

Data Modeling

From Conceptual Model to DBMS

All material (c) Sparx Systems

<http://www.sparxsystems.com>

Table of Contents

INTRODUCTION.....	3
MODELING: FROM CONCEPT TO STRUCTURE.....	3
LEVELS OF ABSTRACTION IN DATA MODELING	3
1. <i>Conceptual Model</i>	5
2. <i>Logical Model</i>	6
3. <i>Physical Model</i>	7
WORKING WITH DDL SCRIPTS.....	9
AUTOMATING LOGICAL TO PHYSICAL (MODEL TRANSFORMATIONS).....	10
RELATING DATABASE SCHEMAS TO THE MODEL.....	11
REVERSE ENGINEERING: ‘EVOLVING’ A STRUCTURE.....	13
<i>Relating to Reverse-Engineered Code</i>	13
<i>Abstracting to Logical Level</i>	13
<i>Comparing New and Old Structures</i>	14
CONCLUSION.....	14
DBMS DATA TYPES:.....	15
DIAGRAM REPRESENTATIONS.....	15
DATA DICTIONARY.....	16
GLOSSARY.....	19

Introduction

Enterprise Architect supports comprehensive functionality for modeling database structures. This paper covers the core features for data modeling over the full lifecycle of an application. Initially, we discuss the basic modeling process – that is outlining a conceptual model and then working through the steps to form a concrete database schema. We will then look at re-engineering or ‘evolving’ an existing database schema for software version updates or porting to a new DBMS.

Database modeling can be performed using different notations. The notations Enterprise Architect supports include; a UML Profile for DDL, Entity Relationship Diagrams (ERD), IDEF1X and “Information Engineering”. For the purpose of this document we will focus on the UML profile for DDL, but include examples using the ERD notation.

Note: For a clearer overview on how to use the UML Profile for DDL modeling in Enterprise Architect, see the [Database Modeling in UML](#) paper.

Modeling: From Concept to Structure

Levels of Abstraction in Data modeling

Development of systems typically involves numerous levels of abstraction. These range from formal requirements modeling, Use Case modeling through to Class definition etc. Database modeling traditionally includes a well established three tiered approach:

1. *Conceptual Level* – this documents the basic entities of a proposed system and relationships between them
2. *Logical Level* – this specifies entities and their relationships without implementation details
3. *Physical Level* – this defines the database structure for a technology specific format (a DBMS)

These define the core stages in the design process of a database.

The models at each of the three levels of abstraction correspond to Model Driven Architecture (MDA) concepts. MDA's Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM) relate to the Conceptual, Logical and Physical models respectively. How you can use MDA transformations with data modeling and DDL generation are covered in more detail below.

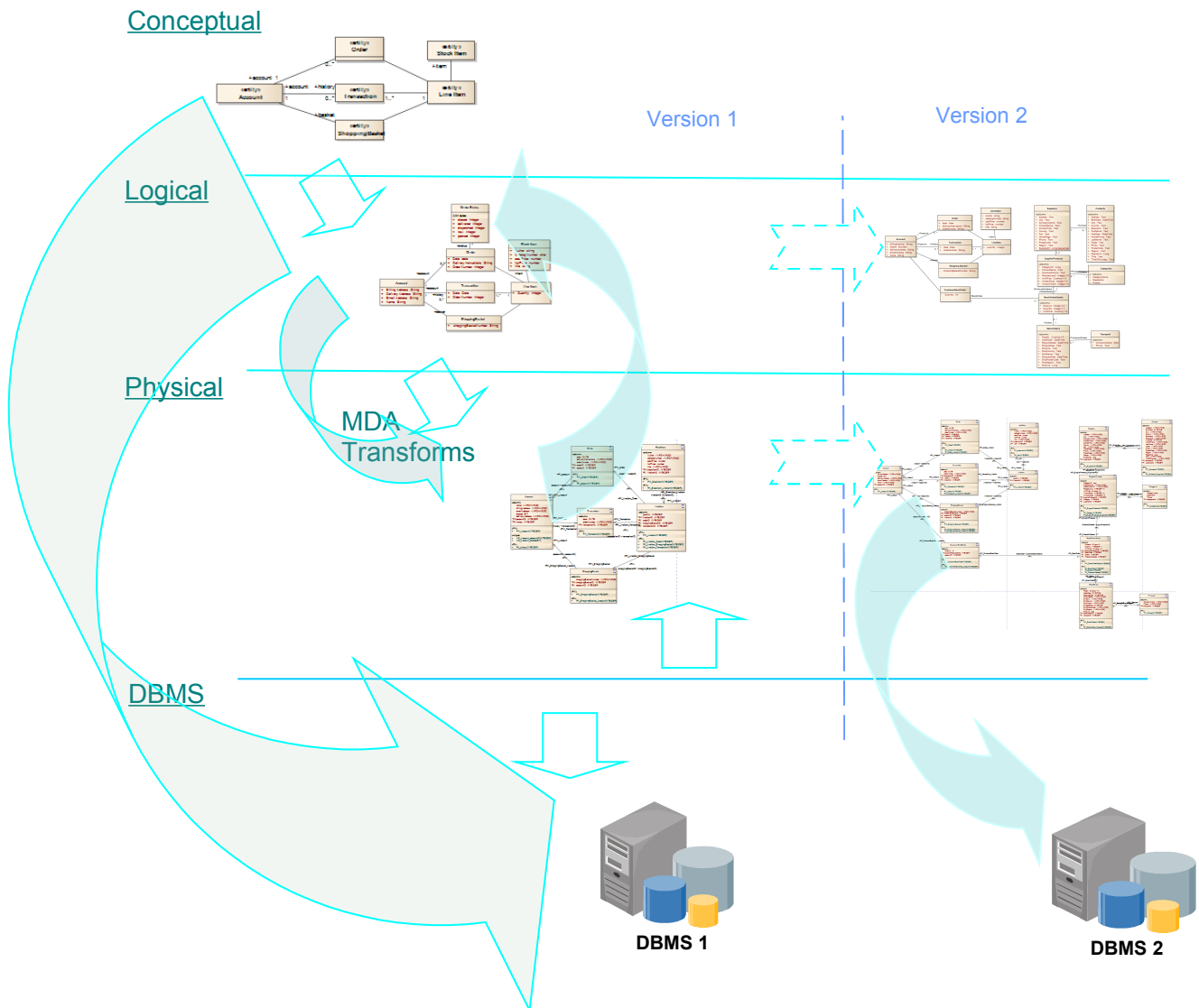


Figure 1: Flow through the levels of modeling a database.

