ENTERPRISE ARCHITECT

User Guide Series

Information Engineering

Author: Sparx Systems
Date: 2026-05-04
Version: 17.1

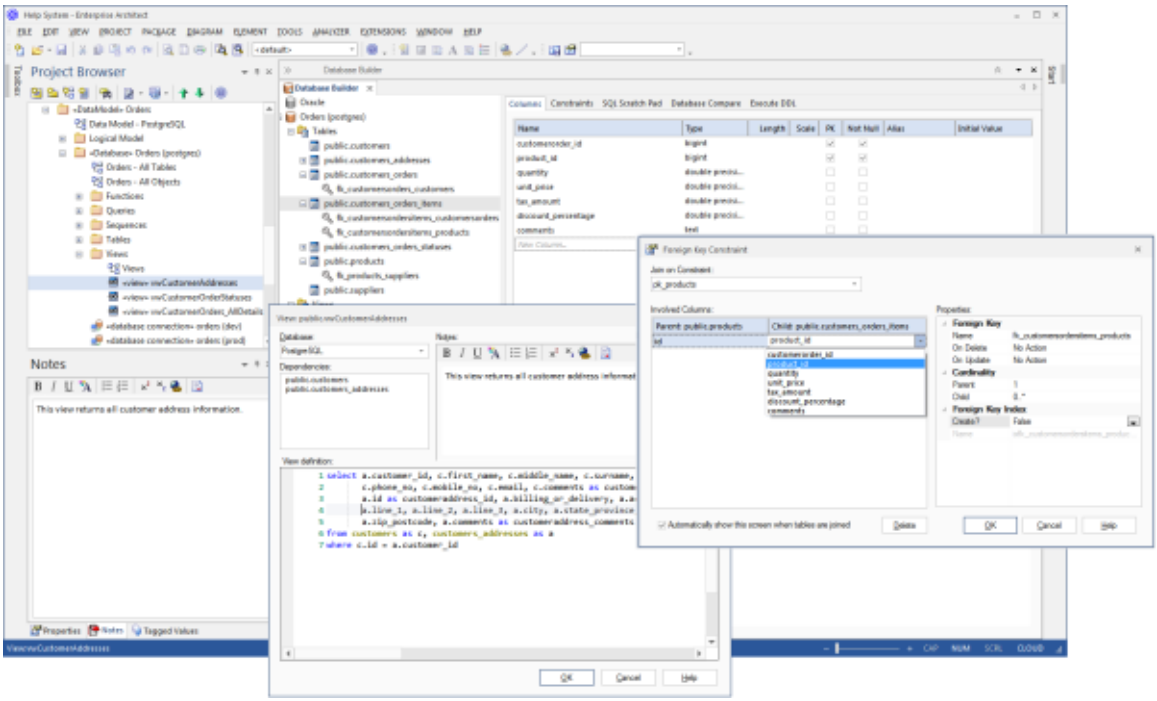
CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Information Engineering

Design, Create and Manage Conceptual, Logical and Physical Data Models

The power of model-based system development is the ability to visualize, analyze and design all aspects of a system. Being able to view and manage information and data alongside other models of a system provides great clarity and reduces the chance of error. Enterprise Architect has extensive support for the data modeling discipline, ranging from the representation of information in a conceptual model right down to the generation of database objects. Whether you are generating database objects from the UML model or reverse engineering legacy DBMS into a model for analysis, the tool features will save time and valuable project resources.



This illustration shows the Database Builder Interface including DDL Generation and the Foreign Key dialog.

Enterprise Architect supports the modeling of information at the conceptual, logical and physical layers. Using a number of standard features, these models can be interconnected, providing traceability. The logical and physical models can also be generated automatically using a fully customizable Transformation engine. Legacy systems can be imported, analyzed and compared using the handy reverse engineering facility.

In this topic you will learn how to use the feature rich toolset including the Database Builder to design, create, manage, visualize data including reverse and forward engineering of data models to live database.

The Database Builder tool can be used to create and maintain physical data models and can connect to a running DBMS, so you can therefore import, generate, compare and alter a live database.

Getting Started

Information Modelers, Data Modelers and Architects are responsible for creating models of an organization's information that span multiple levels of abstraction, from conceptual through to logical and physical. The conceptual models are technology independent and can be used for discussions with business people and domain experts, allowing the basic concepts in the domain to be represented, discussed and agreed upon. The logical model elaborates the conceptual model, adding more detail and precision but is still typically technology neutral, allowing Information Analysts to discuss and agree on logical structures. The physical model applies technology specific data to the models and allows engineers to discuss and agree on technology decisions in preparation for generation to a target environment, such as a database management system.

Selecting the Perspective

Enterprise Architect partitions the tool's extensive features into Perspectives, which ensures that you can focus on a specific task and work with the tools you need without the distraction of other features. To work with the Data Modeling features you first need to select one of these Perspectives:


<perspective name> > Database Engineering > Database Engineering


<perspective name> > Database Engineering > Entity Relationships

Setting the Perspective ensures that the Database Engineering diagrams, their tool boxes and other features of the Perspective will be available by default.

Example Diagram

An example diagram provides a visual introduction to the topic and allows you to see some of the important elements and connectors that are created in specifying or describing the way a data model is defined including: Tables, Views, Procedures, Sequences, Functions.

Data Model Types

Information can be modeled at a number of level of abstraction starting with a conceptual model that is typically created by or for business people, a Logical model which is used by business and systems analysts and a physical model which is the concern of the technologists such as database engineers. In this topic you will learn how to manage all three levels of information models.

Creating and Managing Data Models

In this topic you will learn how to work in detail using Enterprise Architect to manage your Physical Database schema. This includes the use of the Database Builder tool which allows you to interact with any number of live database through an ODBC connection.

Import Database Schema

This topic will show you how to connect to a live database including Production, Test and Development systems and reverse engineer the database into a model creating Tables, Views, Procedures, Declarative Referential Integrity and

more. A diagram of the database is automatically created and the elements such as tables can be related to other elements in the model including Conceptual and Logical Models, Programming classes tests and more.

Generate Database Definition Language (DDL)

In this topic you will learn how to harness the power of the data models by generating Database Definition Language code directly from the model. Enterprise Architect can generate code into a wide range of Database Management systems.

Supported Database Management Systems

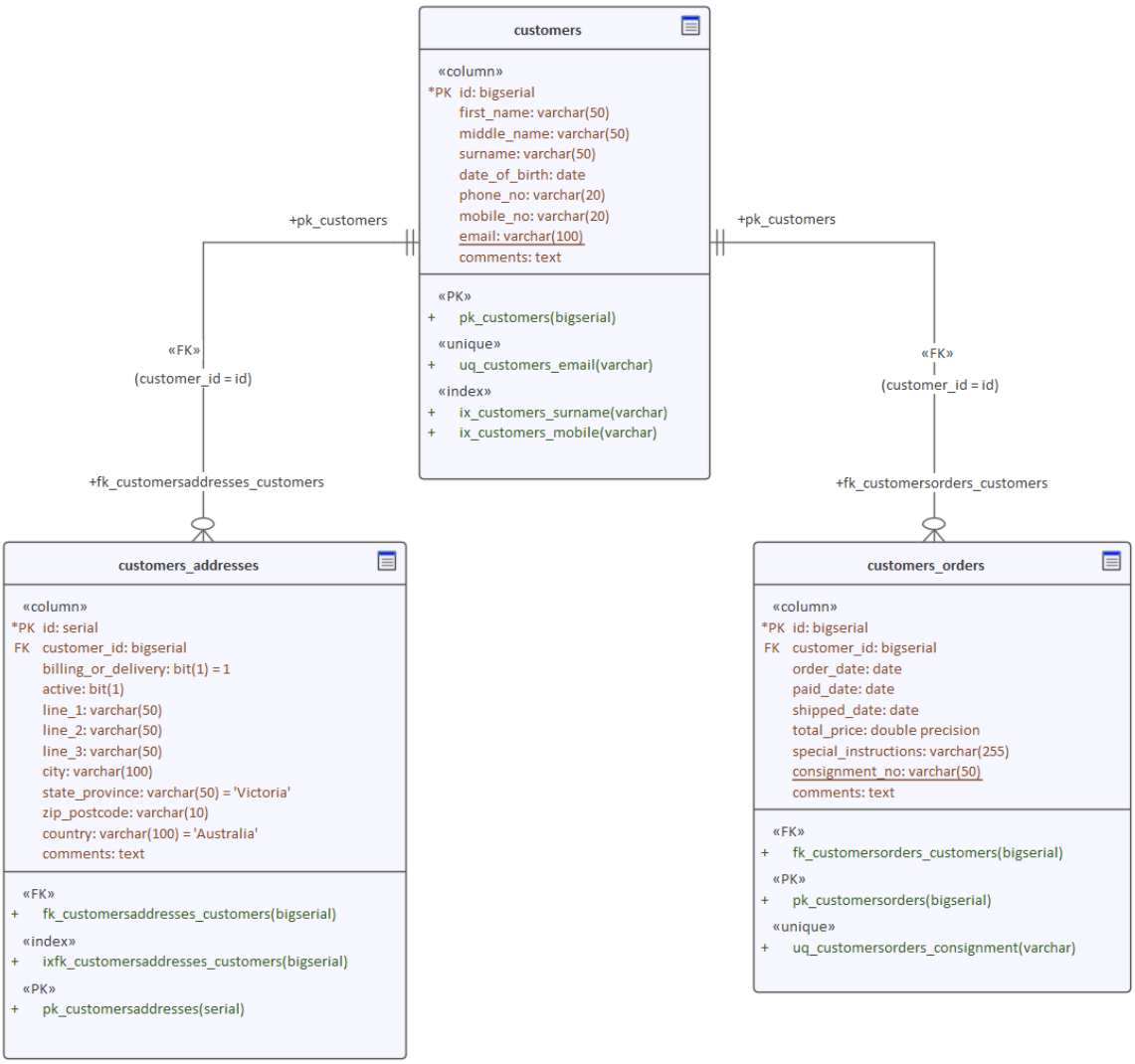
Enterprise Architect has rich support for most of the main stream Database Management Systems (DBMS). This feature allows models from disparate systems to be compared either for code generation or for analysis by using the import feature. This topic lists the supported DBMS and

More Information

This section provides useful links to other topics and resources that you might find useful when working with the Data Modeling tool features.

Example Diagram

Using the Database engineering features of Enterprise Architect you can create rich models of the objects that make up a data model at any level of abstraction from Conceptual through Logical to Physical. These models are created by adding tables and other database objects from the toolbox or by reverse engineering and existing database into a model from a range of RDBMSs. A database diagram can contain Tables, VIews, Procedures, Sequences and Functions. Table Columns are annotated as Primary and Foreign Keys are modeled using specialized association relationships. In this example the user has created a simple physical data model of Customers and their Addresses and Orders.



Physical Data model showing Tables with Columns and Primary and Foreign Keys.

Working with Data Model Types

Enterprise Architect provides a number of features to assist in the process of creating models of information, including the ability to develop conceptual, logical and physical models and to be able to trace the underlying concepts between the models. The physical models can be developed for a wide range of database systems, and forward and reverse engineering allows these models to be synchronized with live databases.

Data Models

Type	Description
Conceptual Data Models	Conceptual data models, also called Domain models, establish the basic concepts and semantics of a given domain and help to communicate these to a wide audience of stakeholders. Conceptual models also serve as a common vocabulary during the analysis stages of a project; they can be created in Enterprise Architect using Entity-Relationship or UML Class models.
Logical Data Models	Logical data models add further detail to conceptual model elements and refine the structure of the domain; they can be defined using Entity-Relationship or UML Class models. One benefit of a Logical data model is that it provides a foundation on which to base the Physical model and subsequent database implementation. Entity-relationship modeling is an abstract and conceptual database modeling method, used to produce a schema or semantic data model of, for example, a relational database and its requirements, visualized in Entity-Relationship Diagrams (ERDs). ERDs assist you in building conceptual data models through to generating Data Definition Language (DDL) for the target DBMS. A Logical model can be transformed to a Physical data model using a DDL Transformation.
Physical Data Models	Physical data models in Enterprise Architect help you visualize your database structure and automatically derive the corresponding database schema; you use Enterprise Architect's UML Profile for Data Modeling specifically for this purpose. The profile provides useful extensions of the UML standard that map database concepts of Tables and relationships onto the UML concepts of Classes and Associations; you can also model database columns, keys, constraints, indexes, triggers, referential integrity and other relational database features. Because Enterprise Architect helps you visualize each type of data model in the same repository, you can easily manage dependencies between each level of abstraction to maximize traceability and verify completeness of system implementation.

Conceptual Data Model

A Conceptual data model is the most abstract form of data model. It is helpful for communicating ideas to a wide range of stakeholders because of its simplicity. Therefore platform-specific information, such as data types, indexes and keys, is omitted from a Conceptual data model. Other implementation details, such as procedures and interface definitions, are also excluded.

This is an example of a Conceptual data model, rendered using two of the notations supported by Enterprise Architect.

Entity Relationship diagram showing a One-to-Many relationship

Using Entity-Relationship (ER) notation, we represent the data concepts 'Customers' and 'Customers Addresses' as Entities with a one-to-many relationship between them. We can represent exactly the same semantic information using UML Classes and Associations.

Unified Modeling Language diagram showing the same One-to-many Relationship

Whether you use UML or ER notation to represent data concepts in your project depends on the experience and preferences of the stakeholders involved. The detailed structure of the data concepts illustrated in a Conceptual data model is defined by the Logical data model.

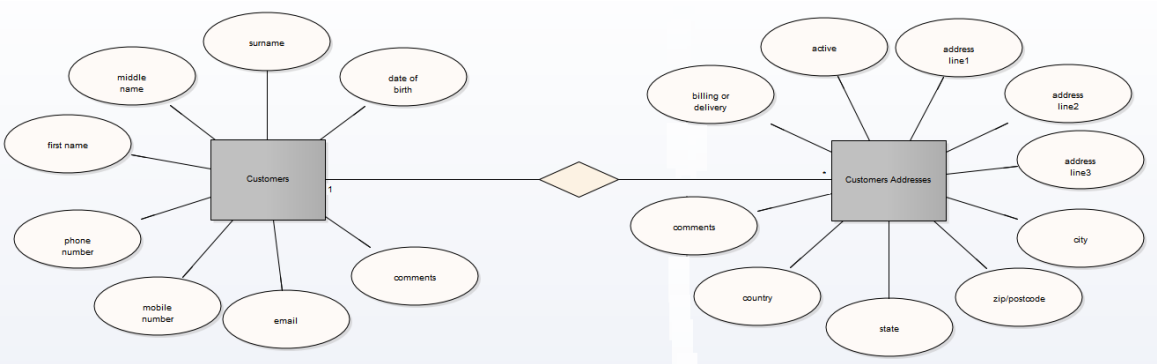
Entity Relationship Diagrams (ERDs)

According to the online Wikipedia:

An entity-relationship model (ERM) is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often a relational database, and its requirements in a top-down fashion. Diagrams created by this process are called Entity-Relationship Diagrams, ER Diagrams, or ERDs.

Entity Relationship Diagrams in Enterprise Architect

Entity Relationship diagrams in Enterprise Architect are based on Chen's ERD building blocks: entities (tables) are represented as rectangles, attributes (columns) are represented as ellipses (joined to their entity) and relationships between the entities are represented as diamond-shape connectors.



ERD technology in Enterprise Architect assists you in every stage from building conceptual data models to generating Data Definition Language (DDL) for the target DBMS.

ERD and ERD Transformations

Enterprise Architect enables you to develop Entity Relationship diagrams quickly and simply, through use of an MDG Technology integrated with the Enterprise Architect installer.

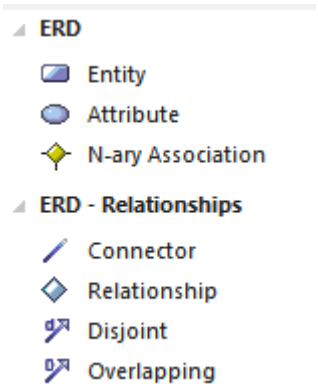
The Entity Relationship diagram facilities are provided in the form of:

- An Entity Relationship diagram type, accessed through the 'New Diagram' dialog
- An Entity Relationship Diagram page in the Diagram Toolbox
- Entity Relationship element and relationship entries in the 'Toolbox Shortcut' menu and Quick Linker

Enterprise Architect also provides transformation templates to transform Entity Relationship diagrams into Data Modeling diagrams, and vice versa.

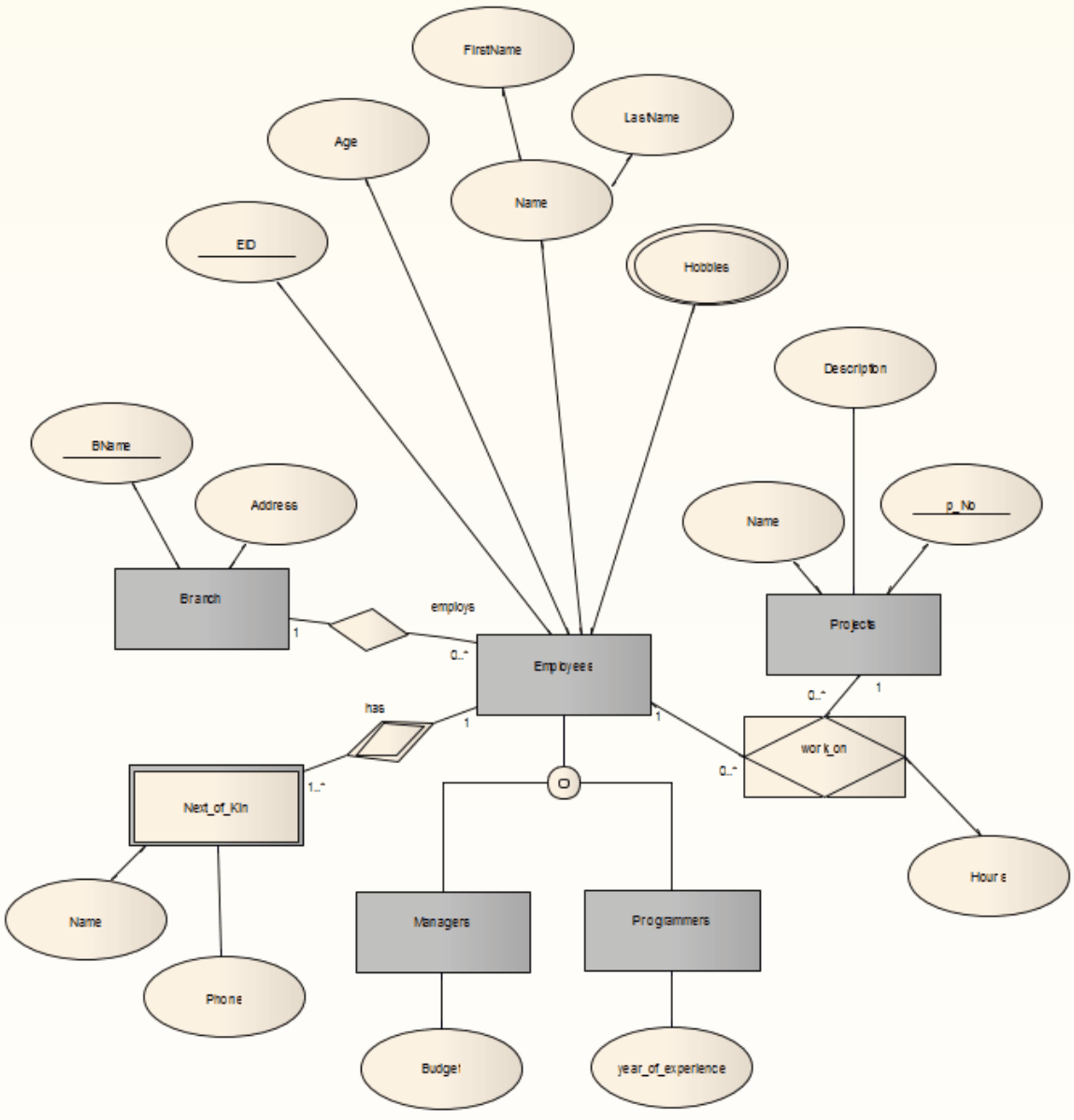
Entity Relationship Diagram Toolbox Page

You can access the 'Entity Relationship Diagram' page of the Diagram Toolbox by specifying 'Entity Relationship Diagrams' in the Toolbox 'Find Toolbox Item' dialog



- Entity is an object or concept that is uniquely identifiable; the property of 'Multiplicity' in the SourceRole and TargetRole definitions for the Relationship connector can be used to define the cardinality of an Entity that participates in this relationship
- Attribute is a property of an entity or a relationship type
- N-ary Association represents unary (many-to-many recursive) or ternary relationships and can also be used to represent relationships that have attributes among the entities; the N-ary Association element should always be at the target end of a connector
- Connector is a connector between an Entity and an Attribute, and between two Attributes
- Relationship is a diamond-shape connector, representing the meaningful association among entities
- Disjoint and Overlapping represent the relationships between the super-class Entity and the sub-class Entity

A typical Entity Relationship diagram



Tagged Values

Some of the Entity Relationship diagram components can be modified by Tagged Values, as indicated:

Component	Tagged Value / Notes
Entity	isWeakEntity Notes: If true, this entity is a weak entity.
Attribute	attributeType Notes: There are four valid options: 'normal', 'primary key', 'multi-valued' and 'derived'
Attribute	commonDataType Notes: Defines the common data type for each attribute.

Attribute	dbmsDataType Notes: Defines the customized DBMS data type for each attribute. This option is only available when the *commonDataType* tag is set to 'na'. You must define the customized type first through the 'Settings > Reference Data > Settings > Database Datatypes' ribbon option.
N-ary Association	isRecursive Notes: If true, the N-ary Association represents the many-to-many recursive relationship. For one-to-many and one-to-one recursive relationships, we suggest using the normal Relationship connector. Sometimes you might want to limit the stretch of the diamond-shape Relationship connectors; simply pick a Relationship connector, right-click to display the context menu, and select the 'Bend Line at Cursor' option.
Relationship	isWeak Notes: If true, the Relationship is a weak relationship.
Disjoin Overlapping	Participation Notes: There are two valid options, 'partial' and 'total'.

Notes

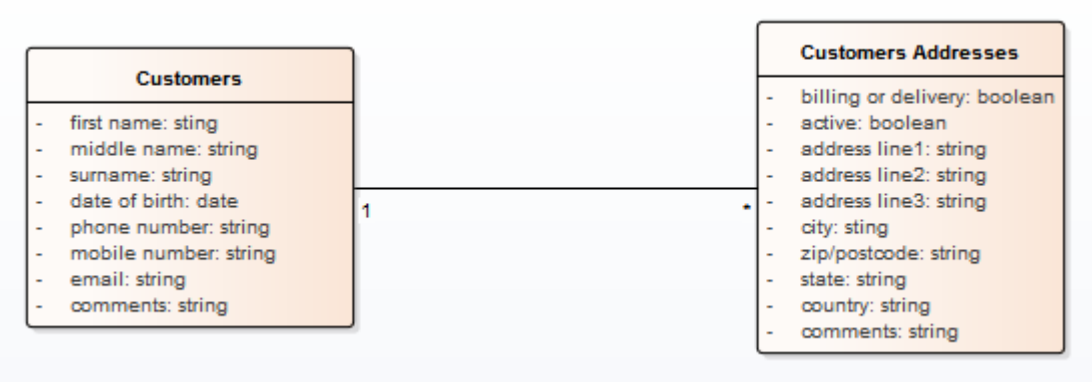
- Entity Relationship diagrams are supported in the Corporate, Unified and Ultimate Editions of Enterprise Architect

Logical Data Model

Logical data models help to define the detailed structure of the data elements in a system and the relationships between data elements. They refine the data elements introduced by a Conceptual data model and form the basis of the Physical data model. In Enterprise Architect, a Logical data model is typically represented using the UML Class notation.

