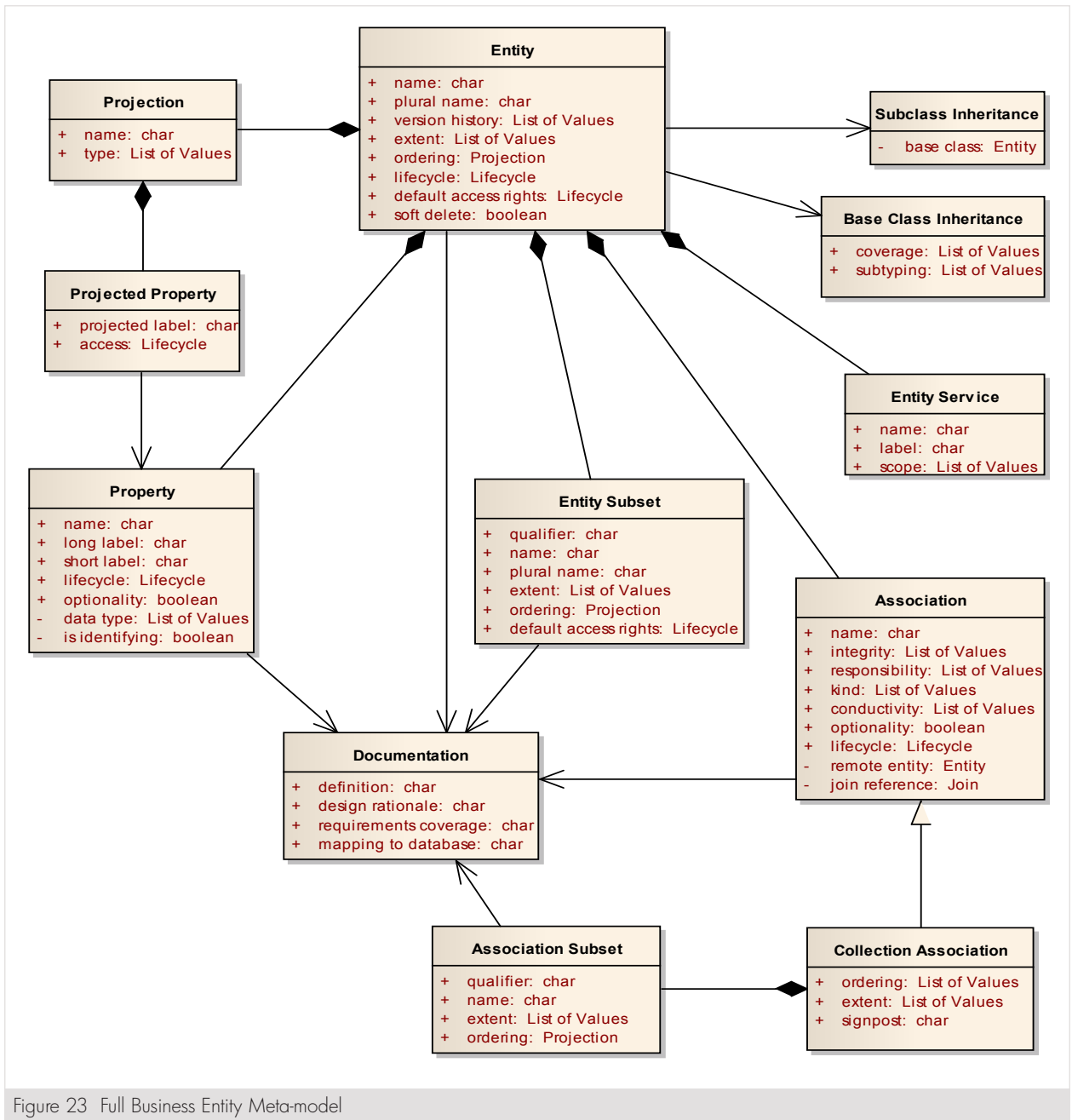


Figure 23 Full Business Entity Meta-model

Figure 24 Full Access