Note: When working with a UML modeling tool there are strong parallels to defining the Class structure and the Data model. The UML concept of Classes with Attributes relates directly to Entities and their Attributes at the Conceptual level, and to Tables containing Fields on the Physical level.

Below is a summary of the data modeling aspects addressed at the Conceptual, Logical and Physical levels. The table also indicates which parts of the model can be derived by an MDA Transformation from the Logical model to Physical model.

	Conceptual	Logical	Physical	MDA Transform
Entities	✓			
Entity Relations	✓			
Attributes	Un-typed			
Class Names		✓		
Class Attributes		UML Typed		
Class Connectors		✓		
Table Names			✓	✓
Column Names			✓	✓
Column Types			DBMS Typed	✓
Primary Keys			✓	✓
Foreign Keys			✓	✓
Stored Procedures			✓	
Views			✓	
Triggers & Constraints			✓	

Figure 2: Conceptual, Logical and Physical modeling

1. Conceptual Model

The purpose of a Conceptual model is to simply establish the Entities, their Attributes and their 'high-level' relationships.

When modeling using UML, the Domain Model is used to define the initial structural layout (later to be used for Classes). Where the Class design is parallel to the data structure design, it is sensible to use the Domain model as a seed for the Conceptual model.

At the conceptual level there is little detail. The diagrams consist basically of Entities and their simple relationships. If there are Attributes defined, these are loosely typed (for example - no length settings), and connectors between Entities do not define relationships to specific Attributes.

Figure 3 below is an example of a simple Conceptual diagram for an Online Bookstore.

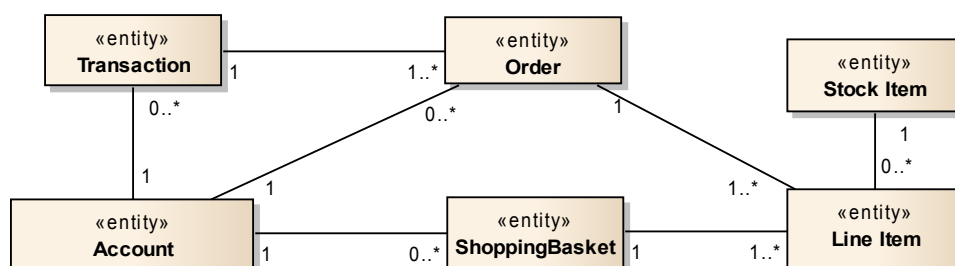


Figure 3: Conceptual Model Diagrams using UML to model the Online Bookstore

Enterprise Architect supports two different approaches for data models:

- UML Class diagram (DDL)
- Entity Relationship Diagram (ERD)

Using UML Class modeling, the Conceptual model consists of defining the data entities as an Element of type "Class". These Classes can later include internal Attributes, whereas with ERD based modeling Attributes are defined as related Elements (the Attributes are external to the Entities).

Figure 4 shows the Online Bookstore Conceptual model in ERD notation.

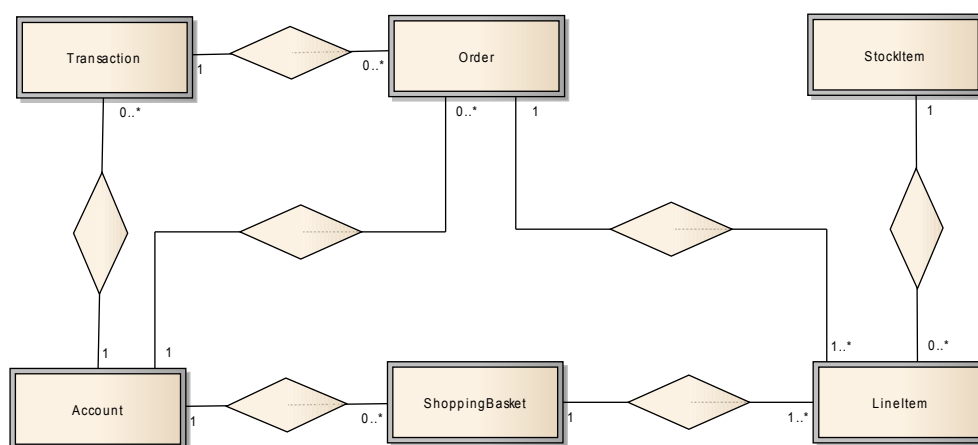


Figure 4: Conceptual model using ERD to model Online Bookstore

2. Logical Model

The Logical model includes more detail, specifically Attributes, but the modeling is still generic as it is not bound to a specific DBMS. The process of creating a Logical model based on a Conceptual model involves:

- **Setting the Attributes**
At the Logical level, the attributes (which later become Table Columns), are modeled independently of any DBMS product. They are typed using primitive UML data types, such as integer, boolean and string.").
- **Setting the Relationships**
At the Logical level we do not yet set the Primary Keys & Foreign Keys etc. At this level it is best to verify and adjust the Connector 'multiplicity' (also known as 'cardinality' in database terminology) details that were set earlier for relationships in the Conceptual model.

Figure 5 is a diagram of the Logical model derived from the Conceptual model in Figure 3.