Example

This diagram is a simple example of a Logical data model. The Logical Model adds detail to the Conceptual Model but without going to the level of specifying the Database Management System that will be used.



Conceptual Data Model with tables modeling Customers and their Addresses.

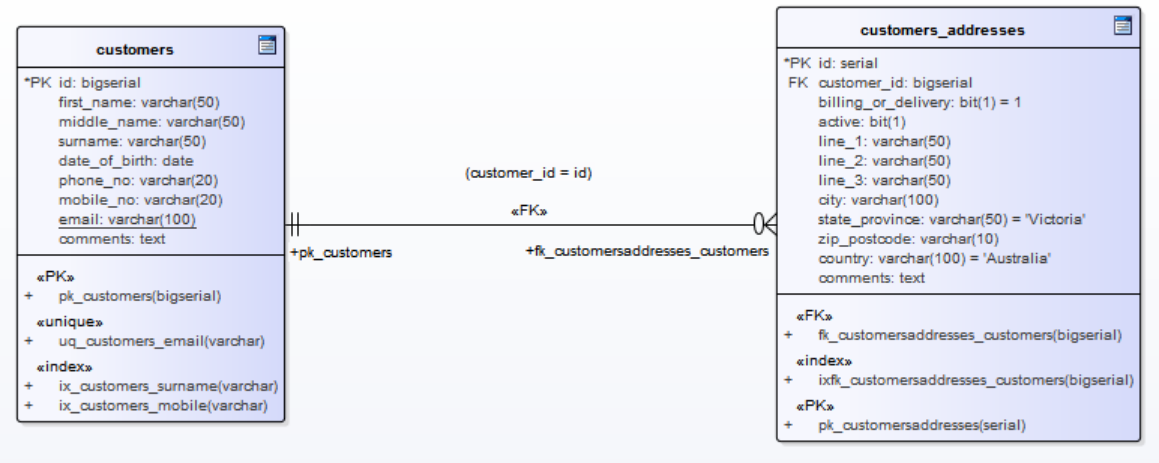
Note that the data elements 'Customers' and 'Customers Addresses' contain UML attributes; the names and generic data types to remain platform-independent. Platform-specific data types and other metadata that relate to a specific DBMS implementation are defined by the Physical data model.

Physical Data Models

A Physical Data Model visually represents the structure of data as implemented by a relational database schema. In addition to providing a visual abstraction of the database structure, an important benefit of defining a Physical Data Model is that you can automatically derive the database schema from the model. This is possible due to the richness of metadata captured by a Physical Data Model and its close mapping to aspects of the database schema, such as database Tables, columns, Primary Keys and Foreign Keys.

Example Data Model

This example shows a Physical Data Model that could be used to automatically generate a database schema. Each Table is represented by a UML Class; Table columns, Primary Keys and Foreign Keys are modeled using UML attributes and operations. This model demonstrates the use of the Information Engineering connector style.



Notation

The example model is defined using Enterprise Architect's UML Profile for Data Modeling; the relationship between the Tables uses the default Information Engineering notation.

Information Engineering is one of three notations that Enterprise Architect supports to help Data Modelers identify cardinality in relationships. You can change the notation by selecting the 'Design > Diagram > Manage > Properties' ribbon option, clicking on the 'Connectors' page and selecting the required option in the 'Connector Notation' drop-down list. You would most probably change the notation to IDEFX1, but the UML2.1 notation is also available.

Default DBMS

Prior to creating a Physical Data Model it is advisable for you to set the default DBMS for the project. Setting a default DBMS ensures that all new database elements that are created on diagrams are automatically assigned the default DBMS.

If the default DBMS is not set, new Tables are created without a DBMS assigned, this restricts Enterprise Architect's ability to model the physical objects correctly. For example Enterprise Architect is unable to determine the correct list of datatypes for columns.

You can set the default DBMS type using:

- 'Start > Appearance > Preferences > Preferences > Source Code Engineering > Code Editors', or
- 'Settings > Reference Data > Settings > Database Datatypes or

- 'Develop > Data Modeling > Datatypes or
- The second data entry field in the Code Generation Toolbar

Note: When modeling via the Database Builder the default DBMS is defined at the model level (as a Tagged Value 'DBMS' against the <<Database>> Package) instead of at the project level, thereby allowing for greater flexibility when projects involve multiple DBMSs.

DDL Transformation

The DDL transformation converts the logical model to a data model structured to conform to one of the supported DBMSs. The target database type is determined by which DBMS is set as the default database in the model (see the *Database Datatypes* Help topic, 'Set As Default' option). The data model can then be used to automatically generate DDL statements to run in one of the system-supported database products.

The DDL transformation uses and demonstrates support in the intermediary language for a number of database-specific concepts.

Concepts

Concept	Effect
Table	Mapped one-to-one onto Class elements. 'Many-to-many' relationships are supported by the transformation, creating Join tables.
Column	Mapped one-to-one onto attributes.
Primary Key	Lists all the columns involved so that they exist in the Class, and creates a Primary Key Method for them.
Foreign Key	A special sort of connector, in which the Source and Target sections list all of the columns involved so that: • The columns exist • A matching Primary Key exists in the destination Class, and • The transformation creates the appropriate Foreign Key

MDG Technology to customize default mappings

DDL transformations that target a new, user defined DBMS require an MDG Technology to map the PIM data types to the new target DBMS.

To do this, create an MDG Technology .xml file named 'UserDBMS Types.xml', replacing UserDBMS with the name of the added DBMS. Place the file in the EA\MDGTechnologies folder. The contents of the MDG Technology file should have this structure:

```
<MDG.Technology version="1.0">
    <Documentation id="UserdataTypes" name="Userdata Types" version="1.0" notes="DB Type mapping for UserDBMS"/>
    <CodeModules>
        <CodeModule language="Userdata" notes="">
            <CodeOptions>
                <CodeOption name="DBTypeMapping-bigint">BIGINT</CodeOption>
                <CodeOption name="DBTypeMapping-blob">BLOB</CodeOption>
                <CodeOption name="DBTypeMapping-boolean">TINYINT</CodeOption>
```

```
            <CodeOption name="DBTypeMapping-text">CLOB</CodeOption>
            ...
        </CodeOptions>
    </CodeModule>
</CodeModules>
</MDG.Technology>
```

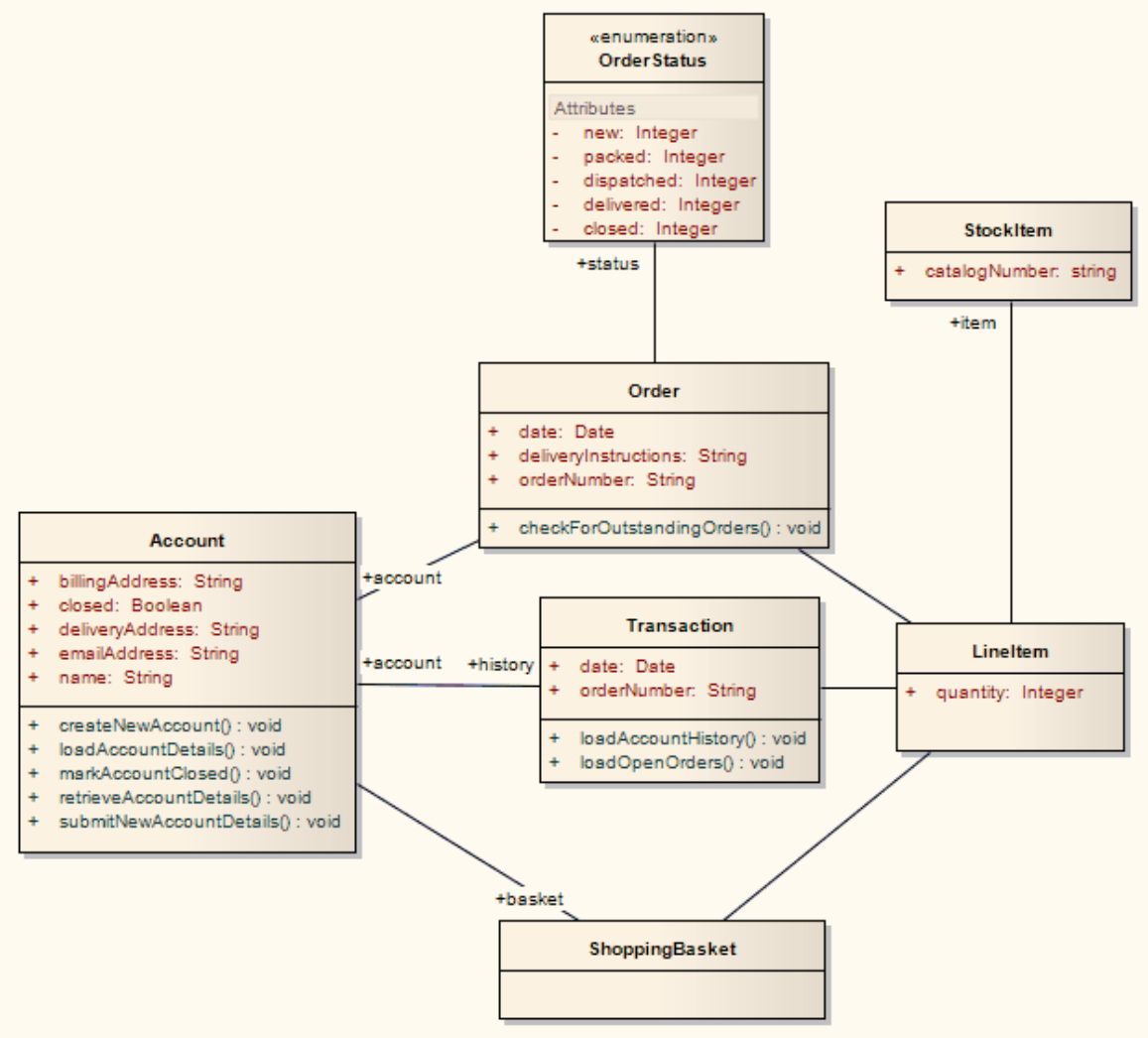
As an example, 'text' is a Common Type (as listed in the 'Database Datatypes' dialog) that maps to a new DBMS's 'CLOB' data type.

Notes

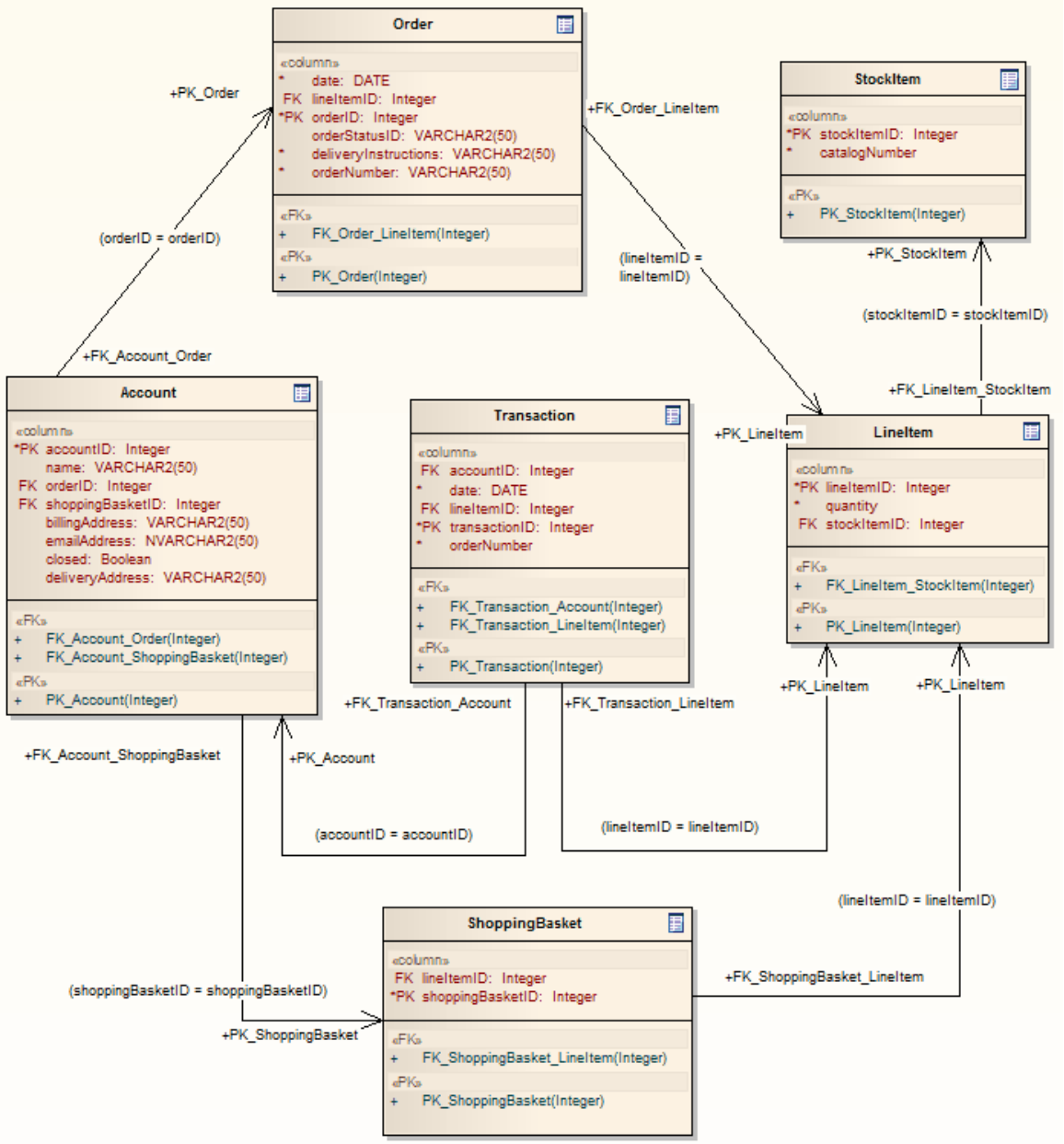
- You can define DBMS-specific aspects not depicted in a Logical model, such as Stored Procedures, Triggers, Views and Check Constraints, after the transformation; see the *Physical Data Model* Help topic

Example

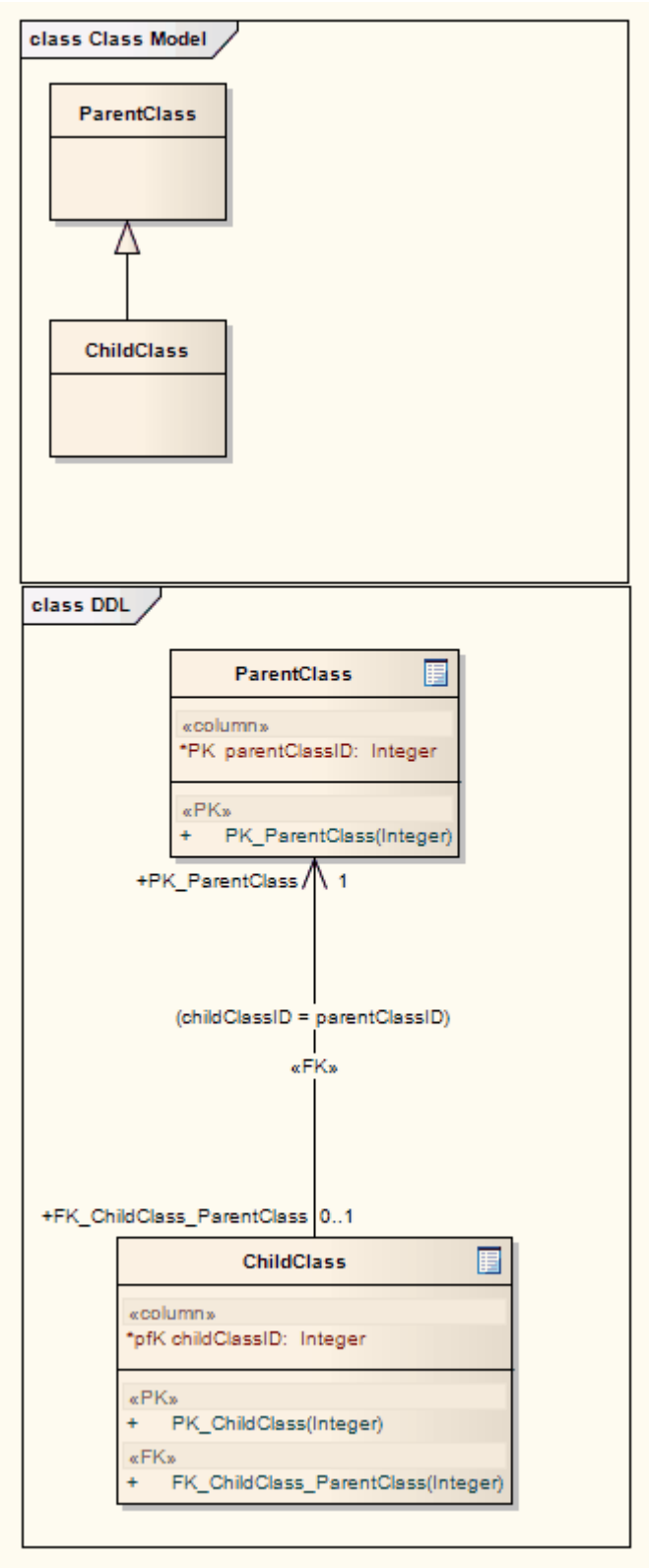
The PIM elements



After transformation, become the PSM elements

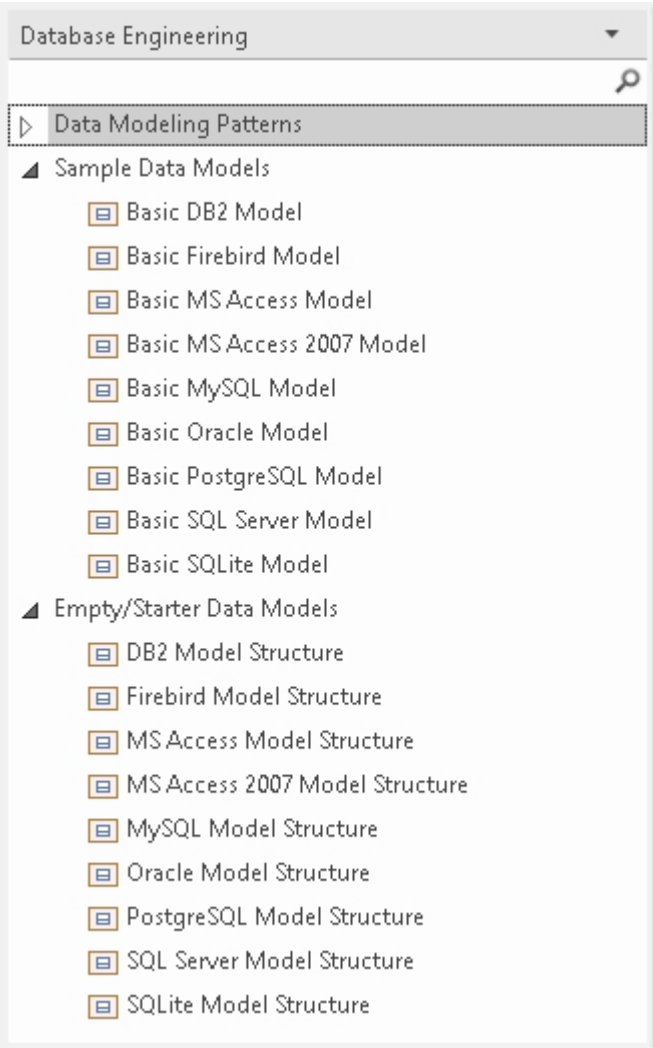


Generalizations are handled by providing the child element with a Foreign Key to the parent element, as shown. Copy-down inheritance is not supported.


class Class Model
ParentClass
ChildClass
class DDL
ParentClass
«column»
*PK parentClassID: Integer
«PK»
+ PK_ParentClass(Integer)
+PK_ParentClass
1
(childClassID = parentClassID)
«FK»
+FK_ChildClass_ParentClass
0..1
ChildClass
«column»
*pfK childClassID: Integer
«PK»
+ PK_ChildClass(Integer)
«FK»
+ FK_ChildClass_ParentClass(Integer)

Creating and Managing Data Models

Enterprise Architect is a fully featured database modeling platform that enables the user to work with their Physical Data models at all stages, from design right through to the implementation of the live database, for a wide range of database management systems such as Microsoft SQL Server, Oracle, PostgreSQL and MySQL.



This figure shows the starter model wizard patterns for database design for a range of RDBMS.

Create a Data Model from a Model Pattern

The easiest way to create a Data Modeling workspace is to use the predefined Database Model Patterns, available through the Model Builder. Enterprise Architect provides a Pattern for each DBMS supported by the system.

Access

Display the Model Builder using any of the methods outlined here.

In the Model Builder dialog, select the 'Database Engineering' Perspective.

Ribbon	Start > Personal > Model Builder Design > Package > Model Builder
Context Menu	Right-click on Package | Model Builder (pattern library)
Keyboard Shortcuts	Ctrl+Shift+M
Other	Browser window caption bar menu | Model Builder (pattern library)

Create a Data Model

Field/Button	Action
Add to Package	Displays the name of the selected root Package.
Technology	Click on 'Database'.
Name	If necessary, expand the Database Engineering group of Patterns. Click on the checkbox against each Database Management System you are supporting in the model.
All	Click on this button to select the checkboxes for all Database Engineering model types and the Entity Relationship diagram, to include them all in the model.
None	Click on this button to clear all selected checkboxes so that you can re-select certain checkboxes individually.
OK	Click on this button to add to the Browser window the Packages and diagram for each Database Management System you are modeling.

What each Data Modeling Pattern provides

- A summary diagram of the model

- A Report Specification Artifact element (on the summary diagram) that can be used to quickly document the data model
- A Package for each of the Logical and Physical models
- Within the Physical Model Package, a predefined hierarchy of sub-Packages, one for each object type supported by the DBMS being modeled (such as Tables, Views, Procedures and Functions); these automatically organize the database objects as they are added
- The DBMS type for the workspace
- A default owner
- A Data Modeling diagram in each Package with the connector notation set to IDEF1X

Notes

- Once a data modeling workspace has been created, you can begin to develop your model in one of two ways:
  - Through the Database Builder, which is a purpose-built view that supports database modelers
  - Through the Browser window and diagrams, which is the traditional method that might suit users who are experienced UML modelers

Create a Data Model Diagram

To model the structure of a relational database you use Data Modeling diagrams, which are extended Class diagrams. When you open a Data Modeling diagram the matching Diagram Toolbox is automatically opened, which contains the diagram elements:

- Table
- View
- Procedure
- Sequence
- Function
- Association and
- Database Connection

Access

Display the 'New Diagram' dialog using any of the methods outlined here.