Control Meta-model

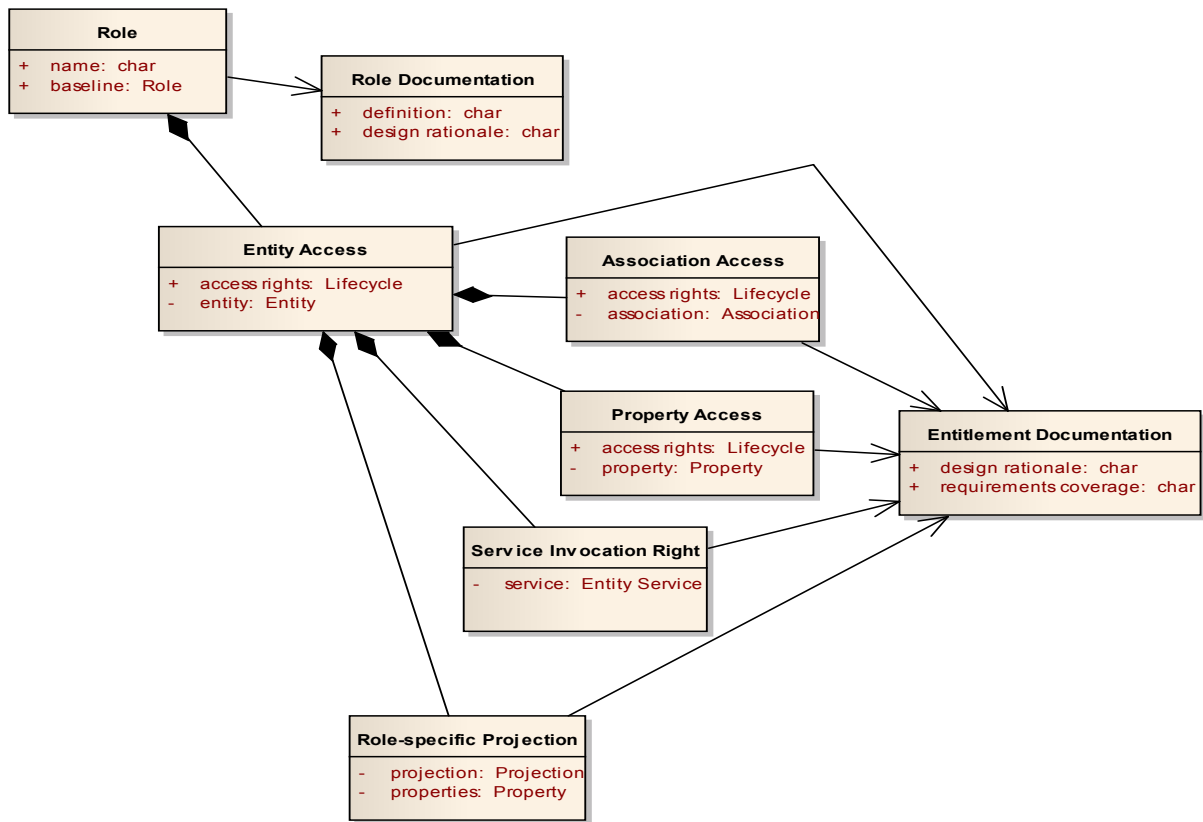
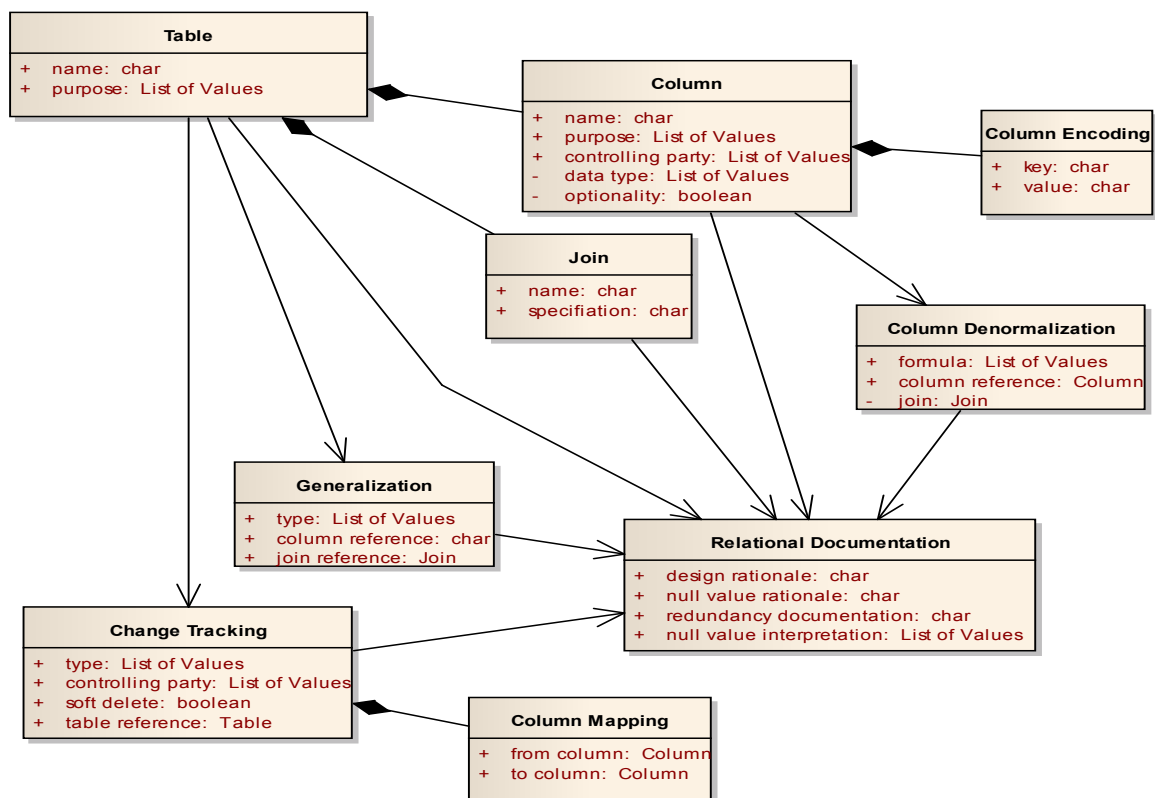