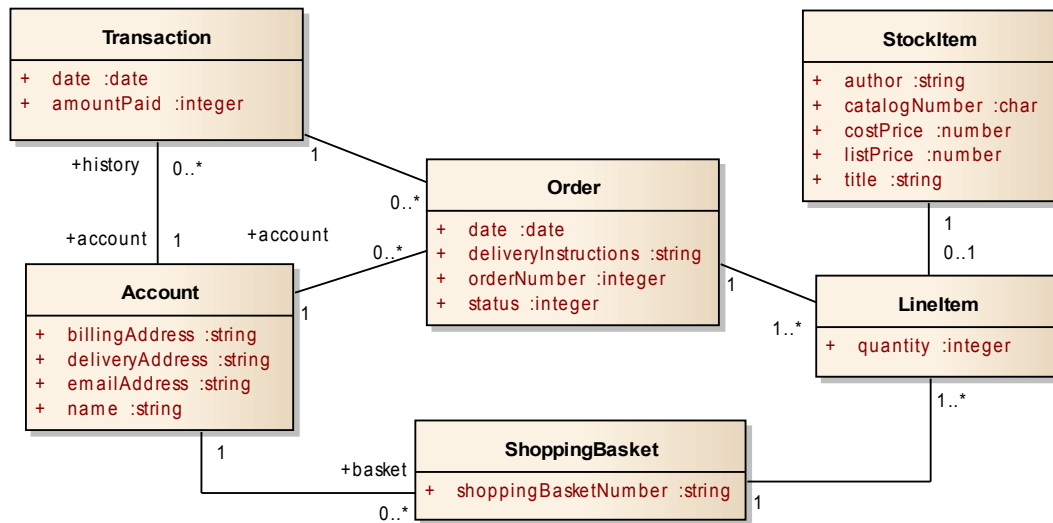


Figure 5: Logical model of the Online Bookstore using UML

Figure 6 shows an equivalent model of the Account and Transaction elements using an Entity Relationship Diagrams (ERD). The Entities do not contain attributes, rather the ‘table-columns’ are represented as ellipses connected to the Entity. The Entity relationships are represented as diamond-shape connectors.

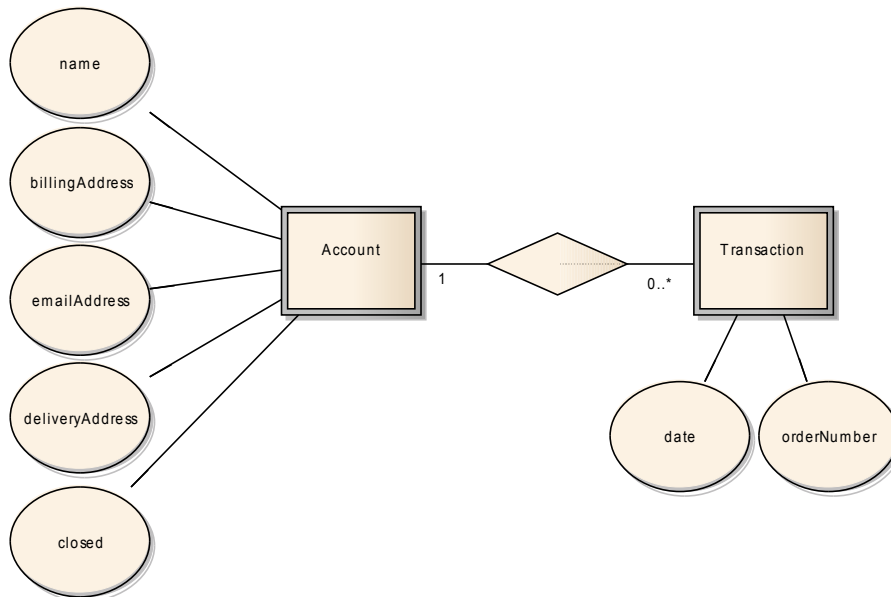


Figure 6: Logical diagram using ERD (for account & transaction)

3. Physical Model

Modeling on the Physical level involves adding “platform specific” detail to the model. That is, detail specific to the DBMS where the database is to be deployed. For more detail on manually setting the definition of the Physical database model using DDL diagramming see the paper on [Database Modeling in UML](#).

To set up the Physical model you can create a copy of the Logical Model and start the process of adding the Physical definitions to this model.

The key aspects of this are:

- For each 'Class':
 - o The Stereotype must be set to 'Table'.
 - o The Database setting must be set to a specific DBMS. For more information see: [Working with Tables](#).
 - o Update the Attributes to reflect Columns 'Typed' to the specific DBMS Field types. For more information see: [Create Columns](#).
- Add more detail to the Connectors (relationships), to define the Primary Key (& Foreign Key) linking. For more information see: [Database Key](#).

Note: All of the above Physical level additions and alterations can be automated using an MDA transformation from a Logical model. See 'Automating Logical to Physical (MDA transforms)' below.

Figure 7 shows the Physical model derived from the above Logical model for a specific DBMS. It uses the UML Profile for DDL modeling to represent MySQL specific details:

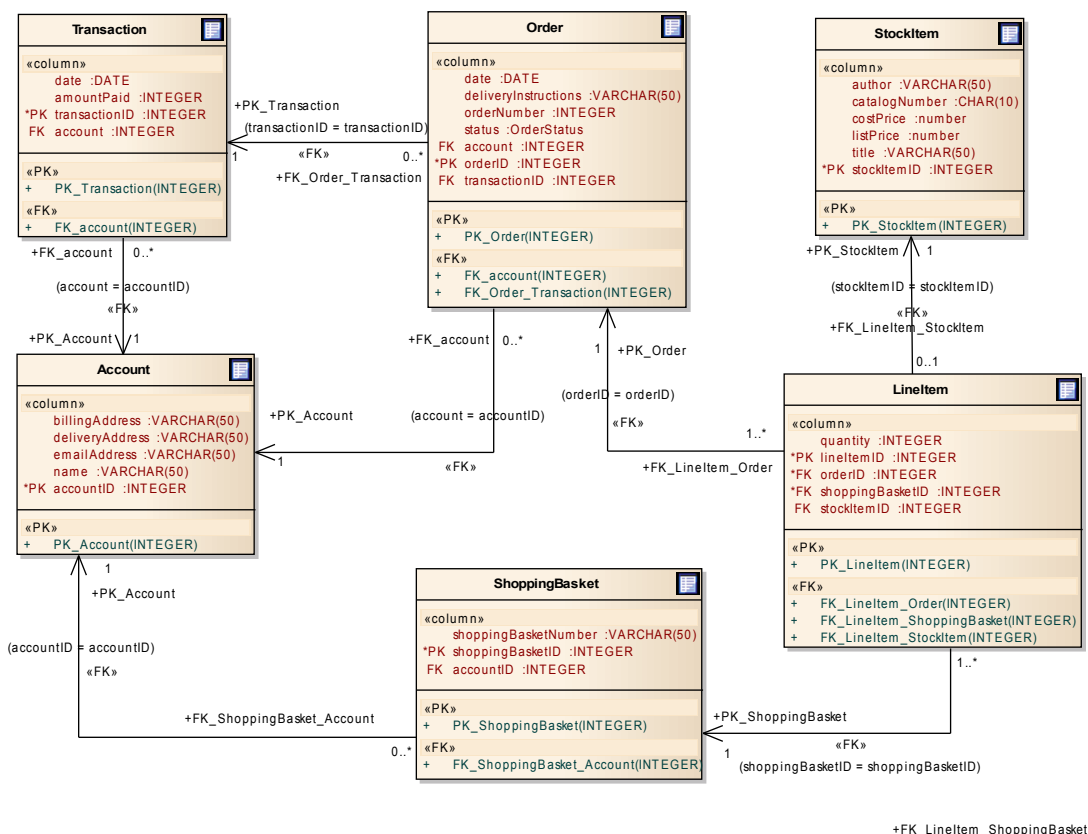


Figure 7: Physical level model set to a specific DBMS.

Further details can be added to the derived physical model. These include setting:

- Stored Procedures: see [Stored Procedures](#)
- Views: see - [Views](#)

- Constraints, Triggers: see – [Indexes](#), [Triggers](#) and [Check Constraints](#).

Note: These three features are outside the MDA transform covered in the section below: Automating Logical to Physical (Model Transformations).

Working with DDL Scripts

The final step is to generate the SQL script that will be ready to load into the DBMS. For more details on this see: [Generate DDL](#).

These scripts can also be imported back into Enterprise Architect and edited with SQL syntax highlighting in the Code Viewer.

The SQL script can be generated for the whole package or for single tables. If a single script for the package (a complete database) has been generated, the script file can be imported back to the Package. Otherwise for single table scripts, you can import these back to each Table Element.

To import the script, select either the package or the table-element, then in the **Properties** view – under **Filename** - import the generated script (see the Properties view, bottom right, in the image below).

To Edit or View the script, open the **Source Code** editor (**Alt+7**), then select the package or table-Element.

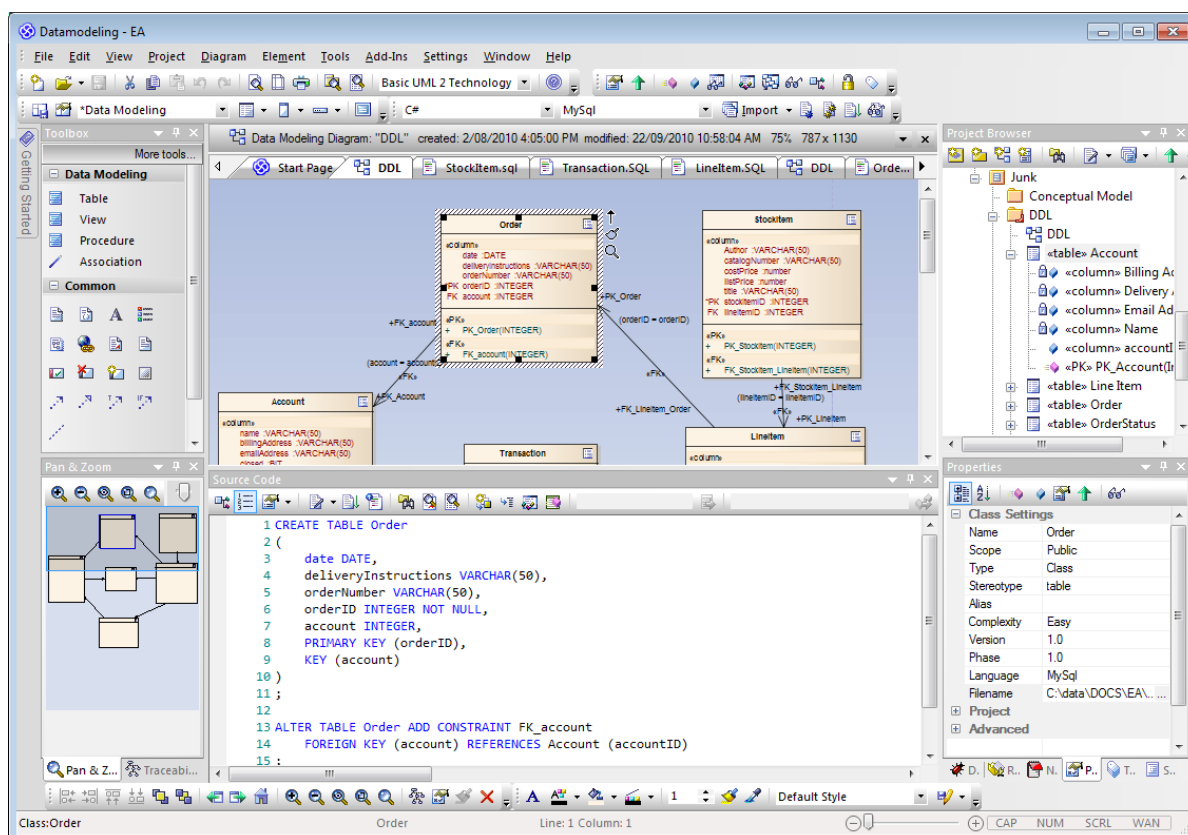


Figure 8: Database model with the DDL script for the table selected in the diagram

Automating Logical to Physical (Model Transformations)

Model Driven Architecture (MDA) transformations can automate the generation of lower level models from a higher level of abstraction.