Ribbon	Design > Diagram > Add Diagram
Context Menu	Right-click on Package | New Diagram Right-click on element | Add | New Diagram
Keyboard Shortcuts	Ctrl+Insert
Other	Browser window caption bar menu | New Diagram

Create a Data Modeling diagram

Field/Button	Action
Package	Defaults to the name of the Package selected in the Browser window or, if the parent is an element, the name of the Package containing that element. If you are adding a diagram directly to a Package and notice that it is not the correct Package, click on the  button and browse for the correct Package.
Parent	If you are adding a diagram to an element, this field displays the element name.
Diagram	This field defaults to the name of the parent Package or element. If required, overtype the default name with your preferred name.
Select From	Click on this header and select the Perspective Group and Perspective or Workspace most appropriate to the area you are working in (in this instance, 'Information Engineering > Database Models'). From the options listed in the panel, click on 'Extended'.

Diagram Types	Click on 'Data Modeling'.
OK	Click on this button to create the diagram. The Diagram View displays the blank diagram, and the 'Data Modeling' pages display in the Diagram Toolbox. Drag elements and connectors from the Toolbox onto your diagram, to create your data model.

Notes

- The default diagram connector notation for all new diagrams is Information Engineering, although many data modelers prefer the notation IDEF1X; to make this change select 'Design > Diagram > Manage > Properties > Connectors' and click on the required option in the 'Connector Notation' drop-down list

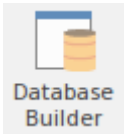
Example Data Model Diagram

This example of a Data Model diagram shows a data model of a bookstore warehousing system. The tables are modeled using a stereotyped class with a compartment for Columns which displays the name and type of the Columns. Primary and Foreign Keys are indicated by stereotypes on the columns. You can examine this model in greater detail in the Example model, installed with Enterprise Architect and available from this ribbon location.

Start > Help > Help > Open the Example Model

Data modeling diagram with suppressed operation compartment showing tables connected to indicate foreign key relationships.

The Database Builder

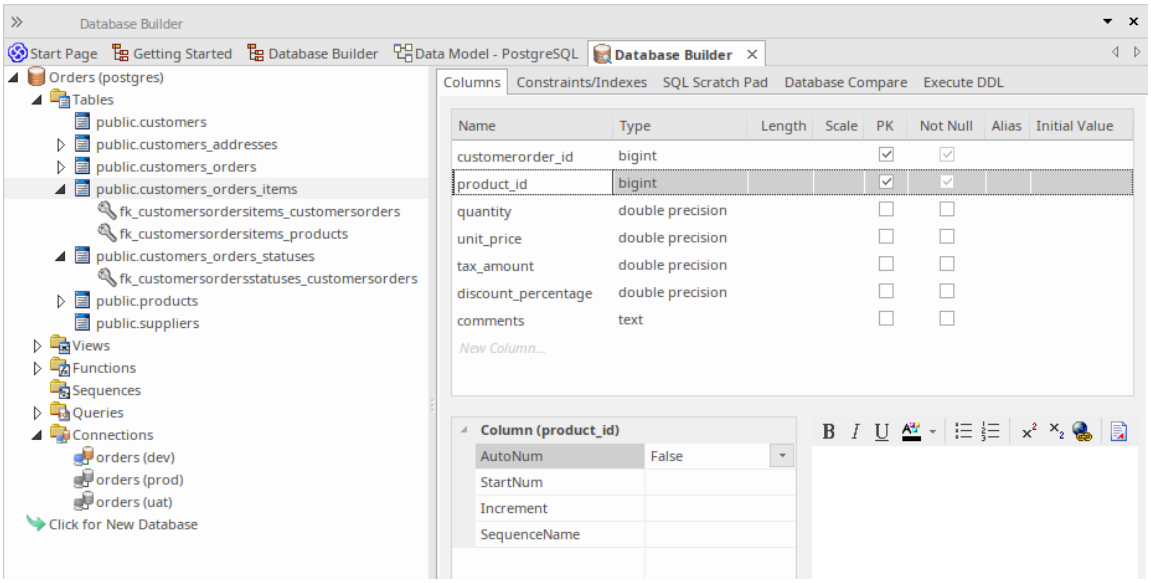


The Database Builder is a tailored interface for the data modeler; all database-related modeling tasks can be performed in a single location. The interface and its related screens include only the information relevant to data modeling, thereby streamlining and simplifying the modeling process.

Access

Ribbon	Develop > Data Modeling > Database Builder

Database Builder



This figure shows the Database Builder loaded with the 'Orders (postgres)' data model as it appears in the Example model.

Overview

The interface of the Database Builder consists of:

- A Tree of data models, listing all defined data models in the current repository
- A 'Columns' tab through which you directly manage the Table columns
- A 'Constraints/Indexes' tab for the direct management of Table constraints such as Primary Keys, Foreign Keys and Indexes
- An SQL Scratch Pad that you can use to run ad-hoc SQL queries against a live database

- A 'Database Compare' tab that displays the results of comparisons between the data model and a live database
- An 'Execute DDL' tab on which you can execute generated DDL against a live database, instantly

You can use the Database Builder to:

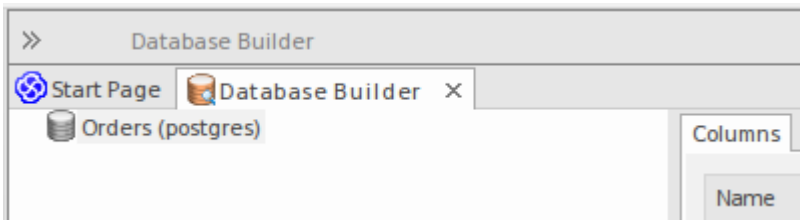
- Create, edit and delete database objects (Tables, Views, Procedures, Sequences and Functions)
- Create, edit and delete Table constraints (Primary Keys, Indexes, Unique Constraints, Check Constraints and Triggers)
- Create, edit and delete Table Foreign Keys
- Reverse engineer database schema information
- Generate DDL from a modeled database
- Compare a live database schema with a modeled database
- Execute generated DDL against a live database
- Execute adhoc SQL statements against a live database

Notes

- The Database Builder is available in the Corporate, Unified and Ultimate Editions of Enterprise Architect

Opening the Database Builder

When you first open the Database Builder, it searches the entire project for all Packages that have the stereotype <<Data Model>> and loads the corresponding data models as root nodes into the tree. A grayed-out icon indicates that the details of the data model are not loaded.



This figure shows the Database Builder with a single unloaded data model called 'Orders (postgres)'.

Using the Database Builder

You can start working in the Database Builder in one of these two ways:

Task	Action
Create a new data model	Once you have opened the Database Builder view, right-click in the empty space of the tree and select 'New Data Model' to invoke the Model Builder.
Load an existing Data Model	Once the Database Builder view is opened, load any of the defined data models by either: • Right-clicking on the name and selecting 'Load', or • Double-clicking on the name

Data Model Properties

In earlier versions of Enterprise Architect (prior to the introduction of the Database Builder) it was necessary for the data modeler to manually set properties on database objects before some tasks were allowed. For example, Enterprise Architect would not allow the definition of a Table column without the Table first being assigned a DBMS. This was because the DBMS controls the list of available datatypes.

To improve efficiency and the user experience, the Database Builder defines defaults for a number of properties at the data model level and then applies these default values automatically whenever new objects are created.

Properties

Option	Description
DBMS	*Defined against:* The data model's <<Database>> Package *Defined as:* Tagged Value *Details:* Defines the DBMS of the current data model

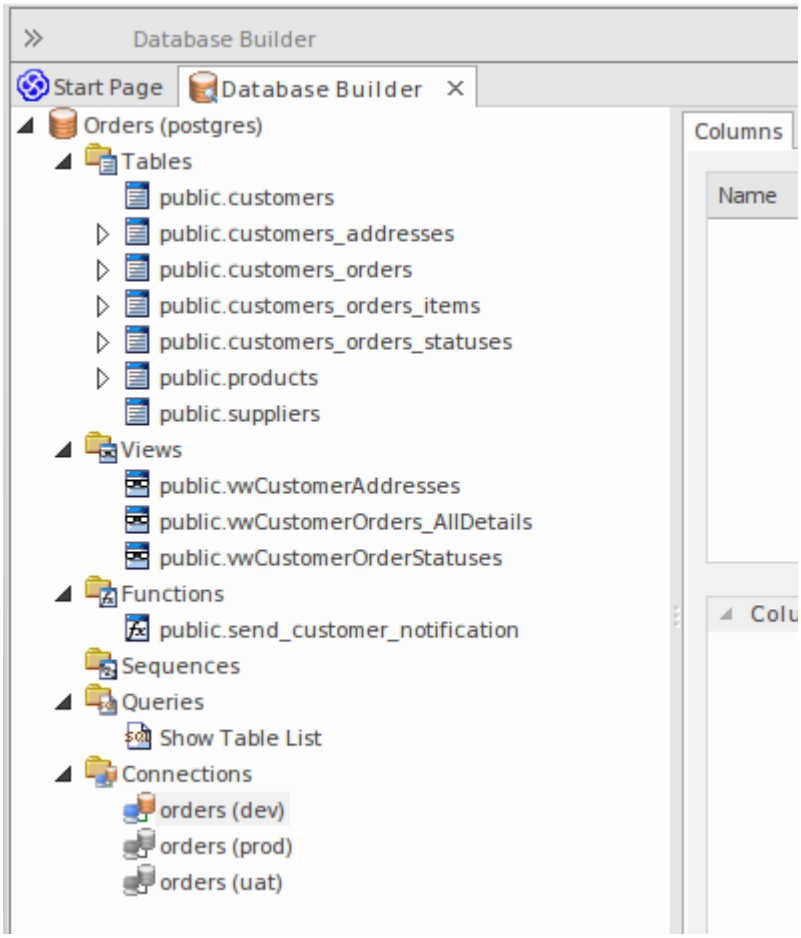
	Extra Information: • Controls which logical folders are shown for the current data model in the Database Builder's tree • Controls what DBMS rules are applied during database comparisons • Is automatically assigned to every new database object created in the current data model
DefaultOwner	*Defined against:* The data model's <<Database>> Package *Defined as:* Tagged Value *Details:* Defines the default Owner for the current data model *Extra Information:* • Is automatically assigned to every new database object created in the current data model, if the DBMS supports owners/schemas
DefaultConnection	*Defined against:* The data model's <<Database>> Package *Defined as:* Tagged Value *Details:* (Optional) the name of the default connection *Extra Information:* • Whenever a data model is loaded, the 'DefaultConnection' property is checked; if present, the Connection by that name is automatically made active • The database engineering model Patterns do not define a value for this property, it is created or updated whenever a user sets a Connection as the default

Notes

- If a data model is selected in the Browser window when the Database Builder is opened, that model's details will be automatically loaded

Working in the Database Builder

When a data model is loaded, the Database Builder creates a set of logical folders, one for each object type supported by the current DBMS. Each logical folder is populated with all objects of that type found in the data model's hierarchy of Packages (as shown in the Browser window).



In this image the data model 'Orders (postgres)' shows logical folders for Tables, Views, Functions, Sequences, Queries and Connections. It is worth noting there is no folder for 'Procedures' since PostgreSQL does not support database procedures.

Available Actions in the Database Builder Tree

The majority of the Database Builder functions are accessible via context menus. Each object in the Tree has its own set of unique menu items based on its type and status. This table describes the available context menu items and identifies which objects they apply to.

Menu Option	Applies to / Description
New data model	*Applies To:* Blank Space *Description:* Opens the Model Builder.
Refresh All	*Applies to:* Blank Space *Description:* Reloads the complete list of data models.
Load	*Applies to:* Root Node

	Description: Loads the full details of the data model.
Unload	*Applies to:* Root Node *Description:* Unloads the full details of the data model.
Import DB Schema	*Applies to:* Loaded Root Node *Description:* Opens the 'Import DB schema' dialog using the current active connection as the Live database source.
Generate DDL	*Applies to:* Loaded Root Node, Folder, Table, View, Procedure, Function, Sequence, Package *Description:* Opens the 'Generate DDL' dialog with the current object(s) selected.
Show Differences	*Applies to:* Loaded Root Node, Folder, Table, View, Procedure, Function, Sequence *Description:* Compares the selected objects to the current active connection.
Show Differences with Options	*Applies to:* Loaded Root Node, Folder, Table, View, Procedure, Function, Sequence, Package *Description:* Compares the selected objects to the current active connection and optionally ignore some of the differences based on the specified compare options.
Manage DBMS Options	*Applies to:* Loaded Root Node *Description:* Opens the 'Manage DBMS Options' dialog, which can be used to change the allocated DBMS and Owner of multiple objects.
View Record Count	*Applies to:* Table, View *Description:* Builds and runs a SELECT query (formatted to suit the element's DBMS) to show the number of records in the selected Table or View. If there is no active connection, you are prompted to select one.
View Top 100 Rows	*Applies to:* Table, View *Description:* Builds and runs a SELECT query (formatted to suit the element's DBMS) to show the top 100 rows of the selected Table or View. If there is no active connection, you are prompted to select one.
View Top 1000 Rows	*Applies to:* Table, View *Description:* Builds and runs a SELECT query (formatted to suit the element's DBMS) to show the top 1000 rows of the selected Table or View. If there is no active connection, you are prompted to select one.
View All Rows	*Applies to:* Table, View *Description:* Builds and runs a SELECT query (formatted to suit the element's DBMS) to show all rows of the selected Table or View. If there is no active connection, you are prompted to select one.
Properties	*Applies to:* Loaded Root Node, Folder, Table, View, Procedure, Function, Sequence, Package, Connection *Description:* Opens the standard 'Properties' dialog for the selected object.
Find in Project Browser	*Applies to:* Loaded Root Node, Folder, Table, View, Procedure, Function,

	Sequence, Package, SQL Query, Connection *Description:* Finds the selected object in the Browser window.
Refresh	*Applies to:* Loaded Root Node *Description:* Reloads the details of the current loaded data model. This is necessary when objects are added, changed or deleted by other users or when the changes are performed outside of the Database Builder.
Add new <type>	*Applies to:* Folder, Table, View, Procedure, Function, Sequence, Package, SQL Query, Connection *Description:* Creates a new object of the specified type.
Clone <name>	*Applies to:* Folder, Table, View, Procedure, Function, Sequence, Package, SQL Query, Connection *Description:* Makes a new copy of the selected object. When you select this option, a prompt displays on which you set the name and owner of the new object. For Table objects, you can choose which existing constraints should be copied (and set a name for each one) along with which Foreign Keys should be copied. For SQL-based objects, you can make any necessary changes to the SQL for the new element.
Delete <name>	*Applies To:* Table, View, Procedure, Function, Sequence, Package, SQL Query, Connection *Description:* Permanently deletes the selected object from the repository.
Add new Foreign Key on <name>	*Applies to:* Table *Description:* Creates a new relationship between the selected Table and another one, then shows the 'Foreign Key Constraint' screen for the new relationship.
SQL Object Properties	*Applies to:* View, Procedure, Function, Sequence *Description:* Opens the 'SQL Object Editor' screen.
Edit	*Applies to:* SQL Query *Description:* Loads the SQL (as defined in the selected element) into the SQL Scratch Pad.
Run	*Applies to:* SQL Query *Description:* Loads the SQL in the SQL Scratch Pad and runs it. If there is no active connection, you are prompted to select one.
Set as active DB Connection	*Applies to:* Connection *Description:* Flags the selected Database Connection as the active one for the current session.
Set as Default DB Connection	*Applies to:* Connection *Description:* Flags the selected Database Connection as the active one each time the data model is loaded.
DB Connection Properties	*Applies to:* Connection *Description:* Opens the 'Database Connection Properties' screen, to manage the connection settings.

Create/Edit/Delete Database Objects

The pages listed in this section describe in detail how to use the Database Builder's interface to create and manipulate database Tables; however, the process of creating and manipulating SQL-based database objects is documented in other areas. See these topics for details:

- Database Views
- Database Procedures
- Database Functions
- Database Sequences
- Database Connections

Database Connections in the Database Builder

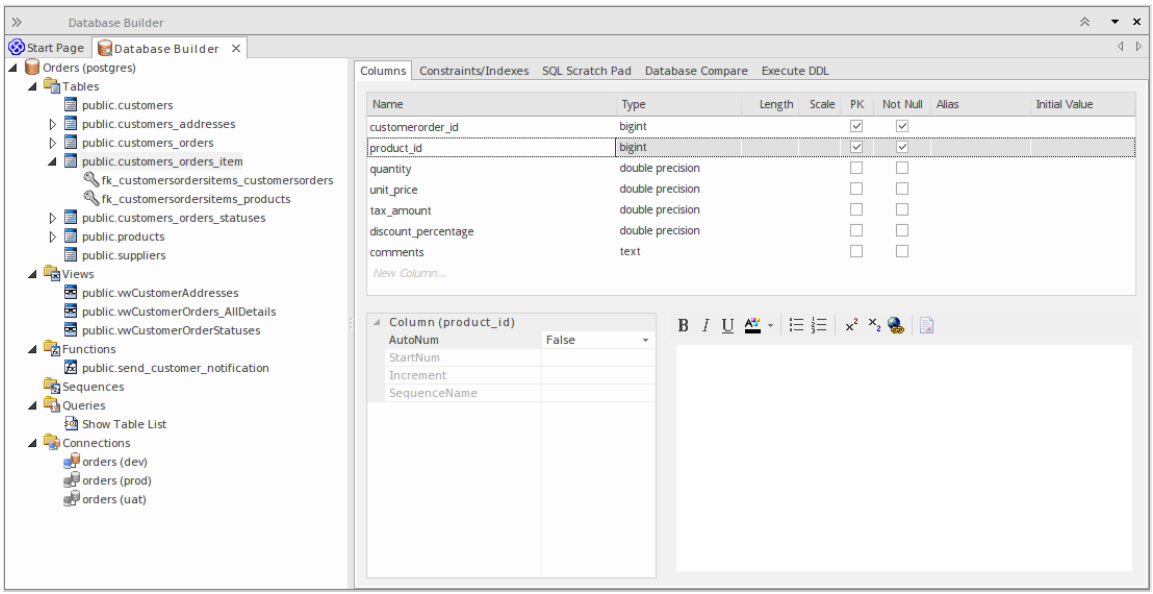
When performing certain tasks such as 'Compare' or 'Execute DDL', the Database Builder requires an active database connection. Only one database connection can be made active (indicated by a colored 'Database Connection' icon, while the others are gray) at a given time. If a database connection is not currently active and you try to perform a task that requires one, the Database Builder performs one of these actions based on how many connections are defined:

- 0 Connections – prompts you to create a connection and, if successful, continues
- 1 Connection – sets it as active and continues
- 2 (or more) Connections – prompts you to select one and, if successful, continues

Columns

Tables are the fundamental database object, and Columns (and their properties) are the most frequently modified Table feature updated and changed by data modelers, therefore the 'Columns' page is conveniently located as the first page of the Database Builder's interface.

Once a Table is selected in the Database Builder's tree, the 'Columns' page is populated with the currently defined list of columns for that Table. The data modeler can then make changes to main column properties directly in the list or grid. As the data modeler selects individual columns in the list, the column's extended properties (and Comments) are shown immediately under the list, allowing modification to these extended properties.



This figure show the Database Builder interface showing the tree of objects and the Columns tab showing the Columns for the selected table.

Notes

- The 'Columns' page will only be populated when a Table item is selected in the Database Builder's tree

Create Database Table Columns

A database Table column is represented in the UML Data Modeling Profile as an attribute with the <<column>> stereotype. For a selected Table, you can review the existing columns and create new columns, on the 'Columns' page of the Database Builder or on the 'Columns and Constraints' screen.

You can define column details directly on the list of columns on the 'Columns' tab. The changes are automatically saved as you complete each field. Some fields have certain restrictions on the data you can enter, as described here. The tab also contains a 'Properties' panel and a 'Notes' field, which are populated with the existing information on the selected column. Each new column that you create is automatically assigned a set of default values and added to the bottom of the list.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table > Columns > Right-Click > Add new Column
Context Menu	In diagram, right-click on required Table | Features | Columns | Right-Click | Add new Column
Keyboard Shortcuts	Select a table | F9 | Tab Key (to set input focus on the 'Columns' tab) | Ctrl+N

Create columns in a Table

Option	Action
Name	Overtype the default name with the appropriate column name text.
Type	Click on the drop-down arrow and select the appropriate datatype for the column. The available datatypes depend on the DBMS assigned to the parent Table.
Length	(Optional) Some datatypes have a length component - for example, VARCHAR has a length that defines the number of characters that can be stored. If the datatype does not have a length component, this field is disabled. If the field is available and if you need to define a number of characters, type the value here.
Scale	(Optional) Some datatypes have a scale component - for example, DECIMAL has a scale that defines the number of decimal places that can be held. If the datatype does not have a scale component, this field is disabled. If the field is available and if you need to define a scale, type the value here.
PK	Select the checkbox if the column is part of the Primary Key for this Table.
Not Null	Select the checkbox if empty values are forbidden for this column. The checkbox is disabled if the 'PK' checkbox is selected.

Alias	If required for display and documentation purposes, type in an alternative name for the field.
Initial Value	If required, type in a value that can be used as a default value for this column.
Notes	Type in any additional information necessary to document the column. You can format the text using the Notes toolbar at the top of the field.

Column Properties

The appropriate properties for the Table's Database Management System automatically display in the 'Property' panel (expand the 'Column (<name>)' branch if they are not visible).

Property	DBMS
Autonum (Startnum Increment)	Oracle MySQL SQL Server DB2 PostgreSQL Notes: If you require an automatic numbering sequence, set this property to True and, if necessary, define the start number and increment.
Generated	DB2 Notes: Set this additional property for auto numbering in DB2, to 'By Default' or 'Always'.
NotForRep	SQLServer Notes: Set this property to True if you want to block replication.
Zerofill	MySQL Notes: Set this property to True or False to indicate if fields are zerofilled or not.
Unsigned	MySQL Notes: Set this property to True or False to indicate whether or not fields accept unsigned numbers.
LengthType	Oracle Notes: Set this property to define the character semantics as 'None', 'Byte' or 'Char'.

Delete Database Table Columns

For a selected database Table, you can review the existing columns and delete any individual column, on the 'Columns' tab of the Columns and Constraints screen.

Access

Use one of the methods outlined here to display a list of columns for a table, then select a column and delete it.

When you select the 'Delete column '<name>" option, if all validation rules are satisfied the column is immediately deleted.

Ribbon	Develop > Data Modeling > Database Builder > Click on Table > Columns > Right-click on column name > Delete column <name>
Context Menu	In diagram, right-click on required Table | Features | Columns | Right-click on column name | Delete column <name>
Keyboard Shortcuts	F9 | Use 'Up Arrow' or 'Down Arrow' to select a column | Ctrl+D

Notes

- If the deleted database Table column is involved in any constraints it will automatically be removed from them

Reorder Database Table Columns

If you have several columns defined in a database Table, you can change the order in which they are listed. The order in the list is the order in which the columns appear in the generated DDL.