Figure 25 Full Schema Meta-model



5.2 Future Work

Automation of the data access code

Based on our earlier analysis of patterns for data management¹¹, we believe it is possible to fully automate the generation of data access code from models created using our proposed modeling concepts. Work is underway to generate a full data access tier in Java.

Automation of the data management user interface

The modeling enhancements proposed in this report have a strong focus on alignment with business requirements gathering – we believe the success of a data modeling exercise is critically dependent on the ability to capture the requirements related to the information model *early on*, before technical data modelers appear on stage. Many business-specific expectations are not directly concerned with the data model itself, but rather with how the end user *experiences* the business information. For that reason, many of our proposed modeling concepts have a direct effect on the end-user experience, i.e. the user interface for data management. Because of this, the resulting model has sufficient information to generate a satisfying user interface, or at least a comprehensive *specification* of such an interface. A patent application for the latter has been filed with the USPTO¹².

Introduction of a dialog-based modeling tool

Plans exist to develop a lightweight modeling tool that uses a dialog-driven interaction style, rather than a graphical user interface. Using dialogs, the full richness of an elaborate domain model can be developed incrementally, focusing first on entities and their associations, adding properties and projections only once the overall domain structure is in place. Advanced concepts including role modeling, tracking changes over time and logical schema mapping can be added later.

Legacy schema integration

In an earlier white paper, we analyzed the requirements for mapping an object model onto a legacy database¹³. The modeling techniques introduced in the current paper, especially entity subsets and table-level subtyping, go a long way towards support for legacy schema mapping.

We believe that the concept of a *view*, as known in the relational database world, may also be applied in the object/relational mapping tier. Here, the view would be resolved to physical table(s) before the RDBMS is accessed. It would act as a virtual table that only exists in the O/RM tier. This would allow working against existing legacy schemas without any modification of those schemas.

Such views can easily accommodate the intricacies of

transforming and combining content from multiple columns across multiple tables. Views would be modeled similar to tables, as discussed in chapter 4.

The difference would be in the modeling of columns, which, for a view, would describe how the virtual column is derived from one or more physical columns.

What would be needed to effectively describe the physical-to-virtual column content transformation is a simple transformation language, for example something like SSIS Expression Language¹⁴ or the Excel Formulas.

Constraint-based validation modeling

The modeling concepts proposed in this paper do not cover any model constraints or necessary validations. We believe that property-level validation is a subset of the broader model validation, which in turn can be captured in terms of constraints. Similar to how our model layers role-based access control on an underlying entity model, we expect that a constraint model can be added in much the same way, as an extension to the present model.

¹¹ See "Data Management: Repositories v1.0", available on the web at http://www.mphasis.com/knowledgeBank/kbank_whitepapers_all.html#2005

¹² "GENERATOR FOR DATA MANAGEMENT APPLICATIONS" is the property of HP and is the subject of one or more pending U.S. patent applications. The information in this document concerning the features and operation of "GENERATOR FOR DATA MANAGEMENT APPLICATIONS" is the copyrighted, confidential, and proprietary information of HP and may not be copied, used, or disclosed without the express permission of HP.

¹³ See "Integrating Legacy Data", available on the web at http://www.mphasis.com/knowledgeBank/white_papers.asp

¹⁴ SQL Server Integration Services is a platform for building enterprise-level data integration and data transformations solutions. See for example "SSIS Expression Language and the Derived Column Transformation", available on the web at http://sqlblog.com/blogs/andy_leonard/archive/2009/02/04/ssis-expression-language-and-the-derived-column-transformation.aspx

6. Glossary

Abstract – A possible value for coverage.