For database modeling, MDA transformations take a DBMS independent Logical Data Model as input and generate a corresponding Physical model (DBMS specific).

Enterprise Architect supports the following MDA transformations for database modeling:

- **UML Classes to DBMS specific DDL tables.**
For more information, see the [DDL Transforms](#) help topic.
- **ERD Elements to DBMS specific DDL Tables.**
For more information, see the [ERD Transformation](#) help topic.

Note: the Enterprise Architect default DBMS needs to be set for the Logical Model to be transformed to a specific DBMS. To set the DBMS default, see help: [DBMS Datatypes](#)

There is also support for a reverse transform from DDL to ERD. For more details on this, see below: [Data Model to ERD Transformation](#)

A point to note is, that where there are ‘many-to-many’ relationships in the logical model, the DDL transformation generates a join-table to accommodate the relationship.

Returning to the Online Bookstore example (see Figure 5), we could re-model "StockItem" as two separate tables, Publication and Author, related using a many-to-many UML Association. This is shown as:

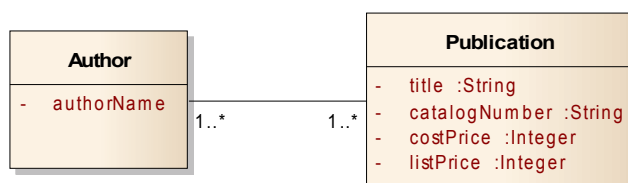


Figure 9: ‘Stockitem’ redefined in the logical model as Author and Publication with a many-to-many relationship.

Applying the MDA DDL transformation to this logical model returns:

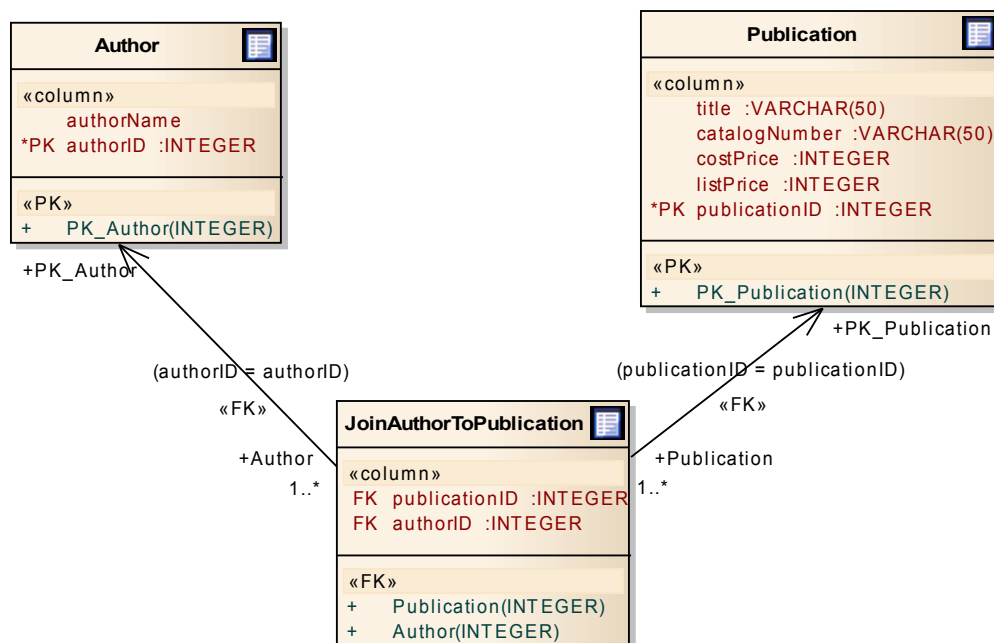


Figure 10: MDA Transformation of a many-to-many logical relation.

Relating Database Schemas to the Model

When modeling on different levels of abstraction, there is often a need for traceability between the equivalent entities across each level of abstraction (Conceptual, Logical and Physical), as well as, traceability between Table Fields and their associated application logic (Class Attributes) and User Interface diagrams.

There are a number of methods that can be used for this process:

1. When relating elements across levels of abstraction Enterprise Architect's [Relationship Matrix](#) can be used to create or view links between entities, on their different levels.
 The [Traceability View](#) can also be used to view relationships between entities/elements from a hierarchical perspective.
2. Table-fields can be directly related to Class Attributes using [Connect to Element Feature](#). Below is a simple example of this type of mapping in a diagram.