Access

Use one of the methods outlined here to display a list of columns for a Table, then select a column and reposition it within the list.

Ribbon	Develop > Data Modeling > Database Builder > Click on Table
Context Menu	In diagram, right-click on required Table | Features | Columns
Keyboard Shortcuts	F9

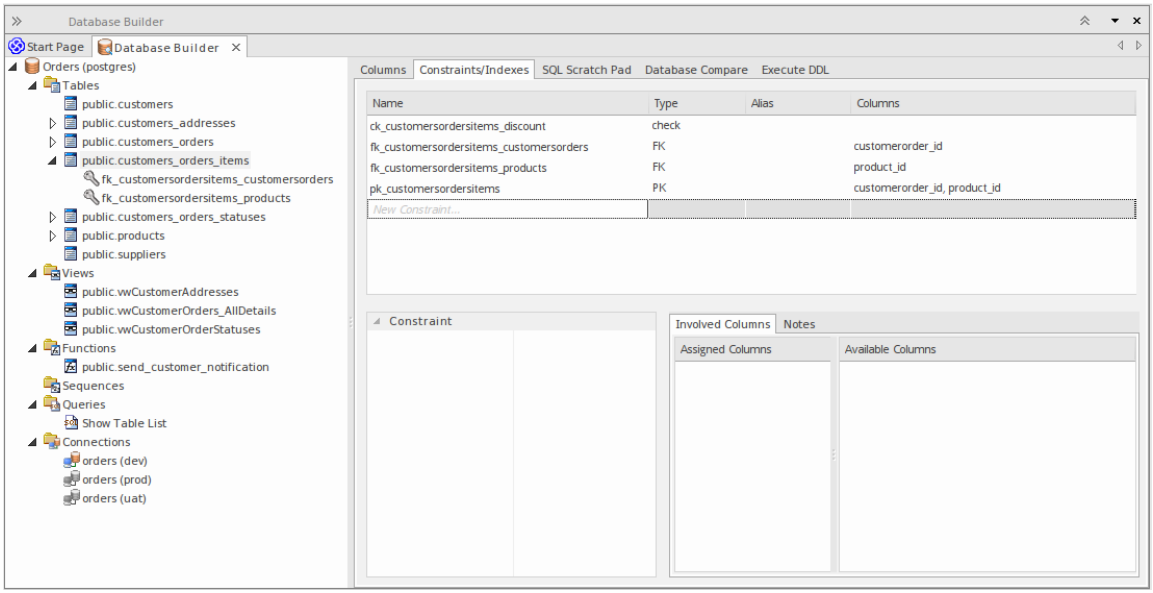
Change the column order

Step	Action
1	In the 'Columns' tab, click on the required column name in the list.
2	Right-click and select the: • 'Move column <name> up' option (or press Ctrl+Up Arrow) to move the column up one position • 'Move column <name> down' option (or press Ctrl+Down Arrow) to move the column down one position These options have an immediate effect both in the 'Columns' tab and on a diagram.

Constraints/Indexes

Tables are the fundamental database object, and Constraints and Indexes (and their properties) are the second most frequently modified Table feature updated and changed by data modelers, therefore the 'Constraints/indexes' page is conveniently located as the second page of the Database Builder's interface.

Once a Table is selected in the Database Builder's tree, the 'Constraints/Indexes' page is populated with the currently defined list of constraints and indexes for the selected Table. The data modeler can then make changes to main properties directly in the list. As the data modeler selects individual constraints or indexes in the list, the constraint's extended properties (and Comments) are shown immediately under the list, allowing modification of these extended properties.



This figure show the Database Builder interface showing the tree of objects and the Columns tab showing the Columns for the selected table.

Notes

- The 'Constraints/Indexes' page will only be populated when a Table item in the Database Builder's tree is selected

Database Table Constraints/Indexes

Within Enterprise Architect, Table Constraints and Indexes are modeled on the same screen; collectivity they are referred to as Constraints. Database Constraints define the conditions imposed on the behavior of a database Table. They include:

- Primary Key - uniquely identifies a record in a Table, consisting of one or more columns
- Index - improves the performance of retrieval and sort operations on Table data
- Unique Constraints - a combination of values that uniquely identify a row in the Table
- Foreign Key - a column (or collection of columns) that enforce a relationship between two Tables
- Check Constraints - enforces domain integrity by limiting the values that are accepted by a column
- Table Trigger - SQL or code automatically executed as a result of data in a Table being modified

In Enterprise Architect, you can define and maintain Table Constraints using either the purpose-designed 'Constraints/Indexes' page of the Database Builder or the Columns and Constraints screen.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes | Right-click | Add New Constraint
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes | Right-click | Add New Constraint
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Constraint

The process of creating any of these constraint types is the same and is achieved in one of the ways described here.

Create a Constraint - Using the context menu or keyboard

Step	Action
1	A new constraint is automatically created and assigned the default name *constraint n* (where *n* is a counter) and a 'Type' of 'index'. Overtype the default name with your own constraint name.
2	If necessary, in the 'Type' field click on the drop-down arrow and select the appropriate constraint type.
3	If you prefer, type an alias for the constraint, in the 'Alias' field. The 'Columns' field is read-only; it is populated with the columns that you assign to the 'Involved Columns' tab.

Create a Constraint - Overtype the template text

Step	Action
1	On the 'Constraints/Indexes' tab for the selected Table, the list of constraints ends with the template text *New Constraint*. Overtype this text with the appropriate constraint name, and press the Enter key.
2	The new constraint is automatically created and assigned the default Type of index. If necessary, in the 'Type' field click on the drop-down arrow and select the appropriate constraint type.
3	If you prefer, type an alias for the constraint, in the 'Alias' field. The 'Columns' field is read-only; it is populated with the columns that you assign to the 'Involved Columns' tab.

Assign Columns to a Constraint

The constraint types of Primary Key, Foreign Key, Index and Unique all must have at least one column assigned to them; this defines the columns that are involved in the constraint.

Step	Action
1	On the 'Constraints/Indexes' tab for the selected Table, click on the constraint to which you are assigning columns.
2	The 'Available Columns' panel lists all columns defined for the Table. For each column to assign to the constraint, right-click on the column name and select 'Assign column <name>'. The column name is transferred to the 'Assigned Columns' list.

Unassign Columns from a Constraint

Step	Action
1	On the 'Constraints/Indexes' tab for the selected Table, click on the constraint from which you are unassigning columns.
2	In the 'Assigned Columns' list, right-click on the name of the column to unassign from the constraint and select 'Unassign column <name>'. The column name is transferred to the 'Available Columns' list.

Reorder the Assigned Columns in a Constraint

If you have a number of columns in the constraint, you can re-arrange the sequence by moving a selected column name one place up or down the list at a time. To do this:

- Right-click on the column name to move and select either:
  - Move column '<name>' up (Ctrl+Up Arrow) or
  - Move column '<name>' down (Ctrl+Down Arrow)

Delete a constraint

To delete a constraint you no longer require, right-click on the constraint name in the list on the 'Constraints/Indexes' tab and select the 'Delete constraint <name>' option. If all validation rules for the given constraint type are met, the constraint is immediately removed from the repository along with all related relationships (if there are any).

Primary Keys

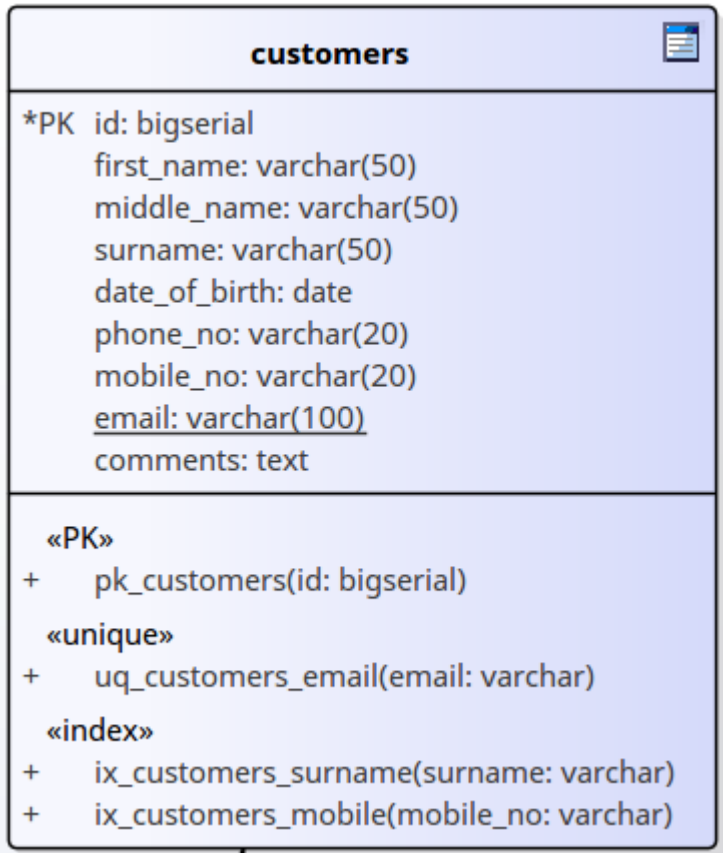
A Primary Key is a column (or set of columns) that uniquely identifies each record in a Table. A Table can have only one Primary Key. Some DBMSs support additional properties of Primary Keys, such as Clustered or Fill Factor.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes

Create a Primary Key

In Enterprise Architect you can create a Primary Key from either the 'Columns' tab or the 'Constraints/Indexes' tab. In either case, when you add a column to a Primary Key constraint, the column is automatically set to be 'Not Null'. Additionally any diagram (assuming the 'Show Qualifiers and Visibility Indicators' option is set) containing the Table element will show the 'PK' prefix against the column name. In this image, see the first column 'id: bigserial'.



Create a Primary Key - from the Columns tab

Step	Action
1	Either: • In the Database Builder, click on a Table with one or more defined columns, and click on the 'Columns' tab, or • On a diagram, click on a Table and press F9 to display the 'Columns' tab
2	For each column to include in the Primary Key, select the 'PK' checkbox. If a Primary Key constraint is not previously defined for the current Table, the system will create a new constraint using the Primary Key Name template.

Create a Primary Key - from the Constraints tab

Step	Action
1	Either: • In the Database Builder, click on a Table with one or more defined columns, and click on the 'Constraints/Indexes' tab, or • On a diagram, click on a Table and press F10 to display the 'Constraints/Indexes' tab
2	Overtype the *New Constraint* text with the Primary Key name, press the Enter key and click on the 'Type' field drop-down arrow, and select 'PK'.
3	Assign the required columns to the PK constraint.
4	Set the Primary Key's extended properties using the property panel. • Fill Factor is a numeric value between 0 and 100 • Is Clustered is a Boolean value that determines the physical order of how the data is stored; for most DBMSs the Is Clustered property defaults to True for Primary Keys

Remove columns from a Primary Key

You can remove columns from a Primary Key using either the 'Columns' tab or the 'Constraints/Indexes' tab.

Remove columns from a Primary Key - using the Columns tab

Step	Action
1	Either: • In the Database Builder, click on the Table with the Primary Key, and click on the 'Columns' tab, or • On a diagram, click on a Table and press F9 to display the 'Columns' tab

2	Against each column you want to remove from the Primary Key, deselect the 'PK' checkbox. If you have removed all columns from the Primary Key constraint and the Primary Key is no longer needed, it must be manually deleted.

Remove columns from a Primary Key - using the Constraints/Indexes tab

Step	Action
1	Either: • In the Database Builder, click on the Table with the Primary Key, and click on the 'Constraints/Indexes' tab, or • On a diagram, click on a Table and press F10 to display the 'Constraints/Indexes' tab
2	Unassign the columns on the PK constraint, as necessary.

Notes

- Warning: Enterprise Architect assumes that Primary Key constraints have at least one column assigned to them; however, Enterprise Architect does not enforce this rule during modeling
  If DDL is generated for a Table whose Primary Key has no column assigned, that DDL will be invalid

Database Indexes

Database indexes are applied to Tables to improve the performance of data retrieval and sort operations. Multiple indexes can be defined against a Table; however, each index imposes overheads (in the form of processing time and storage) on the database server to maintain them as information is added to and deleted from the Table

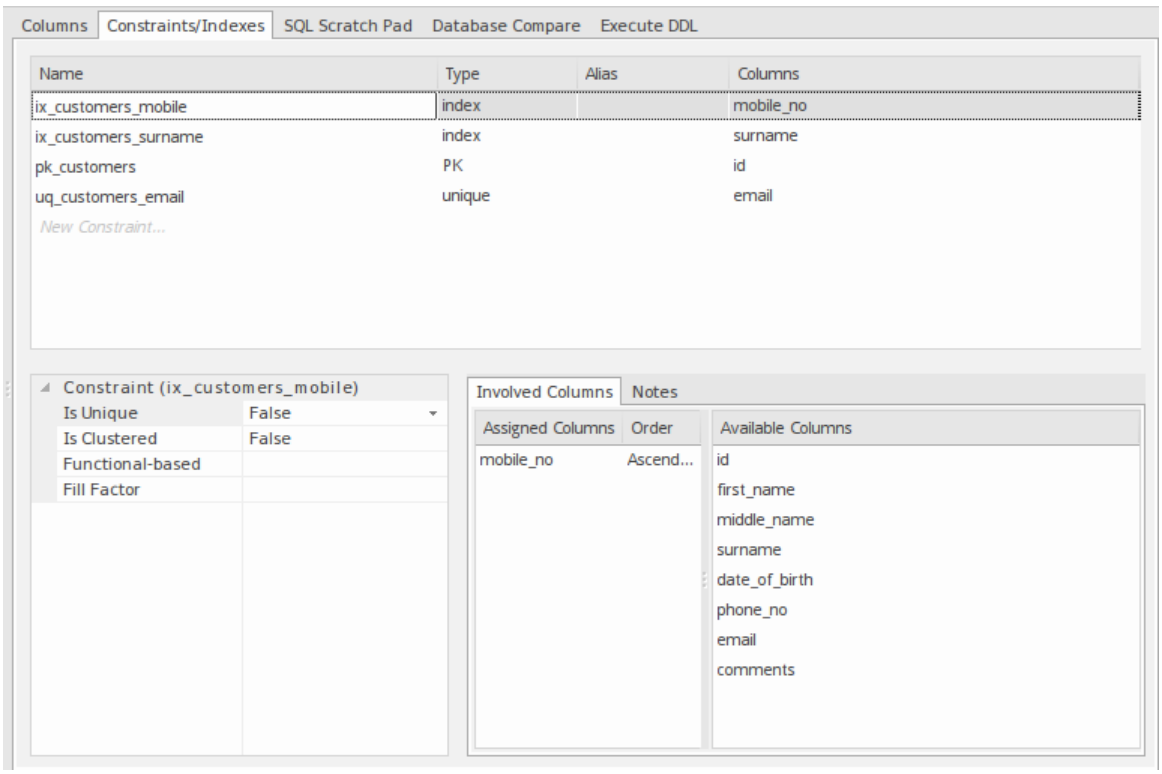
In Enterprise Architect an index is modeled as a stereotyped operation.

Some DBMSs support special types of index; Enterprise Architect defines these using additional properties such as function-based, clustered and fill-factor.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes

Work on an index



Step	Action
1	On the 'Constraints/Indexes' tab for the Table, right-click and select 'Add new constraint'.

	The new constraint is added with the default name 'constraint1' and the Type of 'index'. Overtype the name with your preferred index name.
2	Assign the appropriate columns to the Index. The 'Assigned Columns' list has an additional 'Order' field that specifies the order (Ascending or Descending) in which each assigned column is stored in the index. You can toggle the order for each column, as required. Additionally, for MySQL indexes, a 'Len' field will be visible in which you can define Partial Indexes; that is, an index that uses the leading 'n' number of characters of a text based field. The 'Len' field takes only whole number numeric values of between 0 and the column's defined length. A value of 0 (which is the default) indicates that the entire column is to be indexed.
3	In the 'Property' panel, review the settings of the extended properties that are defined for the current DBMS.

Additional Properties

Property	Description
Is Unique	(True / False) indicates whether the current index is a 'Unique Index'. A Unique Index ensures that the indexed column (or columns) does not contain duplicate values, thereby ensuring that each row has a unique value (or combination of values when the index consists of multiple columns).
Is Clustered	(True / False) indicates whether the current index is a 'Clustered Index'. With a clustered index, the rows of the table are physically stored in the same order as in the index, therefore there can be only one clustered index per table. By default a table's Primary Key is clustered. Not all DBMS's support clustered indexes, therefore the 'Is Clustered' Index property will only be visible for DBMSs that support it.
Is Bitmap	(True / False) indicates whether the current index is a 'Bitmap' index. Bitmap indexes are meant to be used on columns that have relatively few unique values (referred to as 'low cardinality' columns) and that physically consist of a bit array (commonly called bitmaps) for each unique value. Each of the arrays will have a bit for each row in the table. Consider this example: a bitmap index is created on a column called 'Gender', which has the options 'Male' or 'Female'. Physically, the index will consist of two bit arrays, one for 'Male' and one for 'Female'. The female bit array will have a 1 in each bit where the matching row has the value 'Female'. The 'Is Bitmap' and 'Is Unique' properties are mutually exclusive, and so the DDL generation will ignore the 'Is Unique' property when the 'Is Bitmap' property is True. Bitmap Indexes are only supported by Oracle; therefore, this property is only visible while modeling Oracle indexes.
Fill Factor	A numeric value between 0 and 100, that defines the percentage of available space that should be used for data. Not all DBMSs support fill factor, therefore the 'Fill Factor' index property will only be visible for DBMSs that support it.

Functional-based	A SQL statement that defines the function/statement that will be evaluated and the results indexed; for example: LOWER("field") Not all DBMSs support functional-based indexes, therefore the 'Functional-based' Index property will only be visible for DBMSs that support them, such as PostgreSQL and Oracle.
Include	Identifies a comma-separated list (CSV) of non-key Columns from the current table. Not all DBMSs support the 'Include' property on indexes, therefore this property will only be visible for DBMSs that support it.

Notes

- Warning: Enterprise Architect assumes that Indexes have at least one column assigned to them; however, Enterprise Architect does not enforce this rule during modeling
  If DDL is generated for a Table that has an Index defined without column(s) assigned, that DDL will be invalid, unless the index is functional-based
- Any columns assigned to a functional-based index are ignored

Unique Constraints

Unique Constraints enforce the 'uniqueness' of a set of fields in all rows of a Table, which means that no two rows in a Table can have the same values in the fields of a Unique Constraint. Unique Constraints are similar to Primary Keys (in that they also enforce 'uniqueness') but the main difference is that a Table can have multiple Unique Constraints defined but only one Primary Key.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes > Right-click > Add New Constraint
Context Menu	In diagram or Browser window | Right-click on Table element | Features | Constraints/Indexes
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Constraint

Step	Action
1	On the 'Constraints/Indexes' tab, a new constraint is automatically created and assigned the default constraint name and a 'Type' of index. Overtype the constraint name with a name that identifies this as a unique constraint.
2	In the 'Type' field, change the value from 'index' to 'unique'.

Notes

- Warning: Enterprise Architect assumes that Unique Constraints have at least one column assigned to them; however, Enterprise Architect does not enforce this rule during modeling
  If DDL is generated for a Table that has a unique constraint defined without column(s) assigned, that DDL will be invalid

Foreign Keys

A Foreign Key defines a column (or a collection of columns) that enforces a relationship between two Tables. It is the responsibility of the database server to enforce this relationship to ensure data integrity. The model definition of a Foreign Key consists of a parent (primary) Table containing a unique set of data that is then referred to in a child (foreign) Table.

In Enterprise Architect, a Foreign Key is modeled with two different (but related) UML components:

- A Foreign Key constraint (a UML operation with the stereotype of <<FK>>) stored on the child Table and
- An Association connector (stereotype of <<FK>>) defining the relationship between the two Tables

Create a Foreign Key

Although the definition of a Foreign Key can be complex, the Foreign Key Constraint screen simplifies the modeling of Foreign Keys. This screen is purpose-designed to help you select which constraint in the parent Table to use, and will automatically match the child Table columns to those in the parent Table that are part of the constraint. Different aspects of the process of developing a Foreign Key are described here separately for illustration, but the overall process should be a smooth transition.

A number of conditions must be met before a Foreign Key definition can be saved:

- Both Tables must have matching DBMSs defined
- The parent Table must have at least one column
- The parent Table must have a Primary Key, unique constraint or unique index defined

Create a Foreign Key - using the Database Builder

Step	Action
1	In the Database Builder tree, right-click on the child Table name and click on 'Add new Foreign Key on <table name>'. A dialog displays listing all the possible parent Tables.
2	Double-click on the required parent Table name in the list or select it and click on the OK button. The 'Foreign Key Constraint' screen displays.

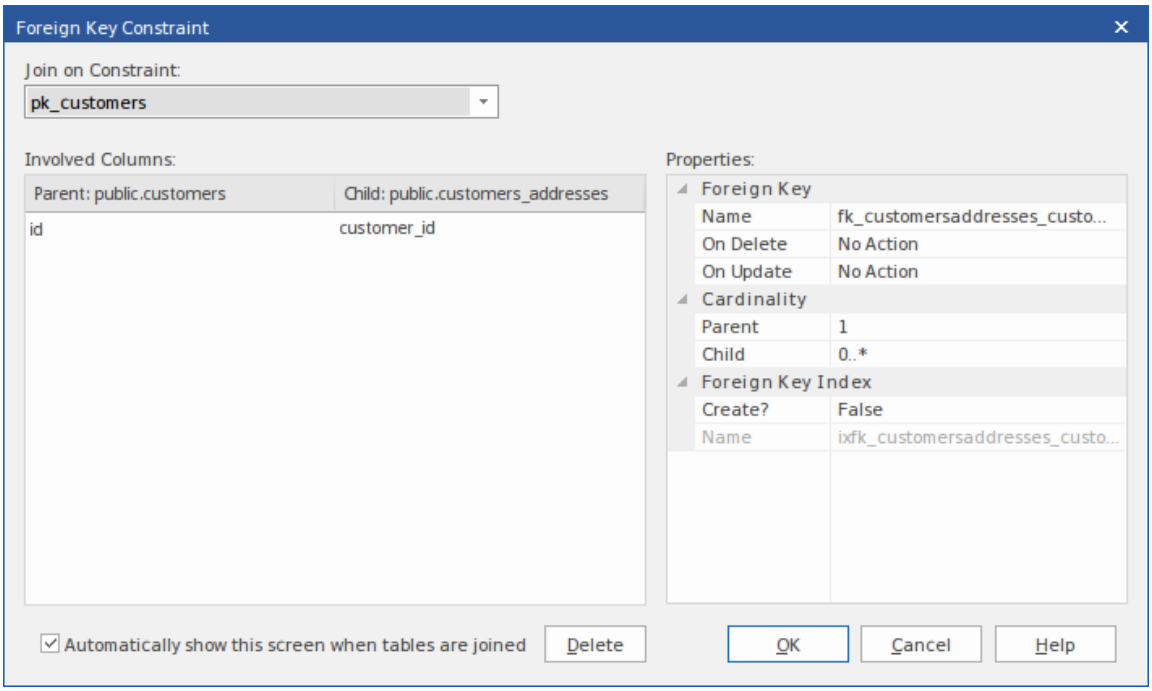
Create a Foreign Key - using a relationship on a diagram

Step	Action
1	In the Data Modeling diagram, locate the required child (Foreign Key) Table and parent (Primary Key) Table.
2	Select an Association connector in the 'Data Modeling' page of the Diagram Toolbox.
3	Click on the child Table and draw the connector to the parent Table.