Access Rights – Access rights define the entitlements of a user (role) in terms of allowed operations, using lifecycle values.

Audit Log – A possible value for version history.

Baseline – Part of role modeling, a baseline role is the starting point of a role definition.

Calculated – A possible value for column control.

Cardinality – See extent.

Cascaded – A possible value of integrity.

Change Tracking – The equivalent to version history at the object level, change tracking models how history tables, audit logs and unit of work tables are treated.

Collection – Part of association modeling, collections are a subtype of association.

Column Control – Part of schema modeling, this attribute describes where data for this column is handled.

Column Encoding – For several column purpose types, the column content is in the form of an encoding scheme.

Column Mapping – Part of change tracking, column mapping describes how data from a current values table is copied to a history table.

Column Purpose – In database schema modeling, every table column is defined as serving a particular purpose.

Column Reference – Part of column denormalization.

Complete – A possible value for coverage.

Conductivity – Part of association modeling, the conductivity of an association defines how changes in the associated entity impact the version history of the associating entity.

Controlled – A possible value for column control.

Coverage – Part of the modeling of object inheritance, coverage describes whether every instance of the base class is always a member of at least one subclass or not.

Delegated – A possible value of integrity.

Denormalization – Part of schema modeling, column denormalization models how a particular data table column is derived from other data present in the database.

Dependent – A possible value for association kind.

Disposable – A possible value for lifecycle.

Encoding Key – Part of column encoding.

Encoding Value – Part of column encoding.

Evolving – A possible value for lifecycle.

Exclusive – A possible value for subtyping.

Expert Mode – A possible value for lifecycle.

Extent – The extent of an entity or of an association models the expected number of instances of that particular element.

Formula – Part of column denormalization.

Full – A possible value for conductivity.

Full – A possible value for lifecycle

Generalization – Part of schema modeling, generalization (or subtyping) is the database equivalent of inheritance.

Handful – A possible value for extent.

Ignored – A possible value of integrity.

Inclusive – A possible value for subtyping.

Incomplete – A possible value for coverage.

Integrity – Part of association modeling, integrity captures the expected system behavior when an entity instance on one end of an association is removed.

Invisible – A possible value for lifecycle.

Join – A join is the database equivalent of an entity association. It models the relationship between tables.

Join Specification – Part of a join, the join specification is the SQL clause that describes the technical join condition.

Kind – Part of association modeling, the association kind is used to distinguish various classes of associations, i.e. dependent, reference, membership and passive associations.

Lifecycle – The allowed set of operations on an entity instance or an association instance.

Logical Delete – See soft delete.

Long Label – One of two descriptive attributes of an entity property. The other one is short label.

Maintained – A possible value for lifecycle.

Membership – A possible value for association kind.

Non-Conductive – A possible value for conductivity.

None – A possible value for version history.

Null – A possible value of integrity.

Numerous – A possible value for extent.

Optionality – An attribute of an entity property. It models the notion of a “required property”.

Partial – A possible value for conductivity.

Passive – A possible value for association kind.

Permanent – A possible value for lifecycle.

Projected Label – Another type of label, modeling the ability to make a property label specific to a particular projection (i.e. a context-specific property label)

Projection – The projection of an entity describes a collection of properties considered for inclusion to accomplish a certain user task. A projection thus describes the data elements that need to be visible on screen.

Qualifier – The specification of a matching criterion for entity and association subsets.

Reference – A possible value for association kind.

Responsibility – Part of modeling association integrity, the responsibility attribute defines where in the system the integrity is maintained.

Restricted – A possible value of integrity.

Role – A role defines a set of entitlements. Users access data whilst working under a specific role.

Service Modeling – The idea of capturing user-visible behavioral aspects of business entities.

Service Scope – Part of service modeling.

Short Label – One of two descriptive attributes of an entity property. The other one is long label.

Signpost – A descriptive attribute of an entity association.

Singleton – A possible value for extent.

Soft Delete – Also known as logical delete, soft delete models the requirement that data, once removed by an end user, is not truly to be removed physically from the system but merely marked as being deleted.

Storyboard – Part of the model documentation, the storyboard is the top-level documentation for the complete domain model. It consists of business task documentation elements.

Subset – An entity or association subset is a qualified subset of the total set of instances, based on a matching criterion.

Subtype – A possible value for generalization.

Subtyping – Part of the modeling of object inheritance, subtyping describes whether any instance of the base class can be an instance of at most one subclass or not.

Supertype – A possible value for generalization.