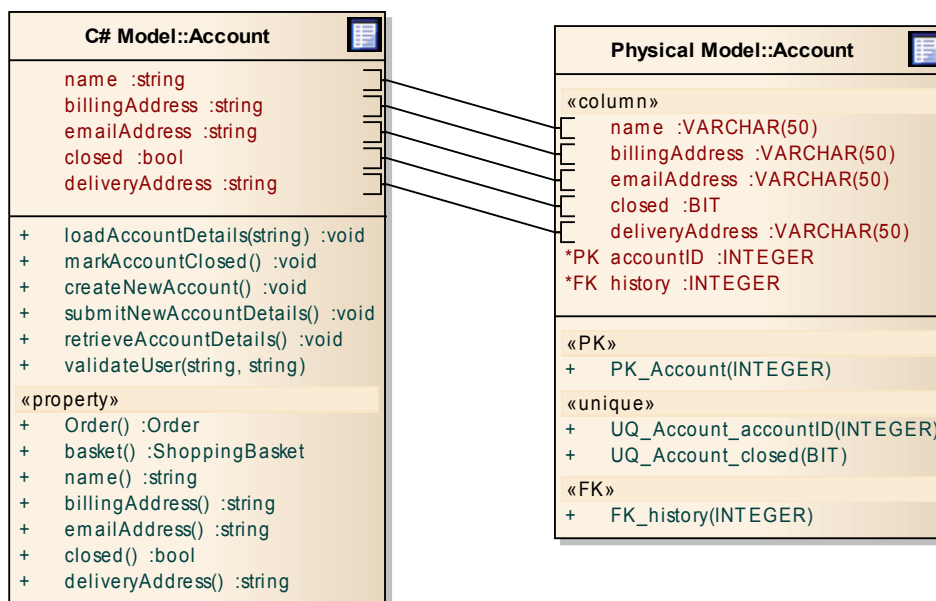


Figure 11: Directly linking Class Attributes to Table Fields

- Some database systems relate explicit Screen entry Fields direct to Table Fields (.e.g. MS Access, PowerBuilder etc.). To model these relationships a Screen Model can be linked using the [Connect to Element Feature](#) connections as shown below.

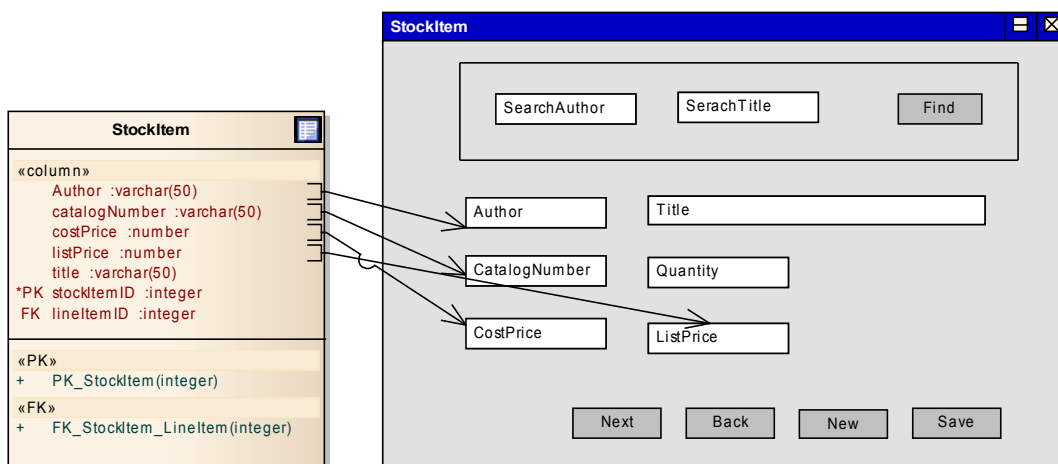


Figure 12: Associating table fields to screen controls

Reverse Engineering: 'Evolving' a Structure

A major version change of a system can involve core structural modifications. These could include changes resulting from different database structures, through to a different data storage solution (another DBMS).

When modeling changes to a database structure, you must view the current schema in a model. Where the model repository has not been maintained over the lifecycle of system changes, you need to re-import the schema back into Enterprise Architect. For details on the process to Reverse Engineer a DBMS see [Import Database Schema](#).

Once the legacy schema is available in the model it can then be manipulated in a number of ways, such as simply converting it to another DBMS format, relating it to reverse-engineered application classes or further abstracting it to form a Platform Independent model.

The options available are:

Convert to another DBMS (Migration)

In cases where the primary change is from one DBMS to another DBMS, a simple process that Enterprise Architect supports is [Data Type Conversion for a Package](#). This supports a direct conversion from one DBMS to another (e.g. MS SQL Server to Oracle).

Post conversion, the structure can be modified to meet any alterations to the design. The updated data model can then be used to generate the schema as a DDL script and then loaded into the new DBMS to create a new database. See: [Generate DDL](#).

Relating to Reverse-Engineered Code

For an overview of the options for relating the reverse engineered database structure to the reverse engineered code see: [Relating Data schemas to the Model](#) outlined above.

Abstracting to Logical Level

Reverse Engineering the database schema will create a Physical DDL model. Where there needs to be more work on the design, and in cases where the code has also been reverse engineered, and the Class structures of the application model are being altered, there may be a need to work on a data modeling at a higher level of abstraction - the Logical model.

Where the original data model used to develop the legacy version is still available, then the Logical data model can be re-used as a starter and updated. Although an MDA transformation from the Physical data model to the Logical data model is not supported, if the code structure is a close reflection of the data schema, then a logical model can be derived by reverse engineering the code and then this Class diagram can be used as a foundation for defining the Database logical structure.

Comparing New and Old Structures

A function that can be of great use when reverse engineering a database schema, then modifying the model, is to compare the new modeled structure against the existing database schema stored on the DBMS. For more details on this see 'Compare DDL for a Database' in [Generate DDL for a Package](#).

Conclusion

This covers some of the key features of Enterprise Architect's support for Data Modeling including modeling from the Conceptual to Physical levels, Forward and Reverse Engineering of Database Schemas, and MDA transformation of the Logical model (platform independent) to Physical DBMS (platform dependant schema). When modeling an application and the database in the same environment there is added benefit of traceability between the application modeling and the data modeling; synchronization between the design of the application, the design of the data model and the final implementation of these, along with, the added benefits of model reuse when starting new designs.

Whether you need to model and manage complex designs and visualize data structures, or build and deploy diverse data systems, Enterprise Architect provides interconnected modeling, development and deployment, for both the database and the system code.