4	If the Foreign Key Constraint screen has been set to display automatically when two Tables are joined, it displays now. Otherwise, either: • Double-click on the connector or • Right-click on the connector and select the 'Foreign Keys' option The Foreign Key Constraint screen displays.

The Foreign Key Constraint Screen

As an example this image shows the Foreign Key Constraint screen loaded with the details of 'fk_customersaddresses_customers' (as defined in the Example model).

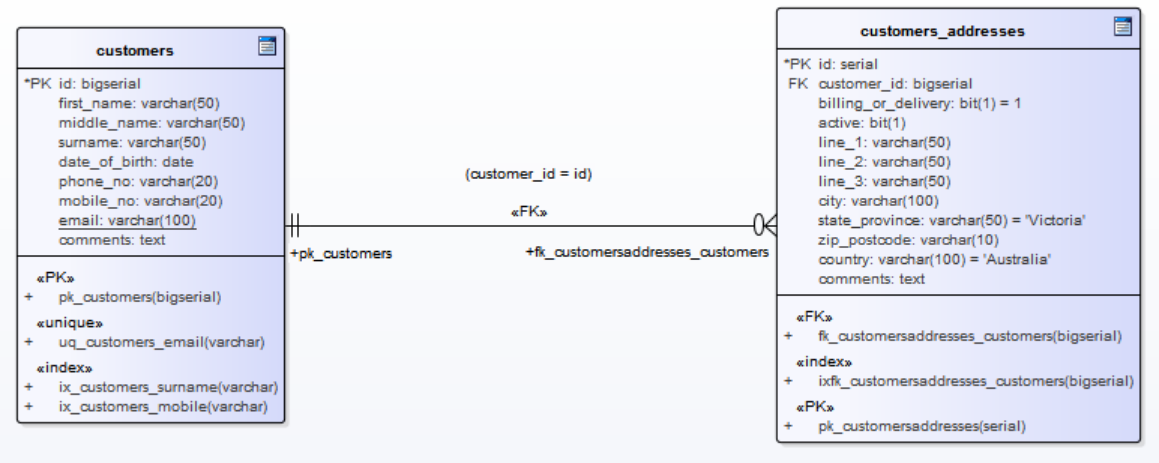


Option	Action
Join on Constraint	This combo box lists all defined constraints in the parent Table that could be used as the basis of a Foreign Key. (These constraints can be Primary Keys, Unique Constraints or Unique Indexes.) The first constraint in the list is selected by default; if this is not the constraint you want, select the correct constraint from the combo box. When you select the constraint, its columns are automatically listed in the 'Involved Columns' panel, under the 'Parent: <tablename>' column.
Involved Columns	This list is divided into two: the columns involved in the selected constraint are listed on the left, and the child columns that are going to be paired to the parent columns are listed on the right. When a constraint is selected (in the 'Join on constraint' field) the parent side is refreshed to display all columns assigned to the selected constraint. On the child side the system will automatically attempt to match each parent column to one of the same name in the child Table. If the child Table does not have a column of the same name, a new column of that name will be added to the list, flagged with (*) to indicate that a new column will be created in the Table.

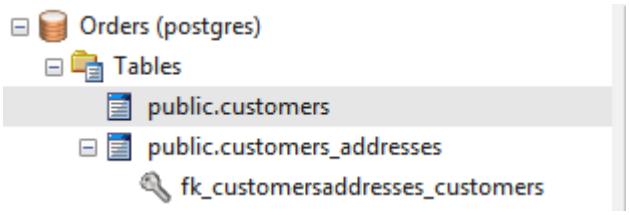
	However, if you want to force the pairing to an existing child Table column or a new column with a different name, click on the column name field and either: • Type in the replacement name, or • Select an existing column (click on the drop-down arrow and select the name from the list)
Name	This field defines the name of the Foreign Key constraint, and defaults to a name constructed by the Foreign Key Name Template. To change the name to something other than the default, simply overtype the value.
On Delete	Select the action that should be taken on the data in the child Table when data in the parent is deleted, so as to maintain referential integrity.
On Update	Select the action that should be taken on the data in the child Table when data in the parent is updated, so as to maintain referential integrity.
Parent	Click on the drop-down arrow and select the cardinality of the parent Table in the Foreign Key.
Child	Click on the drop-down arrow and select the cardinality of the child Table in the Foreign Key.
Create?	If you want to create a Foreign Key Index at the same time as the Foreign Key, set this property to True. The name of the Foreign Key Index is controlled by the Foreign Key Index template, and the generated name is shown in the 'Name' field underneath the 'Create?' field.
Automatically show this screen when tables are joined	(For diagrammatic modeling) Select this checkbox to automatically display this screen whenever an Association is created between two Tables.
Delete	Click on this button to delete the currently selected existing (saved) Foreign Key. A prompt is displayed to confirm the deletion (and the deletion of the Foreign Key Index, if one exists) - click on the Yes button. Deleting a Foreign Key leaves an Association connector in place, which you can either edit or delete (right-click and select 'Delete association: to <Table name>').
OK	Click on this button to save the Foreign Key.

Examples

This example shows simple Foreign Keys in a diagram:



The same Foreign Key will be shown in the Database Builder's tree as a child node under the Table 'customers.addresses'.



Check Constraints

A Check Constraint enforces domain integrity by limiting the values that are accepted by a column.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes > Right-click > Add New Constraint
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes | Right-click | Add New Constraint
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Constraint

Step	Action
1	On the 'Constraints/Indexes' tab of the Columns and Constraints screen, a new constraint is automatically created and assigned the default constraint name and a 'Type' of index. Overtype the constraint name with a name that identifies the constraint as a check constraint, such as 'CHK_ColumnName' (the CHK_ prefix is optional).
2	In the 'Type' field, change the value from 'index' to 'check'.
3	In the 'Properties' panel for the Condition property, type the SQL statement that will be used as the Check Condition; for example, column1 < 1000. If the condition is long, click on the  button to display a SQL editor (with syntax highlighting).

Delete a Check Constraint

If you do not want to keep a check constraint, either:

- Right-click on it in the list and select 'Delete constraint <name>', or
- Click on the item and press Ctrl+D

The constraint is immediately deleted.

Notes

- Any columns assigned to a check constraint are ignored

Table Triggers

A Table trigger is SQL or code that is automatically executed as a result of data being modified in a database Table. Triggers are highly customizable and can be used in many different ways; for example, they could be used to stop certain database activities from being performed during business hours, or to provide validation or perform deletions in secondary Tables when a record in the primary Table is deleted.

In Enterprise Architect, a Table trigger is modeled as a stereotyped operation and managed using the Table's 'Constraints' screen.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes | Right-click | Add New Constraint
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes | Right-click | Add New Constraint
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Table Trigger

Step	Action
1	On the 'Constraints/Indexes' tab, a new constraint is automatically created and assigned the default constraint name and a 'Type' of index. Overtype the constraint name with a name that identifies the constraint as a trigger, such as TRG_OnCustomerUpdate. (The TRG_ prefix is optional.)
2	In the 'Type' field, change the value from 'index' to 'trigger'.
3	In the 'Properties' panel for the Statement property, type in the complete SQL statement (including CREATE TRIGGER) that will define the Trigger. If the condition is long, click on the  button to display a SQL editor (with syntax highlighting).
4	The properties Trigger Time and Trigger Event are currently information-only values and are not used in DDL generation.

Delete a Table Trigger

If you do not want to keep a trigger, either:

- Right-click on it in the list and select 'Delete constraint <name>', or
- Click on the item and press Ctrl+D

The trigger is immediately deleted.

Notes

- Any columns assigned to table triggers are ignored

SQL Scratch Pad

The SQL Scratch Pad provides a mechanism to develop and run ad-hoc SQL Queries against a live database. While you develop your data model you might want to execute and test ad-hoc SQL Queries for a DDL script, or run enquiries on the live database; all of this is possible within the Enterprise Architect Database Builder interface.

The SQL Scratch Pad requires the Database Builder to have a valid connection to a live database. This database connection is shared between the 'SQL Scratch Pad', 'Database Compare' and 'Execute DDL' tabs of the Database Builder.

The Scratch Pad consists of:

- A toolbar providing facilities for importing, saving, executing and clearing the SQL Queries
- An editor panel in which you create or import the SQL Queries - this panel provides SQL-based syntax highlighting for the current data model
- A tabbed panel consisting of two pages, one to show the results of executing the Query and one to display any messages generated during the execution

Access

Open the Database Builder window, then display the 'SQL Scratch Pad' tab.

Ribbon	Develop > Data Modeling > Database Builder > SQL Scratch Pad

The Scratch Pad Toolbar

The functionality of each button on the Scratch Pad Toolbar is described in this table, working from left to right.

Button	Action
Run SQL	Executes the SQL Query currently shown in the Scratch Pad. Check the 'Results' and 'Messages' tabs for the output of executing the Query.
New	Clears the SQL Query editor fields so that you can enter a new query.
Open	Loads an SQL Query from file. A source file browser displays, defaulted to display SQL files. Click on the file name and on the Open button to display the file contents in the Scratch Pad.
Save to SQL Query	Saves this SQL statement to the SQL Query object it came from.
Save to New SQL Query	Creates a new SQL Query object and saves this statement to that object.
Save to File	Saves the currently-displayed Query to the file it came from. If you created the Query from scratch, a source file browser displays in which you type the new file name and click on the Save button to save the Query.
Save to New File	Saves the currently-displayed Query to a new .sql file.

	A source file browser displays on which you type in the new file name and click on the Save button to save the Query.
Clear	Clears the contents of the Scratch Pad. Any Query displayed in the Scratch Pad remains there until you either replace it with another Query from file or you close the model.
Toggle Comment	Applies the SQL comment characters '--' to the beginning of each selected line or, if the selected lines are already commented, removes the comment characters. Alternatively, press Ctrl+Shift+C.
Statement Separator	Type in the character(s) to use to mark the end of each statement.
Help	Displays the Help on the SQL Query Scratch Pad.
Query Description	Displays a label providing a description of the current SQL, whether there are pending changes (indicated by a leading *), and the name of the loaded SQL Query object or Filename.

Notes

- The SQL Scratch Pad does not manipulate your SQL in any way, so you must use the correct syntax for the current DBMS
- While the SQL Scratch Pad can execute multiple SQL statements, and the status and messages of each statement are shown in the 'Messages' list, only the results of one SELECT statement can be shown in the 'Results' list at a time; all subsequent SELECT statements will be ignored

Database Compare

The 'Database Compare' tab provides a mechanism for comparing the current data model with a live database, and optionally synchronizing any differences in either direction. Differences 'pushed' into a live database are performed using 'Alter DDL' statements, while changes imported from the live database can be directly 'pulled' into the model.

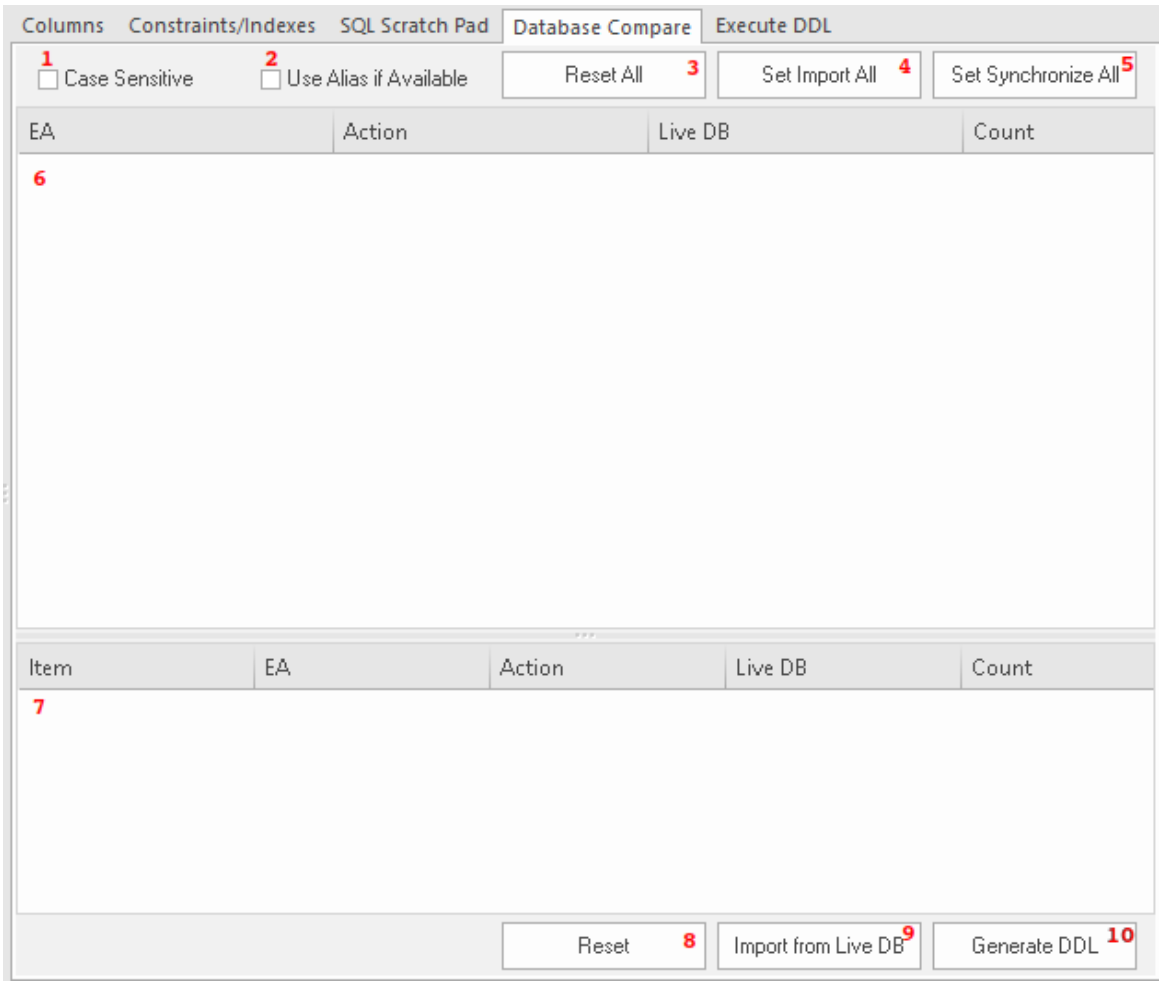
The Database Compare functionality requires the Database Builder to have a valid connection to a live database. This database connection is shared by the 'SQL Scratch Pad', 'Database Compare' and 'Execute DDL' tabs of the Database Builder.

Access

Open the Database Builder window, then display the 'Database Compare' tab.

Ribbon	Develop > Data Modeling > Database Builder > Database Compare

The DDL Compare Tab



The 'Database Compare' tab has a number of controls, as described here.

Number & Name	Description

1 Case Sensitive	Click on this checkbox to make all comparisons of properties recognize differences in letter-case in the property text.
2 Use Alias if Available	Click on this checkbox to indicate that any defined aliases should be used instead of object names (at both object and column level).
3 Reset All	Click on this button to set the 'Action' flag for all objects back to the default value.
4 Set Import All	Click on this button to set the 'Action' flag of all detected differences to <====; that is, update the model with the value(s) from the live database.
5 Set Synchronize All	Click on this button to set the 'Action' flag of all detected differences to ====>; that is, update the live database with the value(s) from the model.
6 Differences	Review the list of objects found to have mis-matches between the model and the live database. Selecting an item in this list will populate the 'Components' list. (See the *Differences List* table for a detailed description of each column.)
7 Components	Review this list of properties of the selected object that differ between the model and the live database. (See the *Component List* table for a detailed description of each column.)
8 Reset	Click on this button to set the 'Action' flag for all properties of the current object back to the default value.
9 Import from Live DB	Click on this button to import all properties' values (with the 'Action' of <===) from the live database into the model.
10 Generate DDL	Click on this button to generate the 'Alter DDL' statements for all objects with an 'Action' of ====>, and send the statements to the 'Execute DDL' tab.

Differences List

Column	Description
EA	Displays the name of each object in the model that has one or more detected differences. Blank values indicate that the object is missing in the model but exists in the live database.
Action	Defaults to 'No Action' as the action to take considering this object's difference(s). Click on the drop-down arrow and select a specific action. The list of available actions in the list will depend on whether or not the given object is paired in the model and live database. **Paired objects** • No Action - do not update the database or model with this change • ===> - update the object in the database from the model • <=== - update the object in the model from the database • Customize - set the items to No Action prior to setting different actions on each

	item in the lower panel • Unpair - separate the paired objects so that they are not compared with each other or updated from each other **Unpaired objects** • Create <object name> - create the missing database object in the database or model, as appropriate • Delete <object name> - delete the object from the model • Drop <object name> - delete the object from the database • Pair with <object name> - pair the object in the database with the named (unpaired) object in the model, so that they are compared for differences between them The 'Action' fields in the 'Components List' (the lower panel) will be updated based on the selection of this field. For example, if the live database has a Table column 'Address1' and the model doesn't, setting the object 'Action' to '**===>**' (update the object in the database from the model) sets the column 'Item Action' to 'Drop Address1', which will remove the column from the live database.
Live DB	Shows the name of each object in the live database that has one or more detected differences. Blank values indicate that the object exists in the model but is missing in the live database.
Count	Shows the total number of detected differences for the object (and all of its components) between the model and live database.

Component List

Column	Description
Item	Shows the component name or description for each detected difference. The differences are grouped into three categories: Properties, Columns and Constraints, in a tree structure.
EA	Shows the value of the given component as detected in the model. Blank values indicate that the value is missing in the model but exists in the live database.
Action	Defaults to the action corresponding to the setting of the object 'Action' field in the 'Differences' list, to indicate the action to take regarding the difference detected for the component. Click on the drop-down arrow to select an alternative action; the available options in the list depend on the component's type and the detected difference. • No Action - do not update the database or model • ===> - update the object in the live database from the model • <=== - update the object in the model from the live database • Add <item name> - create the missing item in the database or model, as appropriate • Delete <item name> - delete the item from the model

	• Drop <item name> - delete the item from the live database
Live DB	Shows the value for the selected component in the live database. Blank values indicate that the value exists in the model but is missing in the live database.
Count	Shows the number of differences between the model and the live database detected in the selected component.

Working with the Database Comparison

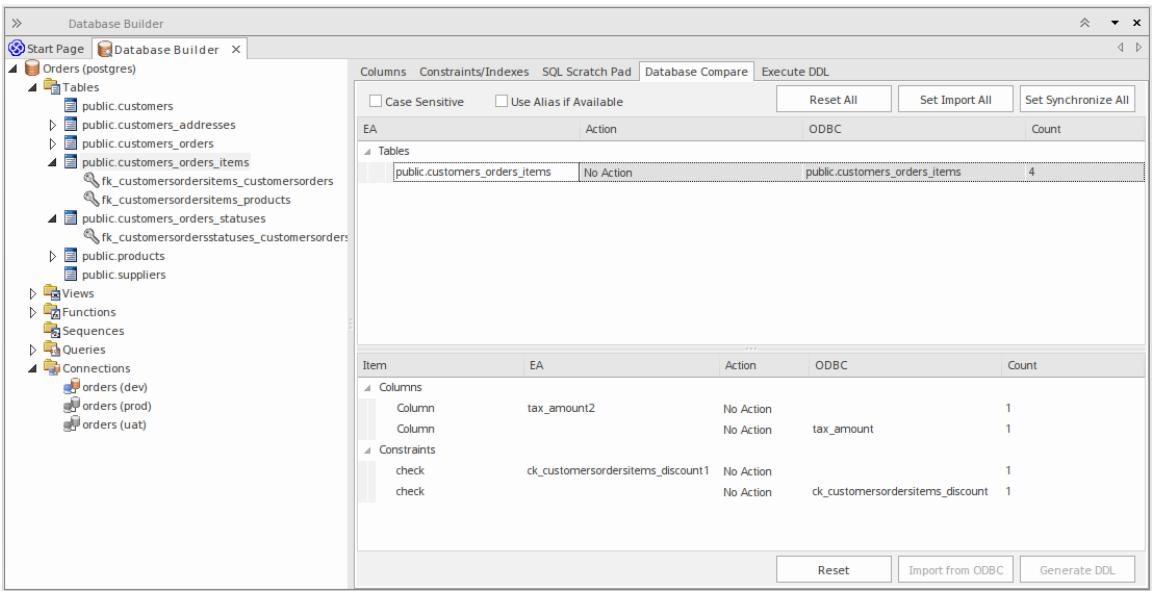
Whenever you perform a comparison, Enterprise Architect reads the definition from both the live database and the model, and then attempts to 'pair' each object from one source with the other, using its name (and schema, if relevant for the current DBMS).

If a match is found, the object name is shown in both the 'EA' and 'Live DB' columns with a default action of 'No Action'. The 'Count' column indicates the total number of differences found for the object and its components or properties.

If a match is not found between the systems, the object name is shown in the source column (either 'EA' or 'Live DB') while the other column is blank. In this state it is possible to pair the object with an object of a different name; the 'Action' dropdown list will present the available objects. If a new pairing is made the two objects' definitions are compared for differences and the results are shown in the 'Components' list, with the default action of '====>' selected.

If you select an action at the object level, this will set the matching action for all of the object's components and properties. However, if you select the 'Customize' action at the object level, you can determine a different action for each component.

As an example, both a column (tax_amount) and constraint (ck_customersordersitems_discount) were renamed in Table 'public.customers_order_items' (in the Example model) and a database compare performed; this image shows the differences found:



In the image there is only one Table that had detected differences - 'public.customers_order_items'; selecting this populates the 'Components' list. From the detected results it can be determined that the data model contains a column (tax_amount2) and a check constraint (ck_customerordersitems_discount1) that the live database doesn't and in turn the live database contains a column (tax_amount) and a check constraint (ck_customerordersitems_discount) that the data model doesn't.

Comparing with Options

The 'Compare with Options' functionality works in the same manner as for a direct comparison, except that you are prompted to choose which object/property comparisons should be performed. This enables you to ignore particular differences that are not of relevance at the current time.

These tables describe the different comparisons that can be enabled or disabled.

All Objects, Owner

Comparison	Action
Owner	Select to indicate that the 'Owner' property of all database objects should be compared, after the objects have been 'paired'.

Table Options

Option	Action
Tables	Select this parent option to enable all of the Table comparison options. Deselect to disable all the other options. You would then deselect or select specific options in the list.
Table - Extended Properties	Select to indicate that extended properties of Tables (such as DB Version and Tablespace) should be compared.
Table - Remarks	Select to indicate that remarks applied to Tables should be compared.
Columns	Select this parent option to enable all of the 'Column comparison' options. Deselect to disable all the other 'Column' options. You would then deselect or select specific options in the list.
Column - Type	Select to indicate that the datatype name for the Table Columns should be compared.
Column - Size	Select to indicate that the datatype size for the Table Columns should be compared.
Column - Default Value	Select to indicate that the default values of the Table Columns should be compared.
Column - Position	Select to indicate that the Table Column positions should be compared.
Column - Not Null	Select to indicate that the not null property of the Table Columns should be compared.
Column - Auto Numbering	Select to indicate that the autonumbering properties for the Table Columns should be compared (such as AutoNum, StartNum and Increment).