Table Purpose – In database schema modeling, every table is defined as serving a particular purpose.

Type Hierarchy – A possible value for generalization.

Unidirectional – A modeling choice: our recommended approach to association modeling is based on separating the association linking A to B from the association linking B to A.

Unit of Work – A possible value for version history.

Unrestrained – A possible value for column control.

Version History – Version history models how changes over time are captured.

Version History – A possible value for version history.

View Modeling – View models are virtual database views that are resolved in the object-relational mapping tier rather than in the RDBMS.

Visible – A possible value for lifecycle.

7. References

- "Data Model Patterns – Conventions of Thought", Hay, David C., Dorset House Publishing, New York, USA, 1996. ISBN 0932633293
- "Data Modeling Essentials", 3rd edition, Simsion, Graeme C., G. C. Witt, Morgan Kaufman Publishers, San Francisco, USA, 2005. ISBN 9780126445510
- "Data Modeling for Everyone", Allen, Sharon, Apress, Berkeley, USA, 2003. ISBN 1590592131
- "Data Modeling for the Business", Hoberman, Steve; D. Burbank; C. Bradley, Technics Publications LLC, Bradley Beach, USA, 2009. ISBN 9780977140077
- "Design Patterns", Gamma, Erich; R.Helm; R. Johnson; J. Vlissides, Addison-Wesley, Indianapolis, USA, 1995. ISBN 0201633612
- "Fundamentals of Database Systems", Elmasri, Ramez and Shamkant B. Navathe, Benjamin/Cummings, Redwood City, USA, 1994. ISBN 0805317538
- "Information Modeling and Relational Databases", Halpin, Terry, Morgan Kaufman Publishers, San Francisco, USA, 2001. ISBN 9781558606722
- "Integrating Legacy Data - Dealing with Legacy Data and Legacy Data Schemas", Bert Hooyman, MphasiS, 2009, available at www.mphasis.com/knowledgeBank/kbank_whitepapers.html
- "Object Design – Roles, Responsibilities, and Collaborations", Wirfs-Brock, Rebecca; A. McKean, Addison Wesley, Boston, USA, 2003. ISBN 0201379430
- "Object-Oriented Modeling and Design", Rumbaugh, James; M. Blaha; W. Premerlani; F. Eddy; W. Lorensen, Prentice Hall, Englewood Cliffs, USA, 1991. ISBN 0136298419
- "Repositories for Data Management – A Data Access Pattern Language", Bert Hooyman, MphasiS, 2005, available at www.mphasis.com/knowledgeBank/kbank_whitepapers_all.html#2005
- "Smalltalk, Objects, and Design", Liu, Chamond, iUniverse.com Inc., Lincoln, USA, 2000. ISBN 1583484906
- "Streamlined Object Modeling – Patterns, Rules, and Implementation", Nicola, Jill; M. Mayfield; M. Abney, Prentice Hall, Upper Saddle River, USA, 2002. ISBN 0130668397
- "The Data Model Resource Book", Silverston, Len; W.H. Inmon; K. Graziano, John Wiley & Sons, New York, USA, 1997. ISBN 0471153648
- "Understanding Application Audit Requirements", Bert Hooyman, MphasiS, 2008, available at www.mphasis.com/knowledgeBank/white_papers.asp

Contact us

USA

MphasiS
460 Park Avenue South
Suite # 1101, New York
NY 10016, U.S.A.
Tel: +1 212 686 6655
Fax: +1 212 686 2422

UK

MphasiS
88 Wood Street
London EC2V 7RS, UK
Tel: +44 208 528 1000
Fax: +44 208 528 1001

AUSTRALIA

MphasiS
410 Concord Road
Rhodes, NSW 2138, Australia
Tel: +61 290 221 146
Fax: +61 290 221 134

INDIA

MphasiS
Bagmane Technology Park
Byrasandra
C.V. Raman Nagar
Bangalore 560 093, India
Tel: +91 80 4042 6000
Fax: +91 80 2534 6760

About MphasiS

MphasiS is a \$1 billion global service provider, delivering technology based solutions to clients across the world. With currently over 39,000 people, MphasiS services clients in Banking and Capital Markets, Insurance, Manufacturing, Communications, Media & Entertainment, Healthcare & Life Sciences, Transportation & Logistics, Retail & Consumer Packaged Goods, Energy & Utilities, and Governments around the world. Our competency lies in our ability to offer integrated service offerings in Applications, Infrastructure Services, and Business Process Outsourcing capabilities. We are uniquely positioned to offer our clients the highest level of expertise and competitive costs. To know more about MphasiS, log on to www.mphasis.com