Appendix:

DBMS data types:

There may be cases where you want to create a physical data model for a DBMS product that is not yet supported natively by Enterprise Architect, or you may need to add datatypes for a more recent version of a supported DBMS. The Database Datatypes screen allows adding a new Database product, or customizing the data-types associated with an existing one.

To do this, select from the main menu: **Settings | Database Datatypes**. See the related help topic for more details.

Diagram Representations

Enterprise Architect supports a number of notations relevant to data modeling. This document primarily uses Class diagrams. Other supported notations include:

- IDEF1X
- ERD
- Information Engineering

The following are examples of the IDEF1X and Information Engineering formats:

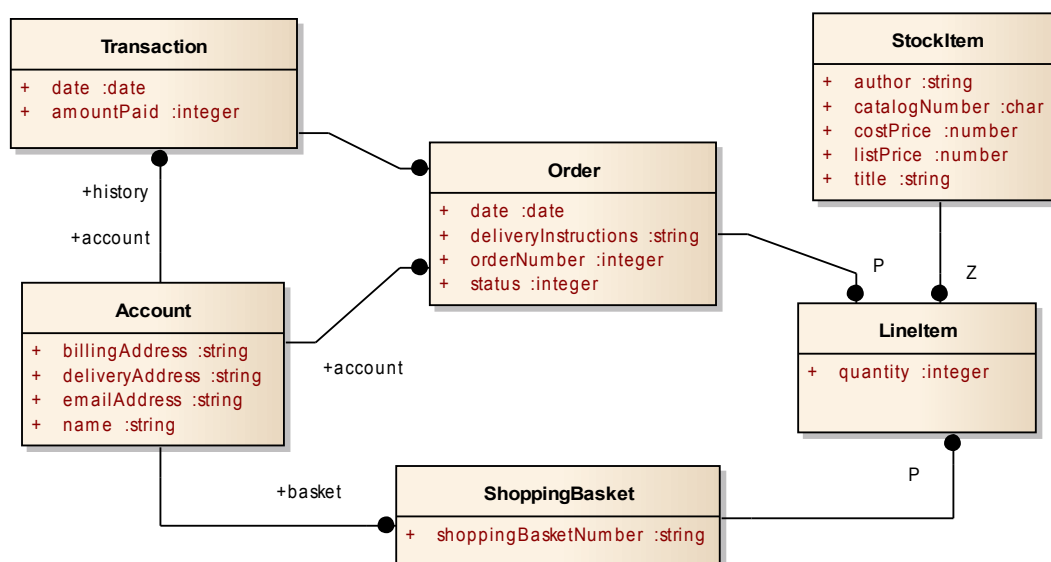


Figure 13: IDEF1X representation of the Online Bookstore data model

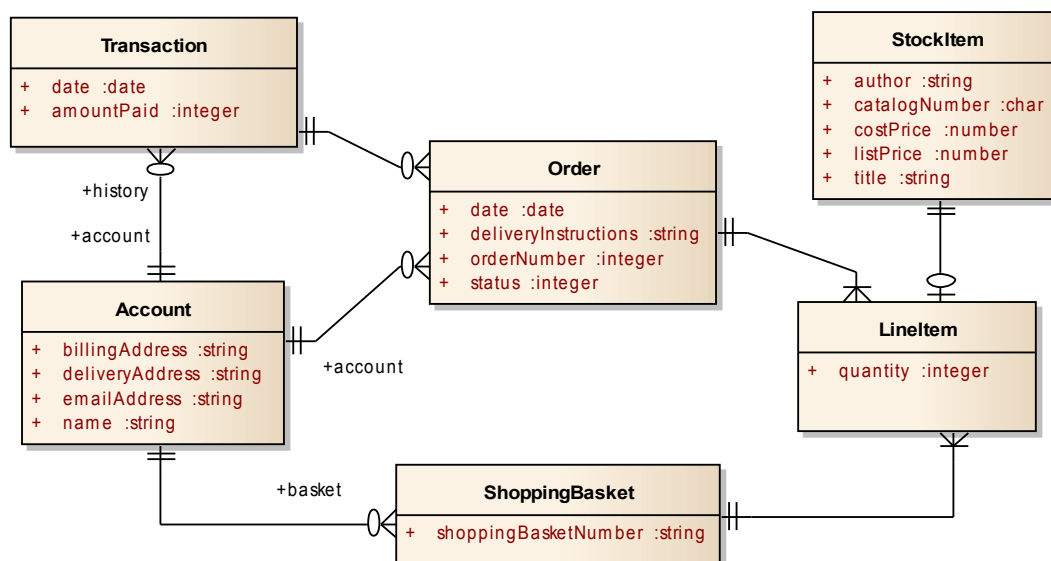


Figure 14: Information Engineering representation of the Online Bookstore data model


To change the format of the diagram between UML Profile for DDL, IDEF1X and Information Engineering, select from the main menu: **Diagram | Properties > Connectors**, then using the dropdown: **Connector Notation** – select the appropriate format.

Data Dictionary

A simple method to create a data dictionary is to use the Enterprise Architect [Model Search](#) facility to create a ‘Data Dictionary’ query. This returns results that can be sorted by many different fields (e.g. by Table name, Field Type, Primary Key, Foreign key etc.).

Enterprise Architect supports storing its model repositories using a number of different DBMS. As the search query format differs between each DBMS, only the simple default .eap SQL script is shown below. Note this is simple to re-arrange to suite the SQL specific to the DBMS of your repository.