Column - Unmatched Columns	Select to indicate that Table Columns that are unmatched between the model and the live database should be compared. Typically these are columns that exist in one system but do not exist in the other.
Column - Extended Properties	Select to indicate that extended properties of Table Columns (such as Unsigned and Zerofill) should be compared.
Column - Remarks	Select to indicate that remarks applied to Table Columns should be compared.
Constraints	Select this parent option to enable all of the 'Table Constraint comparison' options. Deselect to disable all the 'Table Constraint' options. You would then deselect or select specific options in the list.
Constraint - Primary Keys	Select to indicate that properties related to Primary Keys should be compared.
Constraint - Foreign Keys	Select to indicate that properties related to Foreign Keys should be compared.
Constraint - Indexes	Select to indicate that properties related to Indexes should be compared.
Constraint - Unique Constraints	Select to indicate that properties related to Unique Constraints should be compared.
Constraint - Check Constraints	Select to indicate that properties related to Check Constraints should be compared.
Constraint - Table Triggers	Select to indicate that properties related to Table Triggers should be compared.
Constraint - Unmatched Constraints	Select to indicate that Table Constraints that are unmatched between the model and the live database should be compared. Typically these are constraints that exist in one system but do not exist in the other.
Constraints - Extended Properties	Select to indicate that extended properties of Table Constraints (such as Fill Factor and Clustered) should be compared.
Constraints - Remarks	Select to indicate that remarks applied to Table Constraints should be compared.

Notes

- The Database Compare functionality currently can perform comparisons on Table, View, Procedure, Function and Sequence object types

Execute DDL

The 'Execute DDL' tab provides a mechanism to easily execute generated DDL statements against a live database, and provides instant feedback on their success, all within the Enterprise Architect interface and without the need for other products.

There are two different types of DDL statement that Enterprise Architect can generate and send to the 'Execute DDL' tab:

- Create DDL statements, created by the Generate DDL screen, and
- Alter DDL statements, created by the Database Compare window

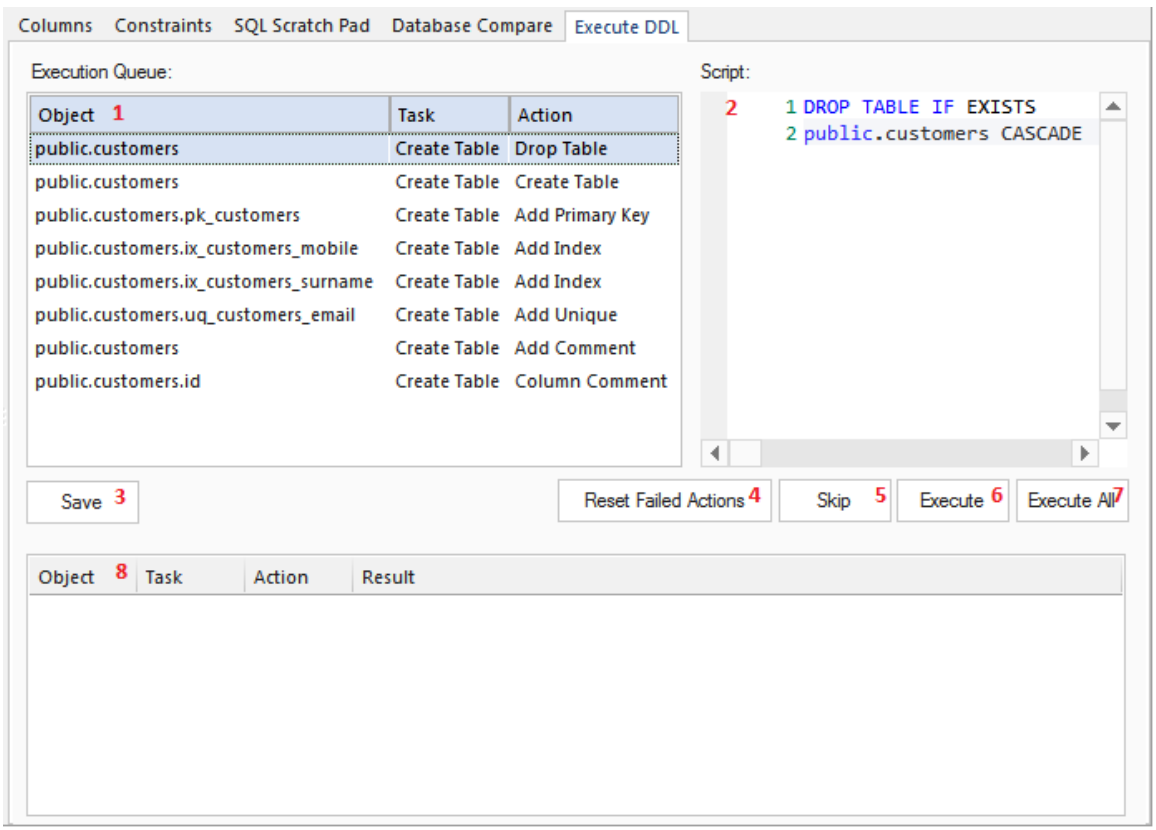
The Execute DDL functionality requires the Database Builder to have a valid connection to a live database. This database connection is shared between the SQL Scratch Pad, Database Compare and 'Execute DDL' tabs of the Database Builder.

Access

Open the Database Builder window, then display the 'Execute DDL' tab.

Ribbon	Develop > Data Modeling > Database Builder > Execute DDL

Execute the DDL



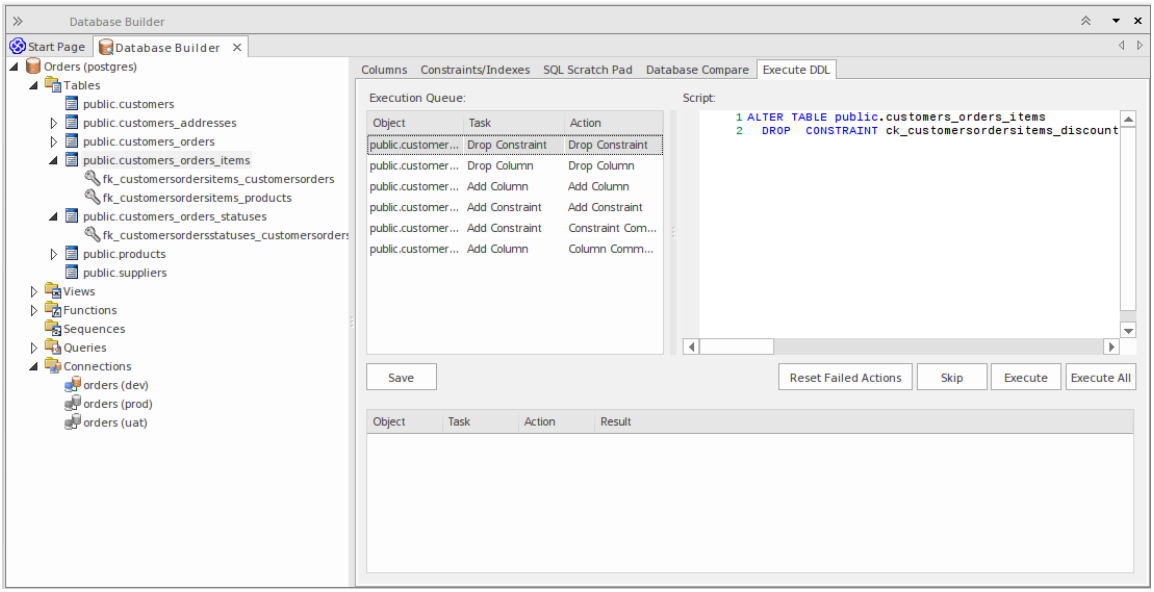
The 'Execute DDL' tab has these fields and buttons:

Field/Button	Action

1 Execution Queue	Lists the tasks (each with an associated DDL statement) that are yet to be executed. The list has three columns that specify the name of the object involved, the task and the action being performed. Selecting an item in the list will display the associated DDL statement (in the 'Script' field) for the given task.
2 Script	A text box with SQL syntax highlighting, showing the DDL statement for the selected task.
3 Save	Click on this button to save all the individual DDL statements from both the 'Execution Queue' and the 'Results List' into a single file.
4 Reset Failed Actions	Click on this button to re-queue any failed or skipped tasks from the 'Results List' to the bottom of the 'Execution Queue'.
5 Skip	Click on this button to skip over the next task in the 'Execution Queue' and not execute it. The task will be moved into the 'Results List' and not given a result. When you click on the Reset Failed Actions button, skipped tasks are returned to the Execution Queue along with any failed tasks.
6 Execute	Click on this button to execute the next task in the 'Execution Queue'. The task is removed from the top of the 'Execution Queue' and added to the end of the 'Results List' with the execution result.
7 Execute All	Click on this button to execute all tasks in the 'Execution Queue'. When execution is complete, the 'Results List' will display the results of each individual task.
8 Results List	Lists the executed tasks with the results of execution for each task. Selecting an item in this list will display the DDL statement that was executed, in the 'Script' field.

Example

In the example used in the earlier section on Database Comparison (when a column and constraint were renamed), if the defaults are used to 'push' the data model changes into the live database the Execute DDL screen is populated with the details shown here.



In summary, DDL is generated to drop both the old column and the old constraint (tasks 'Drop Column' and 'Drop Constraint'), then the column and constraint are created with the new names (tasks 'Add Column' and 'Add Constraint') and finally each has their comments/remarks applied (tasks 'Add Constraint - Constraint Comment' and 'Add Column - Column Comment').

Database Objects

Whilst Tables are the fundamental components of a relational database and allow the definition of Columns, Data Types, Keys and Indexes, there are a number of other Objects that are important in RDBM systems including:

- Views - a View represents the result-set of a pre-defined query; they are dynamically derived from the data stored in one or more Tables (or other Views)
- Procedures - a feature that some DBMS products implement to provide subroutines that can contain one or more SQL statements to perform a specific task such as data validation, access control, or to reduce network traffic between clients and the DBMS servers
- Functions - a feature that some DBMS products implement to provide a mechanism to extend the functionality of the database server; each is a routine that can accept parameters, perform an action (such as a complex calculation) and return the result of that action as a value
- Sequences - a feature that some DBMS products implement to provide a mechanism to generate unique values - the Sequence ensures that each call to it returns a unique value

The UML itself does not specify how data modeling is performed, but Enterprise Architect has a fully integrated UML profile for data modeling and a range of features built in to the core product that will make data modeling easy.

The profile uses stereotypes and Tagged Values to extend standard UML elements into data modeling constructs. This is achieved by adding the database object stereotype to a UML Class; so that you would model:

- Data Modeling diagrams as extended UML Class diagrams
- Tables as UML Class objects with a stereotype of <<table>>
- Views as UML Class objects with a stereotype of <<view>>
- Procedures as UML Class objects with a stereotype of <<procedure>>
- Functions as UML Class objects with a stereotype of <<function>>
- Sequences as UML Class objects with a stereotype of <<dbsequence>>

You can quickly create and configure all of these objects in your database model with Enterprise Architect.

Database Tables

Tables are the fundamental components of a relational database, representing multiple rows of structured data elements (referred to as Columns). Every individual item of data entered into a relational database is represented by a value in a column.

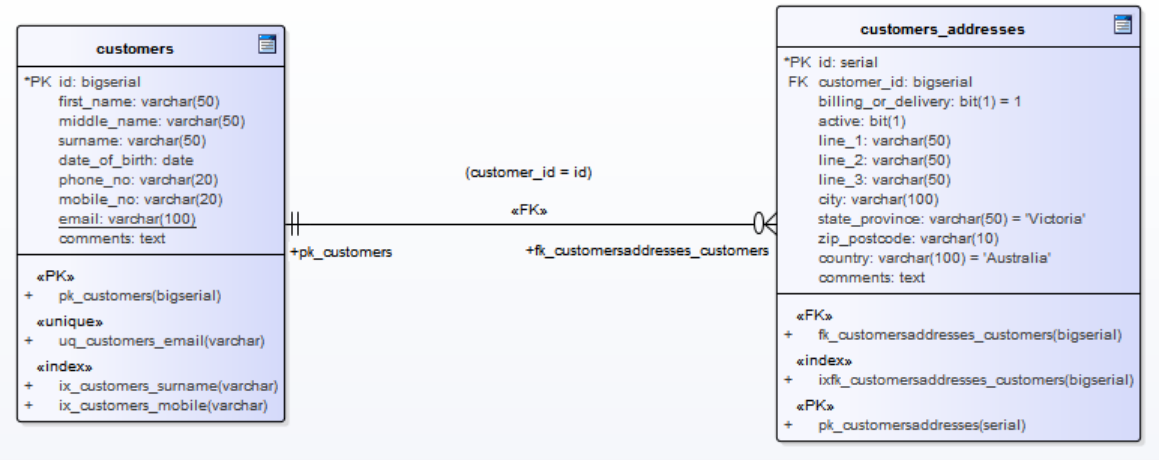
Enterprise Architect's UML Profile for Data Modeling represents:

- Database Tables as UML Class objects with a stereotype of <<table>>
- Table columns as UML attributes of a Table, with a stereotype of <<column>>
- Primary Keys as UML operations/methods of a Table, with a stereotype of <<PK>>
- Foreign Keys as UML operations/methods of a Table, with a stereotype of <<FK>>
- Indexes as UML operations/methods of a Table, with a stereotype of <<index>>
- Unique Constraints as UML operations/methods of a Table, with a stereotype of <<unique>>
- Check Constraints as UML operations/methods of a Table, with a stereotype of <<check>>
- Table Triggers as UML operations/methods of a Table, with a stereotype of <<trigger>>

Enterprise Architect refers to all of the UML operations of a Table collectively as Constraints, hence the screen you use to maintain a Table's UML attributes and operations is called the Columns and Constraints screen.

Example

This simple example of a Physical Data Model diagram in Enterprise Architect consists of two Database Tables represented by UML Classes, named *customers* and *customer_addresses*.



Each Table defines database columns, using UML attributes typed appropriately for the target DBMS (in this case, PostgreSQL).

Notes

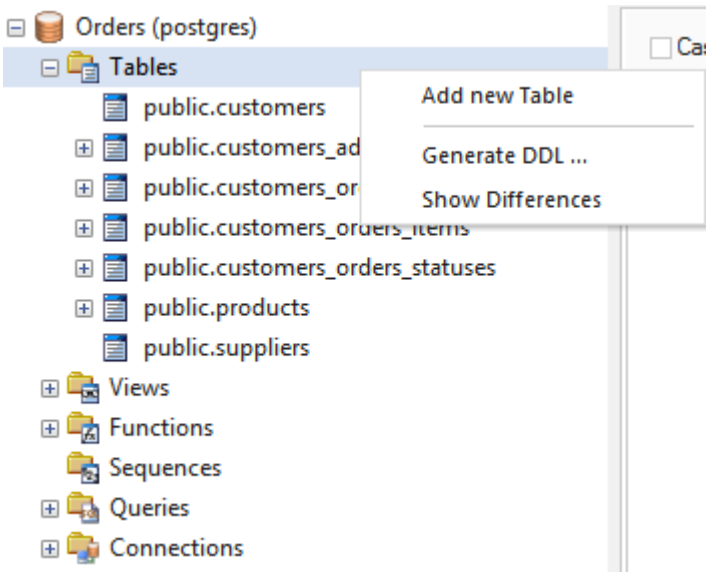
- The Table stereotype is denoted by the icon in the top-right corner of each Class (see the *Data Modeling Notation* topic)
- The Enterprise Architect maintenance screen for managing Table Columns doesn't allow you to change the attributes stereotype, since <<column>> is the only valid option
- It is possible to hide the <<column>> stereotype label shown in the example Tables (see the *Data Modeling Notation* topic)

Create a Database Table

Fundamental to data modeling is the creation of Database Tables within the model. There are three ways to create a Table:

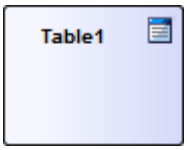
- Within the Database Builder
- On an open Data Model diagram
- Using the Browser New Element option

Add a Database Table with the Database Builder

Step	Action
1	Open the Database Builder ('Develop > Data Modeling > Database Builder').
2	Load or create a Data model.
3	Right-click on the Tables Package and select 'Add New Table'. 
4	Overtype the default name with the appropriate name for the Table, and press the Enter key.
5	Double-click on the Table element to define the Table properties.

Add a Database Table to a diagram

Step	Action
1	Create and/or open a Data Modeling diagram.
2	Drag and drop the 'Table' toolbox icon onto the diagram.

	Table This generates a new Table element: 
3	Double-click on the Table element to define the Table properties.

Database Table Columns

In a relational database, a Table column (sometimes referred to as a field) stores a single data value of a particular type in each row of the Table. Table columns can have various individual properties such as a default value or whether the field accepts Null values.

A Database Table Column is represented in the UML Data Modeling Profile as a stereotyped attribute; that is, an attribute with the <<column>> stereotype. In Enterprise Architect you define and maintain Table Columns using the purpose-designed 'Columns' page of the Database Builder, or the 'Columns and Constraints' dialog.

Create Database Table Columns

A database Table column is represented in the UML Data Modeling Profile as an attribute with the <<column>> stereotype. For a selected Table, you can review the existing columns and create new columns, on the 'Columns' page of the Database Builder or on the 'Columns and Constraints' screen.

You can define column details directly on the list of columns on the 'Columns' tab. The changes are automatically saved as you complete each field. Some fields have certain restrictions on the data you can enter, as described here. The tab also contains a 'Properties' panel and a 'Notes' field, which are populated with the existing information on the selected column. Each new column that you create is automatically assigned a set of default values and added to the bottom of the list.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table > Columns > Right-Click > Add new Column
Context Menu	In diagram, right-click on required Table | Features | Columns | Right-Click | Add new Column
Keyboard Shortcuts	Select a table | F9 | Tab Key (to set input focus on the 'Columns' tab) | Ctrl+N

Create columns in a Table

Option	Action
Name	Overtype the default name with the appropriate column name text.
Type	Click on the drop-down arrow and select the appropriate datatype for the column. The available datatypes depend on the DBMS assigned to the parent Table.
Length	(Optional) Some datatypes have a length component - for example, VARCHAR has a length that defines the number of characters that can be stored. If the datatype does not have a length component, this field is disabled. If the field is available and if you need to define a number of characters, type the value here.
Scale	(Optional) Some datatypes have a scale component - for example, DECIMAL has a scale that defines the number of decimal places that can be held. If the datatype does not have a scale component, this field is disabled. If the field is available and if you need to define a scale, type the value here.
PK	Select the checkbox if the column is part of the Primary Key for this Table.
Not Null	Select the checkbox if empty values are forbidden for this column. The checkbox is disabled if the 'PK' checkbox is selected.

Alias	If required for display and documentation purposes, type in an alternative name for the field.
Initial Value	If required, type in a value that can be used as a default value for this column.
Notes	Type in any additional information necessary to document the column. You can format the text using the Notes toolbar at the top of the field.

Column Properties

The appropriate properties for the Table's Database Management System automatically display in the 'Property' panel (expand the 'Column (<name>)' branch if they are not visible).

Property	DBMS
Autonum (Startnum Increment)	Oracle MySQL SQL Server DB2 PostgreSQL Notes: If you require an automatic numbering sequence, set this property to True and, if necessary, define the start number and increment.
Generated	DB2 Notes: Set this additional property for auto numbering in DB2, to 'By Default' or 'Always'.
NotForRep	SQLServer Notes: Set this property to True if you want to block replication.
Zerofill	MySQL Notes: Set this property to True or False to indicate if fields are zerofilled or not.
Unsigned	MySQL Notes: Set this property to True or False to indicate whether or not fields accept unsigned numbers.
LengthType	Oracle Notes: Set this property to define the character semantics as 'None', 'Byte' or 'Char'.

Delete Database Table Columns

For a selected database Table, you can review the existing columns and delete any individual column, on the 'Columns' tab of the Columns and Constraints screen.

Access

Use one of the methods outlined here to display a list of columns for a table, then select a column and delete it.

When you select the 'Delete column '<name>" option, if all validation rules are satisfied the column is immediately deleted.

Ribbon	Develop > Data Modeling > Database Builder > Click on Table > Columns > Right-click on column name > Delete column <name>
Context Menu	In diagram, right-click on required Table | Features | Columns | Right-click on column name | Delete column <name>
Keyboard Shortcuts	F9 | Use 'Up Arrow' or 'Down Arrow' to select a column | Ctrl+D

Notes

- If the deleted database Table column is involved in any constraints it will automatically be removed from them

Reorder Database Table Columns

If you have several columns defined in a database Table, you can change the order in which they are listed. The order in the list is the order in which the columns appear in the generated DDL.

Access

Use one of the methods outlined here to display a list of columns for a Table, then select a column and reposition it within the list.

Ribbon	Develop > Data Modeling > Database Builder > Click on Table
Context Menu	In diagram, right-click on required Table | Features | Columns
Keyboard Shortcuts	F9

Change the column order

Step	Action
1	In the 'Columns' tab, click on the required column name in the list.
2	Right-click and select the: • 'Move column <name> up' option (or press Ctrl+Up Arrow) to move the column up one position • 'Move column <name> down' option (or press Ctrl+Down Arrow) to move the column down one position These options have an immediate effect both in the 'Columns' tab and on a diagram.

Working with Database Table Properties

Once you have created a Database Table, you can review its properties and check that the DBMS and Owner values are correct. To display the 'Properties' dialog for a Table, either double-click on the Table name in the 'Database Builder Tables' Package or on the Table element on a diagram.

Important

A DBMS must be assigned to a Table before you can add columns in it. If you are using the Database Builder then the DBMS of the data model will be automatically applied to all new Tables; however, if you have added a Table by other means (such as working on a diagram) then this is a manual step.

Tasks

Once the Database Table properties are defined, you are ready to add columns.

Task
Set the database type for a Table - other than the Table name, the most important property to set for a Database Table is the database type.
Set the database Table Owner - For some DBMSs all Tables must be assigned an Owner/Schema; in Enterprise Architect this property is defined as a Tagged Value with the name Owner.
Set extended options - some DBMSs have extended options that are only relevant to that DBMS. These extended properties are stored as Tagged Values.

Default DBMS

Prior to creating a Physical Data Model it is advisable for you to set the default DBMS, which will be automatically applied to new database objects that you create outside of the Database Builder. You can set the default DBMS type in one of these ways:

- Select 'Start > Appearance > Preferences > Preferences > Source Code Engineering > Code Editors', then set the field 'Default Database'
- Select 'Settings > Reference Data > Settings > Database Datatypes', then select a Product Name and select the 'Set as Default' checkbox
- Set the DBMS in the second field of the Code Generation Toolbar

Set the Database Type

The most important property to set for a Database Table (after its name) is the database type or DBMS. The DBMS value selected will control how Enterprise Architect will determine:

- How the Table name will be shown (with or without an Owner)
- What set of validation rules will be applied while database modeling
- The data types that are available when creating columns,
- What set of DDL templates will be used in DDL Generation

Access

Select a Table in the Browser window or on a diagram then, using any of the methods outlined here, open the Table's 'Properties' dialog, display the 'General' tab, then display the 'Main' child tab.