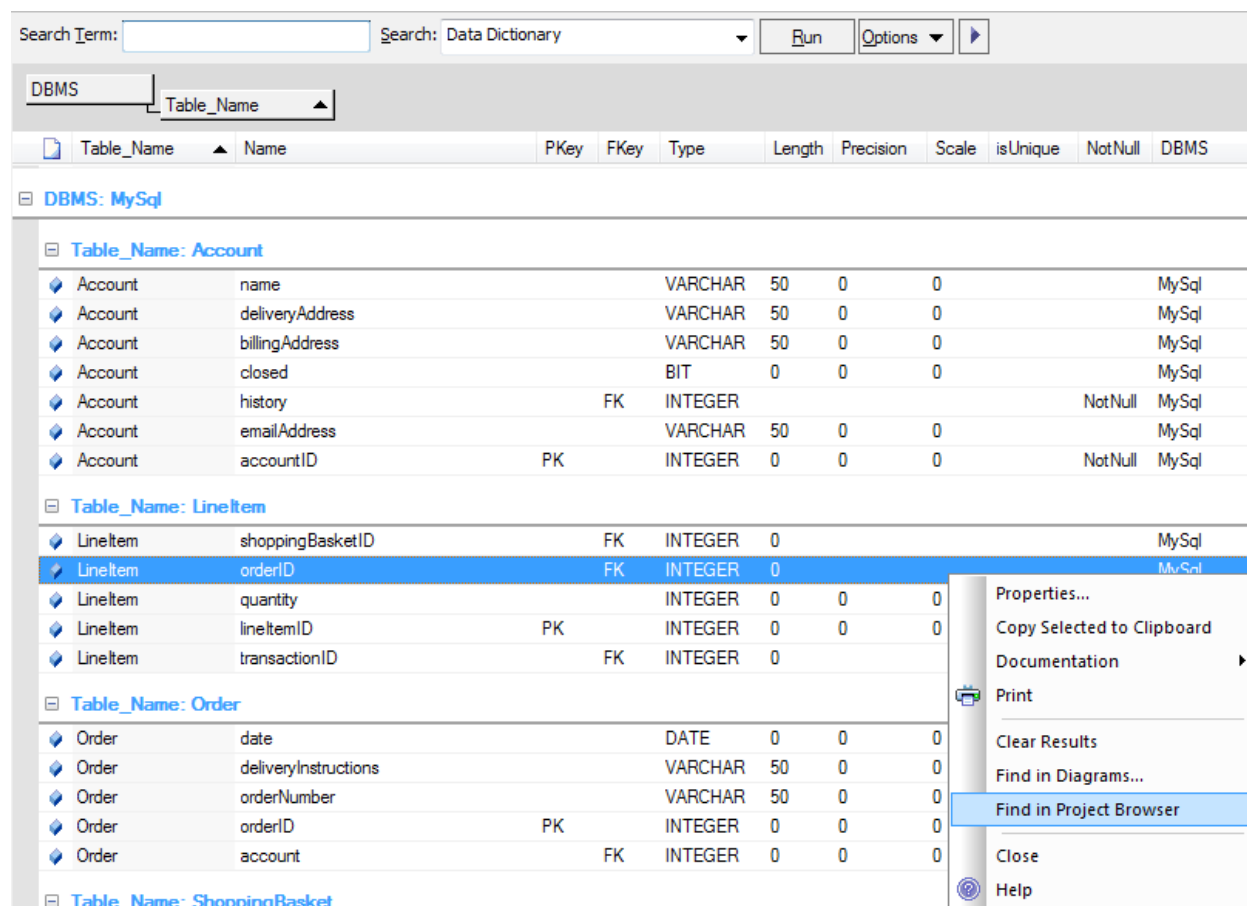
To create the Model Search query:

1. Use **Ctrl+F** to open the [Model Search](#) view
2. Then select: [Options | Manage Searches](#)
3. Select the New Search icon 
4. On the [Create New Search Query](#) dialog:
 - Type in a Name (“Data Dictionary EAP”)
 - Select: [SQL Editor](#)

The SQL search string to use is:


```
SELECT t_attribute.ea_guid AS CLASSGUID, 'Attribute' AS CLASSTYPE,
t_object.Name as Table_Name,
t_attribute.Name,
iif(t_attribute.IsOrdered, "PK", "") as PrimaryKey,
iif(t_attribute.IsCollection, "FK", "") as ForeignKey ,
t_attribute.Type, t_attribute.Length, t_attribute.Precision, t_attribute.Scale,
iif(t_attribute.IsStatic, "Unique", "") as isUnique,
iif(t_attribute.AllowDuplicates, "NotNull", "") as NotNull
FROM t_attribute, t_object
WHERE t_attribute.object_Id = t_object.Object_ID and t_object.Stereotype = "Table"
```

This will support cross-referencing to the Table Fields in the [Project Browser](#) using the [Context menu](#) options in the result set generated (see below):



Glossary

Attribute	A feature within an Entity that describes where a range of values can be stored.
Conceptual Level	The initial design level used to establish the Entities and simple relationships.
Conceptual Model	Models defined at the Conceptual Level.
Database Schema	The structure of a database described in a formal language supported by the database management system (DBMS).
DBMS	Database Management System, is a software system that manages databases on a server.
DDL	A Data Definition Language is a language used for defining data structures.
Entity	A uniquely identifiable object abstracted from the complexities of some domain – fundamental element used in ERD.
ERD	Entity-Relationship Diagrams: a classical data modeling notation, primarily used for conceptual modeling.
IDEF1X	“Integration Definition for Information Modeling” is a data modeling language associated with the IDEF group of modeling languages.
Information Engineering	“Information Engineering” is a database modeling notation for Entity-Relationship Modeling.
Logical Level	This level of abstraction is a simple description of database structure, and what relationships exist among those data entities.
Logical Model	Data Models defined at the logical Level.
MDA	Model Driven Architecture (MDA) is a software design approach for the development of software systems. See OMG - MDA
Physical Level	The level where the database schema modeling has sufficient detail to facilitate implementation on a specific DBMS.
Physical Model	Models defined at the Physical Level.
PIM	Platform Independent Model – used in MDA transformations.
PSM	Platform Specific Model –generated by MDA transformations.
UML	Unified Modeling Language: an object modeling and specification language used in application design. See OMG-UML.