Ribbon	Design > Element > Editors > Properties Dialog > General > Main
Context Menu	Right-click on the Table element | Properties | Special Action | General | Main
Keyboard Shortcuts	Shift+Enter | General | Main
Other	Double-click on the Table element |General | Main

Options

Field/Button	Action
Database	Click on the drop-down button and select the required database type from the list.
Apply	Click on the Apply button to save any pending changes.
OK	Click on the OK button to save any pending changes and close the screen.

Set Database Table Owner/Schema

For some DBMSs all Tables must be assigned an Owner/Schema. In Enterprise Architect this property is physically defined as a Tagged Value with the name Owner. However, a special properties page is provided to help you easily manage the Owner property.

Access

Select a Table in the Browser window or on a diagram then, using any of the methods outlined here, open the Table's 'Properties' dialog, display the 'General' tab and display the 'Table Detail' child tab.

Ribbon	Design > Element > Editors > Properties > << table >>
Context Menu	Right-click on the Table element | Properties |Special Action > General > Table Detail
Keyboard Shortcuts	Shift+Enter | General | Table Detail
Other	Double-click on the Table element |'General' | 'Table Detail'

Set the Database Table owner

Step	Action
1	In the 'Owner' field, type the name of the owner or schema of the Table.

Set MySQL Options

To make use of Foreign Keys in MySQL, you must declare the Database Table type as InnoDB.

Declare the Table type as InnoDB

Step	Action
1	Add a Tagged Value named *Type* to the Table.
2	Set the 'Value' field to 'InnoDB'.

Generate DDL

When you generate DDL for this Table, the Table type is included in the SQL script.
To allow for later versions of MySQL, additional Table options that can be added in the same way include:

Tag	Value (Example)
ENGINE	InnoDB
CHARACTER SET	latin1
CHARSET	latin1
COLLATE	latin1_german2_ci

Set Oracle Database Table Properties

To set additional Oracle Database Table properties, you use the Table's Tagged Values.

Set Properties

The same properties can be added to indexes and constraints, by highlighting the index or constraint Operation and adding the appropriate Tagged Values.

Step	Action
1	Add one or more Tagged Values to the Table, using the names provided in the 'Property/Tag' column of the 'Properties' Table.
2	Specify the appropriate value for each tag. Examples are provided in the 'Value' column of this Properties Table. • CACHE - NOCACHE • DBVERSION - 9.0.111 • FREELISTS - 1 • GRANT OWNER1 - SELECT • GRANT OWNER2 - DELETE, INSERT, SELECT, UPDATE • INITIAL - 65536 • INITRANS - 1 • LOGGING - LOGGING • MAXEXTENTS - 2147483645 • MAXTRANS - 255 • MINEXTENTS - 1 • MONITORING - MONITORING • OWNER - OWNER1 • PARALLEL - NOPARALLEL • PCTFREE - 10 • PCTINCREASE - 0 • PCTUSED - 0 • SYNONYMS - PUBLIC:TABLE_PUB;OWNER2:TABLE_OWNER2 • TABLESPACE - MY_TABLESPACE • TEMPORARY - YES

Database Table Constraints/Indexes

Within Enterprise Architect, Table Constraints and Indexes are modeled on the same screen; collectivity they are referred to as Constraints. Database Constraints define the conditions imposed on the behavior of a database Table. They include:

- Primary Key - uniquely identifies a record in a Table, consisting of one or more columns
- Index - improves the performance of retrieval and sort operations on Table data
- Unique Constraints - a combination of values that uniquely identify a row in the Table
- Foreign Key - a column (or collection of columns) that enforce a relationship between two Tables
- Check Constraints - enforces domain integrity by limiting the values that are accepted by a column
- Table Trigger - SQL or code automatically executed as a result of data in a Table being modified

In Enterprise Architect, you can define and maintain Table Constraints using either the purpose-designed 'Constraints/Indexes' page of the Database Builder or the Columns and Constraints screen.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes | Right-click | Add New Constraint
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes | Right-click | Add New Constraint
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Constraint

The process of creating any of these constraint types is the same and is achieved in one of the ways described here.

Create a Constraint - Using the context menu or keyboard

Step	Action
1	A new constraint is automatically created and assigned the default name *constraint n* (where *n* is a counter) and a 'Type' of 'index'. Overtype the default name with your own constraint name.
2	If necessary, in the 'Type' field click on the drop-down arrow and select the appropriate constraint type.
3	If you prefer, type an alias for the constraint, in the 'Alias' field. The 'Columns' field is read-only; it is populated with the columns that you assign to the 'Involved Columns' tab.

Create a Constraint - Overtype the template text

Step	Action
1	On the 'Constraints/Indexes' tab for the selected Table, the list of constraints ends with the template text *New Constraint*. Overtype this text with the appropriate constraint name, and press the Enter key.
2	The new constraint is automatically created and assigned the default Type of index. If necessary, in the 'Type' field click on the drop-down arrow and select the appropriate constraint type.
3	If you prefer, type an alias for the constraint, in the 'Alias' field. The 'Columns' field is read-only; it is populated with the columns that you assign to the 'Involved Columns' tab.

Assign Columns to a Constraint

The constraint types of Primary Key, Foreign Key, Index and Unique all must have at least one column assigned to them; this defines the columns that are involved in the constraint.

Step	Action
1	On the 'Constraints/Indexes' tab for the selected Table, click on the constraint to which you are assigning columns.
2	The 'Available Columns' panel lists all columns defined for the Table. For each column to assign to the constraint, right-click on the column name and select 'Assign column <name>'. The column name is transferred to the 'Assigned Columns' list.

Unassign Columns from a Constraint

Step	Action
1	On the 'Constraints/Indexes' tab for the selected Table, click on the constraint from which you are unassigning columns.
2	In the 'Assigned Columns' list, right-click on the name of the column to unassign from the constraint and select 'Unassign column <name>'. The column name is transferred to the 'Available Columns' list.

Reorder the Assigned Columns in a Constraint

If you have a number of columns in the constraint, you can re-arrange the sequence by moving a selected column name one place up or down the list at a time. To do this:

- Right-click on the column name to move and select either:
  - Move column '<name>' up (Ctrl+Up Arrow) or
  - Move column '<name>' down (Ctrl+Down Arrow)

Delete a constraint

To delete a constraint you no longer require, right-click on the constraint name in the list on the 'Constraints/Indexes' tab and select the 'Delete constraint <name>' option. If all validation rules for the given constraint type are met, the constraint is immediately removed from the repository along with all related relationships (if there are any).

Primary Keys

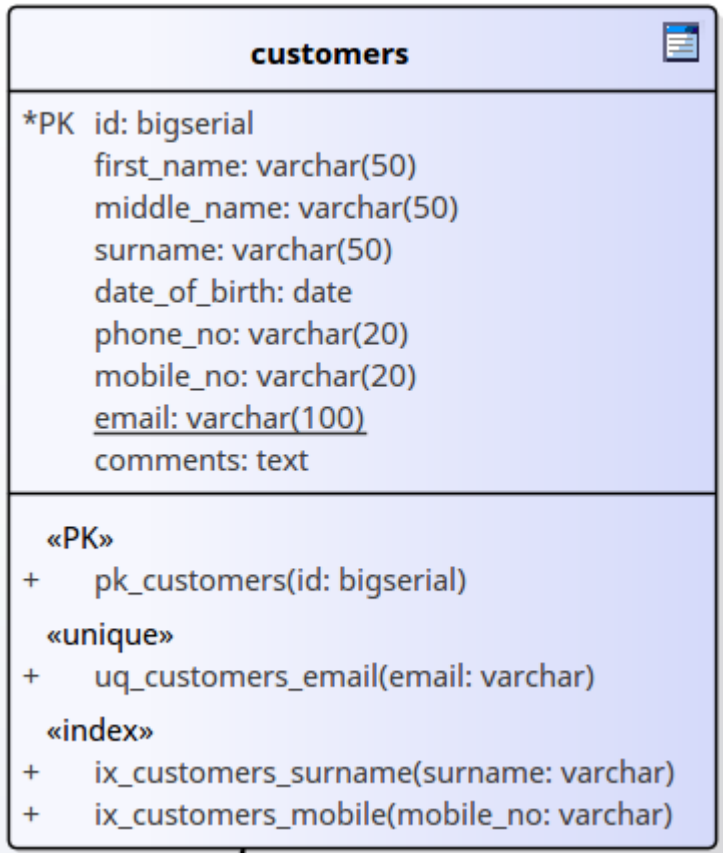
A Primary Key is a column (or set of columns) that uniquely identifies each record in a Table. A Table can have only one Primary Key. Some DBMSs support additional properties of Primary Keys, such as Clustered or Fill Factor.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes

Create a Primary Key

In Enterprise Architect you can create a Primary Key from either the 'Columns' tab or the 'Constraints/Indexes' tab. In either case, when you add a column to a Primary Key constraint, the column is automatically set to be 'Not Null'. Additionally any diagram (assuming the 'Show Qualifiers and Visibility Indicators' option is set) containing the Table element will show the 'PK' prefix against the column name. In this image, see the first column 'id: bigserial'.



Create a Primary Key - from the Columns tab

Step	Action
1	Either: • In the Database Builder, click on a Table with one or more defined columns, and click on the 'Columns' tab, or • On a diagram, click on a Table and press F9 to display the 'Columns' tab
2	For each column to include in the Primary Key, select the 'PK' checkbox. If a Primary Key constraint is not previously defined for the current Table, the system will create a new constraint using the Primary Key Name template.

Create a Primary Key - from the Constraints tab

Step	Action
1	Either: • In the Database Builder, click on a Table with one or more defined columns, and click on the 'Constraints/Indexes' tab, or • On a diagram, click on a Table and press F10 to display the 'Constraints/Indexes' tab
2	Overtype the *New Constraint* text with the Primary Key name, press the Enter key and click on the 'Type' field drop-down arrow, and select 'PK'.
3	Assign the required columns to the PK constraint.
4	Set the Primary Key's extended properties using the property panel. • Fill Factor is a numeric value between 0 and 100 • Is Clustered is a Boolean value that determines the physical order of how the data is stored; for most DBMSs the Is Clustered property defaults to True for Primary Keys

Remove columns from a Primary Key

You can remove columns from a Primary Key using either the 'Columns' tab or the 'Constraints/Indexes' tab.

Remove columns from a Primary Key - using the Columns tab

Step	Action
1	Either: • In the Database Builder, click on the Table with the Primary Key, and click on the 'Columns' tab, or • On a diagram, click on a Table and press F9 to display the 'Columns' tab

2	Against each column you want to remove from the Primary Key, deselect the 'PK' checkbox. If you have removed all columns from the Primary Key constraint and the Primary Key is no longer needed, it must be manually deleted.

Remove columns from a Primary Key - using the Constraints/Indexes tab

Step	Action
1	Either: • In the Database Builder, click on the Table with the Primary Key, and click on the 'Constraints/Indexes' tab, or • On a diagram, click on a Table and press F10 to display the 'Constraints/Indexes' tab
2	Unassign the columns on the PK constraint, as necessary.

Notes

- Warning: Enterprise Architect assumes that Primary Key constraints have at least one column assigned to them; however, Enterprise Architect does not enforce this rule during modeling
  If DDL is generated for a Table whose Primary Key has no column assigned, that DDL will be invalid

Non Clustered Primary Keys

When you create a Primary Key in some DBMSs (such as SQL Server or ASA), it is automatically created with the 'Is Clustered' property set to True. Therefore when you model a Primary Key in an Enterprise Architect data model, the same behavior occurs.

Clustered indexes provide improved performance for accessing the column(s) involved, by physically organizing the data by those columns. There can be only one clustered index per Table.

In some situations, you might be more interested in the performance of columns other than the ones assigned to the Primary Key, and therefore you would need to change the default assignment so that the Primary Key is not clustered.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes
Context Menu	In diagram or Browser window | Right-click on Table | Features | Constraints/Indexes
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes

Define Primary Key as non-clustered

Subsequently, you can model an index for the same Table as clustered.

Step	Action
1	Highlight the existing Primary Key constraint. The Primary Key properties display in the 'Property' panel.
2	For the *Is Clustered* property, in the 'Value' field click on the drop-down arrow and change the value to False.

Database Indexes

Database indexes are applied to Tables to improve the performance of data retrieval and sort operations. Multiple indexes can be defined against a Table; however, each index imposes overheads (in the form of processing time and storage) on the database server to maintain them as information is added to and deleted from the Table

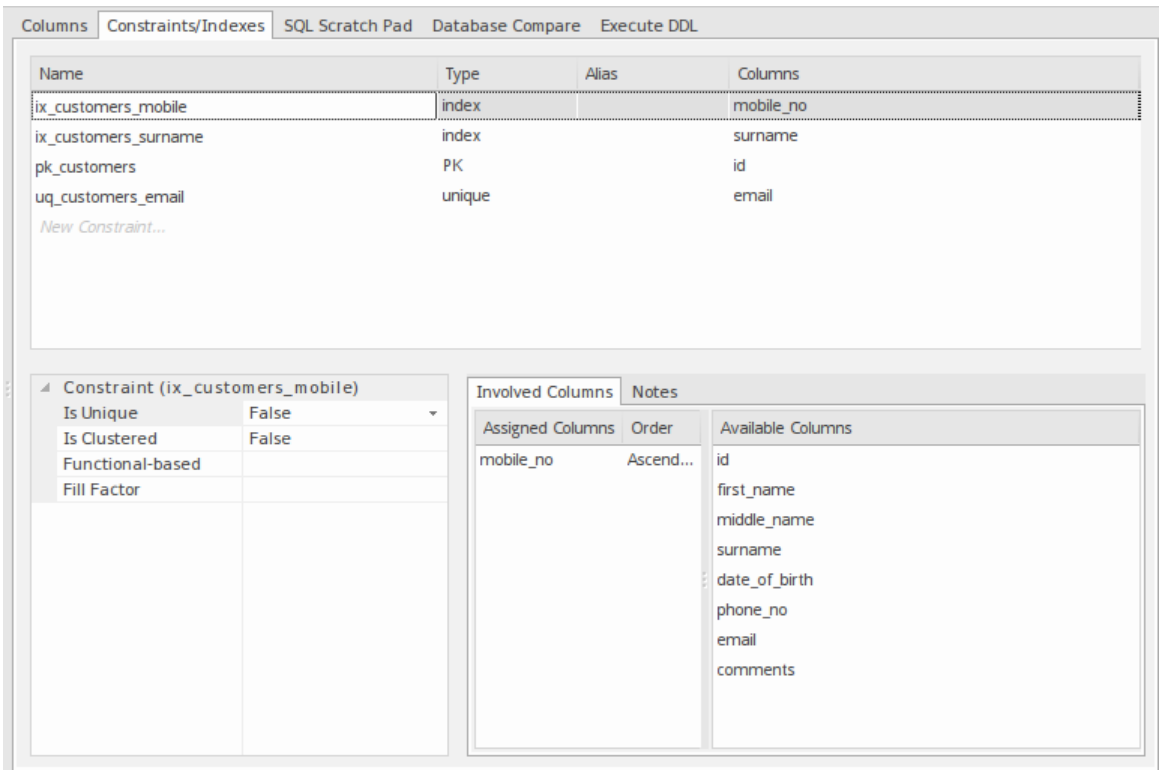
In Enterprise Architect an index is modeled as a stereotyped operation.

Some DBMSs support special types of index; Enterprise Architect defines these using additional properties such as function-based, clustered and fill-factor.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes

Work on an index



Step	Action
1	On the 'Constraints/Indexes' tab for the Table, right-click and select 'Add new constraint'.

	The new constraint is added with the default name 'constraint1' and the Type of 'index'. Overtype the name with your preferred index name.
2	Assign the appropriate columns to the Index. The 'Assigned Columns' list has an additional 'Order' field that specifies the order (Ascending or Descending) in which each assigned column is stored in the index. You can toggle the order for each column, as required. Additionally, for MySQL indexes, a 'Len' field will be visible in which you can define Partial Indexes; that is, an index that uses the leading 'n' number of characters of a text based field. The 'Len' field takes only whole number numeric values of between 0 and the column's defined length. A value of 0 (which is the default) indicates that the entire column is to be indexed.
3	In the 'Property' panel, review the settings of the extended properties that are defined for the current DBMS.

Additional Properties

Property	Description
Is Unique	(True / False) indicates whether the current index is a 'Unique Index'. A Unique Index ensures that the indexed column (or columns) does not contain duplicate values, thereby ensuring that each row has a unique value (or combination of values when the index consists of multiple columns).
Is Clustered	(True / False) indicates whether the current index is a 'Clustered Index'. With a clustered index, the rows of the table are physically stored in the same order as in the index, therefore there can be only one clustered index per table. By default a table's Primary Key is clustered. Not all DBMS's support clustered indexes, therefore the 'Is Clustered' Index property will only be visible for DBMSs that support it.
Is Bitmap	(True / False) indicates whether the current index is a 'Bitmap' index. Bitmap indexes are meant to be used on columns that have relatively few unique values (referred to as 'low cardinality' columns) and that physically consist of a bit array (commonly called bitmaps) for each unique value. Each of the arrays will have a bit for each row in the table. Consider this example: a bitmap index is created on a column called 'Gender', which has the options 'Male' or 'Female'. Physically, the index will consist of two bit arrays, one for 'Male' and one for 'Female'. The female bit array will have a 1 in each bit where the matching row has the value 'Female'. The 'Is Bitmap' and 'Is Unique' properties are mutually exclusive, and so the DDL generation will ignore the 'Is Unique' property when the 'Is Bitmap' property is True. Bitmap Indexes are only supported by Oracle; therefore, this property is only visible while modeling Oracle indexes.
Fill Factor	A numeric value between 0 and 100, that defines the percentage of available space that should be used for data. Not all DBMSs support fill factor, therefore the 'Fill Factor' index property will only be visible for DBMSs that support it.

Functional-based	A SQL statement that defines the function/statement that will be evaluated and the results indexed; for example: LOWER("field") Not all DBMSs support functional-based indexes, therefore the 'Functional-based' Index property will only be visible for DBMSs that support them, such as PostgreSQL and Oracle.
Include	Identifies a comma-separated list (CSV) of non-key Columns from the current table. Not all DBMSs support the 'Include' property on indexes, therefore this property will only be visible for DBMSs that support it.

Notes

- Warning: Enterprise Architect assumes that Indexes have at least one column assigned to them; however, Enterprise Architect does not enforce this rule during modeling
  If DDL is generated for a Table that has an Index defined without column(s) assigned, that DDL will be invalid, unless the index is functional-based
- Any columns assigned to a functional-based index are ignored

Unique Constraints

Unique Constraints enforce the 'uniqueness' of a set of fields in all rows of a Table, which means that no two rows in a Table can have the same values in the fields of a Unique Constraint. Unique Constraints are similar to Primary Keys (in that they also enforce 'uniqueness') but the main difference is that a Table can have multiple Unique Constraints defined but only one Primary Key.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes > Right-click > Add New Constraint
Context Menu	In diagram or Browser window | Right-click on Table element | Features | Constraints/Indexes
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Constraint

Step	Action
1	On the 'Constraints/Indexes' tab, a new constraint is automatically created and assigned the default constraint name and a 'Type' of index. Overtype the constraint name with a name that identifies this as a unique constraint.
2	In the 'Type' field, change the value from 'index' to 'unique'.

Notes

- Warning: Enterprise Architect assumes that Unique Constraints have at least one column assigned to them; however, Enterprise Architect does not enforce this rule during modeling
  If DDL is generated for a Table that has a unique constraint defined without column(s) assigned, that DDL will be invalid

Foreign Keys

A Foreign Key defines a column (or a collection of columns) that enforces a relationship between two Tables. It is the responsibility of the database server to enforce this relationship to ensure data integrity. The model definition of a Foreign Key consists of a parent (primary) Table containing a unique set of data that is then referred to in a child (foreign) Table.

In Enterprise Architect, a Foreign Key is modeled with two different (but related) UML components:

- A Foreign Key constraint (a UML operation with the stereotype of <<FK>>) stored on the child Table and
- An Association connector (stereotype of <<FK>>) defining the relationship between the two Tables

Create a Foreign Key

Although the definition of a Foreign Key can be complex, the Foreign Key Constraint screen simplifies the modeling of Foreign Keys. This screen is purpose-designed to help you select which constraint in the parent Table to use, and will automatically match the child Table columns to those in the parent Table that are part of the constraint. Different aspects of the process of developing a Foreign Key are described here separately for illustration, but the overall process should be a smooth transition.

A number of conditions must be met before a Foreign Key definition can be saved:

- Both Tables must have matching DBMSs defined
- The parent Table must have at least one column
- The parent Table must have a Primary Key, unique constraint or unique index defined

Create a Foreign Key - using the Database Builder

Step	Action
1	In the Database Builder tree, right-click on the child Table name and click on 'Add new Foreign Key on <table name>'. A dialog displays listing all the possible parent Tables.
2	Double-click on the required parent Table name in the list or select it and click on the OK button. The 'Foreign Key Constraint' screen displays.

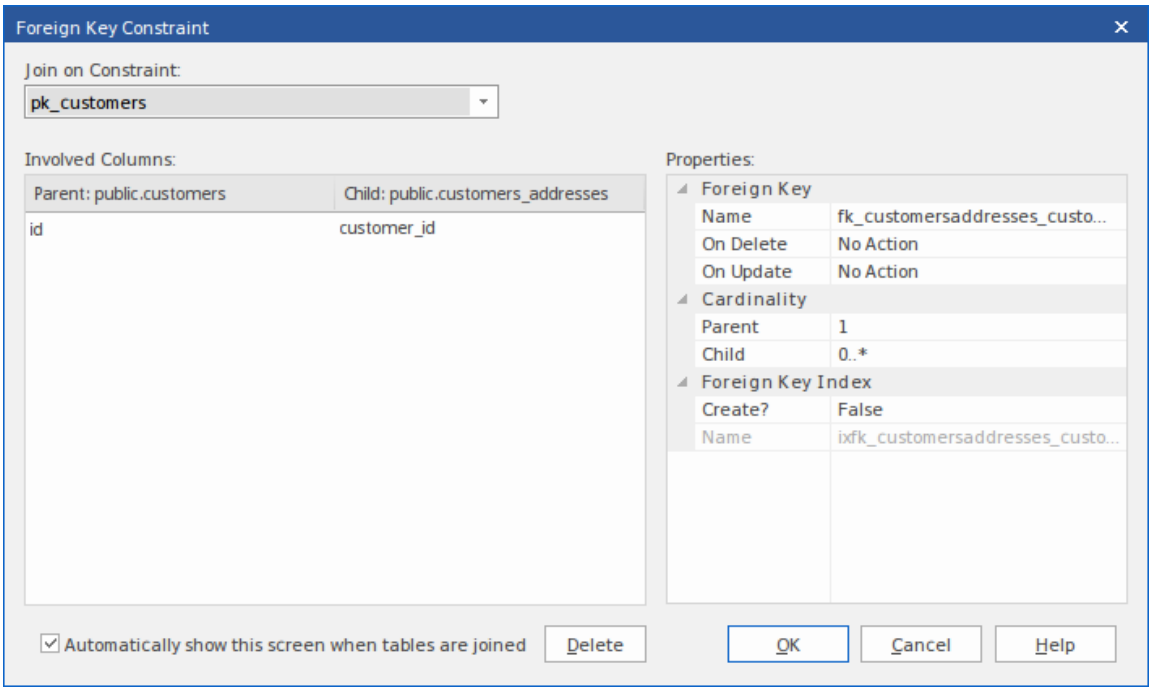
Create a Foreign Key - using a relationship on a diagram

Step	Action
1	In the Data Modeling diagram, locate the required child (Foreign Key) Table and parent (Primary Key) Table.
2	Select an Association connector in the 'Data Modeling' page of the Diagram Toolbox.
3	Click on the child Table and draw the connector to the parent Table.

4	If the Foreign Key Constraint screen has been set to display automatically when two Tables are joined, it displays now. Otherwise, either: • Double-click on the connector or • Right-click on the connector and select the 'Foreign Keys' option The Foreign Key Constraint screen displays.

The Foreign Key Constraint Screen

As an example this image shows the Foreign Key Constraint screen loaded with the details of 'fk_customersaddresses_customers' (as defined in the Example model).

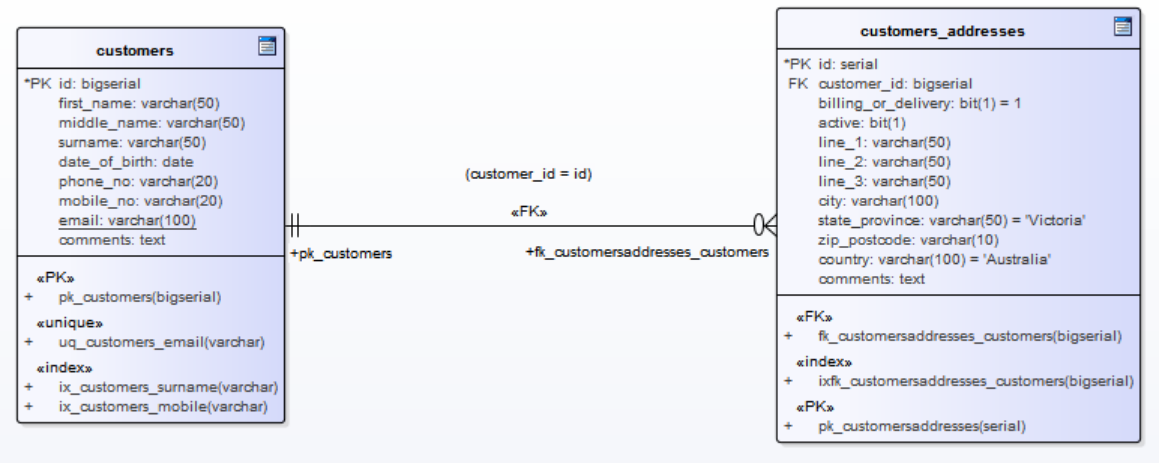


Option	Action
Join on Constraint	This combo box lists all defined constraints in the parent Table that could be used as the basis of a Foreign Key. (These constraints can be Primary Keys, Unique Constraints or Unique Indexes.) The first constraint in the list is selected by default; if this is not the constraint you want, select the correct constraint from the combo box. When you select the constraint, its columns are automatically listed in the 'Involved Columns' panel, under the 'Parent: <tablename>' column.
Involved Columns	This list is divided into two: the columns involved in the selected constraint are listed on the left, and the child columns that are going to be paired to the parent columns are listed on the right. When a constraint is selected (in the 'Join on constraint' field) the parent side is refreshed to display all columns assigned to the selected constraint. On the child side the system will automatically attempt to match each parent column to one of the same name in the child Table. If the child Table does not have a column of the same name, a new column of that name will be added to the list, flagged with (*) to indicate that a new column will be created in the Table.

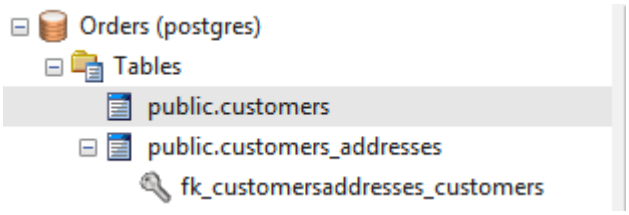
	However, if you want to force the pairing to an existing child Table column or a new column with a different name, click on the column name field and either: • Type in the replacement name, or • Select an existing column (click on the drop-down arrow and select the name from the list)
Name	This field defines the name of the Foreign Key constraint, and defaults to a name constructed by the Foreign Key Name Template. To change the name to something other than the default, simply overtype the value.
On Delete	Select the action that should be taken on the data in the child Table when data in the parent is deleted, so as to maintain referential integrity.
On Update	Select the action that should be taken on the data in the child Table when data in the parent is updated, so as to maintain referential integrity.
Parent	Click on the drop-down arrow and select the cardinality of the parent Table in the Foreign Key.
Child	Click on the drop-down arrow and select the cardinality of the child Table in the Foreign Key.
Create?	If you want to create a Foreign Key Index at the same time as the Foreign Key, set this property to True. The name of the Foreign Key Index is controlled by the Foreign Key Index template, and the generated name is shown in the 'Name' field underneath the 'Create?' field.
Automatically show this screen when tables are joined	(For diagrammatic modeling) Select this checkbox to automatically display this screen whenever an Association is created between two Tables.
Delete	Click on this button to delete the currently selected existing (saved) Foreign Key. A prompt is displayed to confirm the deletion (and the deletion of the Foreign Key Index, if one exists) - click on the Yes button. Deleting a Foreign Key leaves an Association connector in place, which you can either edit or delete (right-click and select 'Delete association: to <Table name>').
OK	Click on this button to save the Foreign Key.

Examples

This example shows simple Foreign Keys in a diagram:



The same Foreign Key will be shown in the Database Builder's tree as a child node under the Table 'customers.addresses'.



Check Constraints

A Check Constraint enforces domain integrity by limiting the values that are accepted by a column.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes > Right-click > Add New Constraint
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes | Right-click | Add New Constraint
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Constraint

Step	Action
1	On the 'Constraints/Indexes' tab of the Columns and Constraints screen, a new constraint is automatically created and assigned the default constraint name and a 'Type' of index. Overtype the constraint name with a name that identifies the constraint as a check constraint, such as 'CHK_ColumnName' (the CHK_ prefix is optional).
2	In the 'Type' field, change the value from 'index' to 'check'.
3	In the 'Properties' panel for the Condition property, type the SQL statement that will be used as the Check Condition; for example, column1 < 1000. If the condition is long, click on the  button to display a SQL editor (with syntax highlighting).

Delete a Check Constraint

If you do not want to keep a check constraint, either:

- Right-click on it in the list and select 'Delete constraint <name>', or
- Click on the item and press Ctrl+D

The constraint is immediately deleted.

Notes

- Any columns assigned to a check constraint are ignored

Table Triggers

A Table trigger is SQL or code that is automatically executed as a result of data being modified in a database Table. Triggers are highly customizable and can be used in many different ways; for example, they could be used to stop certain database activities from being performed during business hours, or to provide validation or perform deletions in secondary Tables when a record in the primary Table is deleted.

In Enterprise Architect, a Table trigger is modeled as a stereotyped operation and managed using the Table's 'Constraints' screen.

Access

Ribbon	Develop > Data Modeling > Database Builder > Click on Table name > Constraints/Indexes | Right-click | Add New Constraint
Context Menu	In diagram | Right-click on Table | Features | Constraints/Indexes | Right-click | Add New Constraint
Keyboard Shortcuts	Click on Table: F9 > Constraints/Indexes: Ctrl+N

Create a Table Trigger

Step	Action
1	On the 'Constraints/Indexes' tab, a new constraint is automatically created and assigned the default constraint name and a 'Type' of index. Overtype the constraint name with a name that identifies the constraint as a trigger, such as TRG_OnCustomerUpdate. (The TRG_ prefix is optional.)
2	In the 'Type' field, change the value from 'index' to 'trigger'.
3	In the 'Properties' panel for the Statement property, type in the complete SQL statement (including CREATE TRIGGER) that will define the Trigger. If the condition is long, click on the  button to display a SQL editor (with syntax highlighting).
4	The properties Trigger Time and Trigger Event are currently information-only values and are not used in DDL generation.

Delete a Table Trigger

If you do not want to keep a trigger, either:

- Right-click on it in the list and select 'Delete constraint <name>', or
- Click on the item and press Ctrl+D

The trigger is immediately deleted.

Notes

- Any columns assigned to table triggers are ignored

Database Views

A Database View represents the results of a pre-defined query. Unlike a Table, a View is dynamically derived from data in one or more Tables (or other Views). Enterprise Architect supports the definition of Views both with and without this statement:

"Create View {viewName} As" statement

The system will automatically add it dynamically (if missing) whenever DDL generation is performed. The advantage of not defining this statement is that when a view object is renamed the 'View Definition' property does not have to be manually updated.

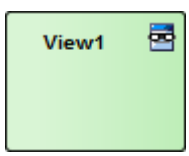
You can create a Database View either:

- Within the Database Builder or
- By dragging the 'View' icon from the Data Modeling Toolbox onto a diagram

Add a Database View with the Database Builder

Step	Action
1	Open the Database Builder.
2	Load or create a Data model.
3	Right-click on the 'Views' Package and select 'Add New View'.
4	Overtype the default name with the appropriate name for the View, and press the Enter key.
5	Double-click on the new View, or right-click on it and select 'SQL Object Properties'. The 'SQL Object Editor' dialog displays.

Add a Database View to a diagram

Step	Action
1	Open your Data Modeling diagram and, if necessary, display the 'Data Modeling' page of the Diagram Toolbox (click on  to display the 'Find Toolbox Item' dialog and specify 'Data Modeling').
2	Drag the 'View' icon onto the diagram.  This generates the View element: 
3	Right-click on the new View element and select 'SQL Object Properties'.

	The 'SQL Object Editor' dialog displays.

SQL Object Editor

The 'SQL Object Editor' dialog is shared by a number of SQL-based database objects (Views, Procedures, Functions and Sequences); it helps the data modeler manage the various properties of the SQL-based object.

Option	Action
Database	If it has already been set, the default database type displays. If the default has not been set, or you want to change the database type for this View, click on the drop-down arrow and select the target DBMS to model.
Dependencies	A list of objects that the current object depends on. The 'Dependencies' list shows: • Each Depends connector between this View and another Table or View • Any object names (specified as a CSV list) in the 'parents' Tagged Values
Notes	If necessary, type in a comment on the current View.
Definition	Type the full SQL View definition. For releases of Enterprise Architect up to 12.1 (Build 1227), this must include the CREATE_VIEW syntax as appropriate for the target DBMS (for later versions this is not needed). For example: CREATE VIEW 'MyViewName' AS [view definition] The code editor provides Intelli-sense for basic SQL keywords, functions and names of all objects in the current data model.

Database Procedures

Database Procedures (sometimes referred to as Stored Procedures or Procs) are subroutines that can contain one or more SQL statements that perform a specific task. They can be used for data validation, access control, or to reduce network traffic between clients and the DBMS servers. Extensive and complex business logic can be embedded into the subroutine, thereby offering better performance.

Database Procedures are similar to Database Functions. The major difference is the way in which they are invoked - Database Functions can be used in the same way as for any other expression within SQL statements, whereas Database Procedures must be invoked using the CALL or EXEC statement, depending on the DBMS.

In Enterprise Architect, Database Procedures can be modeled in one of two ways:

- As individual objects (the default method) or
- As operations in a container

Functionally the two methods result in the same DDL being produced. The main difference is visual - by having several Operations in one container, you have fewer elements and less clutter on the diagram.

Individual objects

Database Procedures modeled as individual objects are UML Classes with the stereotype ?procedure?; you create these either:

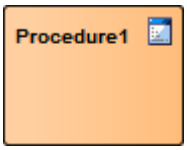
- Within the Database Builder or
- By dragging the 'Procedure' icon from the Data Modeling Toolbox onto a diagram

Add a Database Procedure using the Database Builder

Step	Action
1	Open the Database Builder.
2	Load or create a Data model.
3	Right-click on the Procedures Package and select 'Add New Procedure'.
4	Overtype the default name with the appropriate name for the Procedure, and press the Enter key.
5	Double-click on the new Procedure, or right-click on it and select 'SQL Object Properties'. The SQL Object Editor screen displays.

Add a Database Procedure to a diagram

Step	Action
1	Open your Data Modeling diagram and, if necessary, display the 'Data Modeling' page of the Diagram Toolbox (click on  to display the 'Find Toolbox Item' dialog and specify 'Data Modeling').

2	Drag the 'Procedure' icon onto the diagram.  This generates the Procedure element: 
3	Right-click on the new Procedure element and select 'SQL Object Properties'. The SQL Object Editor screen displays.

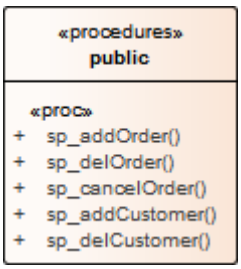
SQL Object Editor

The 'SQL Object Editor' dialog is shared by a number of SQL-based database objects (Views, Procedures and Functions); it helps you to manage the various properties of the SQL-based object.

Option	Action
Database	If it has already been set, the default database type displays. If the default has not been set, or you want to change the database type for this Procedure, click on the drop-down arrow and select the target DBMS to model.
Notes	If necessary, type in a comment on the current Procedure.
Definition	Type the full SQL Procedure definition, including the CREATE PROCEDURE syntax. The code editor provides Intelli-sense for basic SQL keywords, functions and names of all objects in the current data model.

Operations in a Container

Database Procedures modeled as operations have a container object, this being a UML Class with the stereotype «procedures» (with an 's' on the end). Each Database Procedure is an operation with the stereotype «proc». The system provides a dedicated Maintenance window through which you can easily manage the Database Procedures defined as operations.



Database Functions

Database Functions provide you with a mechanism to extend the functionality of the database server. A Database Function is a routine that accepts parameters, performs an action (such as a complex calculation) and returns the result of that action as a value. Depending on the Function, the return value can be either a single value or a result set.

Once created, a Database Function can be used as an expression in an SQL statement.

In Enterprise Architect, Database Functions can be modeled in one of two ways:

- As individual objects (the default method) or
- As Operations in a container

Functionally the two methods result in the same DDL being produced. The main difference is visual - by having several Operations in one container, you have fewer elements and less clutter on the diagram.

Individual objects

Database Functions modeled as individual objects are UML Classes with the stereotype ?function?; you create these either:

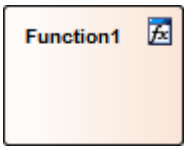
- Within the Database Builder or
- By dragging the Function icon from the Data Modeling Toolbox onto a diagram

Add a Database Function using the Database Builder

Step	Action
1	Open the Database Builder.
2	Load or create a Data model.
3	Right-click on the Functions Package and select 'Add New Function'.
4	Overtype the default name with the appropriate name for the Function, and press the Enter key.
5	Double-click on the new Function, or right-click on it and select 'SQL Object Properties'. The SQL Object Editor screen displays.

Add a Database Function to a diagram

Step	Action
1	Open your Data Modeling diagram and, if necessary, display the 'Data Modeling' page of the Diagram Toolbox (click on  to display the 'Find Toolbox Item' dialog and specify 'Data Modeling').
2	Drag the 'Function' icon onto the diagram.

	Function This generates the Function element: 
3	Right-click on the new Function element and select 'SQL Object Properties'. The SQL Object Editor screen displays.

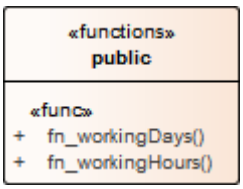
SQL Object Editor

The 'SQL Object Editor' dialog is shared by a number of SQL-based database objects (Views, Procedures and Functions); it helps you to manage the various properties of the SQL-based object.

Option	Action
Database	If it has already been set, the default database type displays. If the default has not been set, or you want to change the database type for this Function, click on the drop-down arrow and select the target DBMS to model.
Notes	If necessary, type in a comment on the current Function.
Definition	Type the full SQL Function definition including the CREATE FUNCTION syntax. The code editor provides Intelli-sense for basic SQL keywords, functions and names of all objects in the current data model.

Operations in a Container

Database Functions modeled as operations have a container object, this being a UML Class with the stereotype «functions» (with an 's' on the end). Each Function is an operation with the stereotype «func». The system provides a dedicated Maintenance window through which you can easily manage the Database Functions stored as operations.



Database Sequences

Sequences are a feature that some DBMS products implement to provide users with a mechanism to generate unique values - the Sequence ensures that each call to it returns a unique value. This is particularly important when the Sequence's result is used as a Primary Key. These can be generated with a schema for loading onto the DBMS server.

Sequences are provided so that database users are not forced to implement their own unique value generator. Not all DBMS products support Sequences; those that do not instead provide functionality for columns to be initialized with an incrementing value.

In Enterprise Architect, Sequences can be modeled in one of two ways:

- As individual objects (the default method) or
- As Operations in a container

Functionally the two methods result in the same DDL being produced. The main difference is visual - by having several Operations in one container, you have fewer elements and less clutter on the diagram.

Individual objects

Sequences modeled as individual objects are UML Classes with the stereotype ?dbsequence?; you create these either:

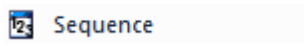
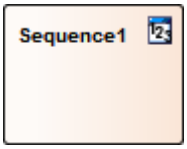
- Within the Database Builder or
- By dragging the 'Sequence' icon from the 'Data Modeling' Toolbox pages onto a diagram

Add a Database Sequence using the Database Builder

Step	Action
1	Open the Database Builder.
2	Load or create a Data model.
3	Right-click on the Sequences Package and select 'Add New Sequence'.
4	Overtype the default name with the appropriate name for the Sequence, and press the Enter key.
5	Double-click on the new Sequence, or right-click on it and select 'SQL Object Properties'. The 'SQL Object Editor' dialog displays.

Add a Database Sequence to a diagram

Step	Action
1	Open your Data Modeling diagram and, if necessary, display the 'Data Modeling' page of the Diagram Toolbox (click on  to display the 'Find Toolbox Item' dialog and specify 'Data Modeling').

2	Drag the 'Sequence' icon onto the diagram.  This generates the Sequence element: 
3	Right-click on the new Sequence element and select 'SQL Object Properties'. The 'SQL Object Editor' dialog displays.

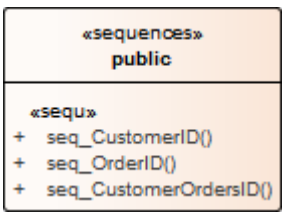
SQL Object Editor

The 'SQL Object Editor' dialog is shared by a number of SQL-based database objects (Views, Procedures and Functions); it helps you to manage the various properties of the SQL-based object.

Option	Action
Database	If it has already been set, the default database type displays. If the default has not been set, or you want to change the database type for this Sequence, click on the drop-down arrow and select the target DBMS to model.
Notes	If necessary, type in a comment on the current Sequence.
Definition	Type the full SQL Sequence definition including the CREATE SEQUENCE syntax. The code editor provides Intelli-sense for basic SQL keywords, functions and names of all objects in the current data model.

Operations in a Container

Database Sequences modeled as operations have a container object, this being a UML Class with the stereotype «sequences» (with an 's' on the end). Each Sequence is an operation with the stereotype «sequ». The system provides a dedicated Maintenance window through which the modeler can easily manage the Sequences defined as operations.



Database SQL Queries

An SQL Query object provides a convenient mechanism for storing an SQL Statement in the repository, for repeated execution on live database(s).

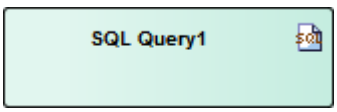
An SQL Query element is represented in the UML Data Modeling Profile as an Artifact element with the stereotype <<sqlquery>>. You can create these elements either:

- Within the Database Builder or
- By dragging the 'SQL Query' icon from the 'Data Modeling' Toolbox pages onto a diagram

Add a Database SQL Query using the Database Builder

Step	Action
1	Open the Database Builder.
2	Load or create a Data model.
3	Right-click on the Queries Package and select 'Add New SQL Query'.
4	Overtype the default name with the appropriate name for the Query, and press the Enter key.
5	Right-click on the new element and select 'Edit'. The 'SQL Scratch Pad' tab displays, on which you can create the SQL Query statement.
6	When you have finished the SQL statement, click on the Save to SQL Query button in the toolbar to save the changes to the query element.

Add a Database Function to a diagram

Step	Action
1	Open your Data Modeling diagram and, if necessary, display the 'Data Modeling' page of the Diagram Toolbox (click on  to display the 'Find Toolbox Item' dialog and specify 'Data Modeling').
2	Drag the 'SQL Query' icon onto the diagram.  This generates the SQL Query Artifact element: 
3	Double-click on the new element and update the element name and other properties as necessary. To edit the element's SQL statement, access the Database Builder, click on the element in the Queries Package and edit the Query on the 'SQL Scratch Pad' tab.

Create Operation Containers

Whilst the default method of modeling Database Functions, Procedures and Sequences is to create them as individual elements, you can also represent a number of each type of structure as operations of a container Class. You add a stereotype to the Class, which specifies:

- The type of data structure the Class will contain
- The stereotype that will be automatically assigned to each operation created in the Class (for a given data structure, the operations can only be of one stereotype)

Access

Toolbox	Drag the 'Class' icon onto the diagram

Create the Container Class

Step	Action
1	Right-click on the Class element on the diagram and select the 'Design > Element > Editors > Properties Dialog' option. The element 'Properties' dialog displays, showing the 'General' tab.
2	In the 'Name' field, type an appropriate name for the container.
3	In the 'Stereotype' field (in the table at the right edge of the dialog) type: • 'functions' for a Database Function container • 'procedures' for a Stored Procedure container • 'sequences' for a Sequence container The 's' at the end of the stereotype name is important.
4	Click on the OK button to save the setting and close the dialog.

Create database structures as operations of the Class

Step	Action
1	Click on the Class element on the diagram and press F10. The 'Database <Structure> container: <Classname>' dialog displays.
2	Right-click in the 'Functions' ('Procedures' or 'Sequences') list and select 'Add New <structure>'.

3	In the 'Name' field, type an appropriate name for the operation, such as: • fn_WorkDays • sp_AddOrder or • seq_AddressID
4	In the 'Notes' field type any supporting comments or explanation of the operation. In the 'Function definition' field (or 'Procedure definition', or 'Sequence definition') type the appropriate text.
5	Repeat steps 2 to 4 until you have defined all the operations you require.
6	Click on the list and then on the Close button to close the dialog and show the operations within the Class on the diagram and in the Browser window.

Oracle Packages

Oracle Packages are database objects that are unique to the Oracle DBMS. They are containers that group logically-related objects into a single definition. Packages have two parts - a specification and a body. The:

- Specification section declares the various components
- Body section provides the full definitions of the components

The Package components can consist of Types, Variables, Constants, Exceptions, Cursors and subprograms.

In Enterprise Architect, an Oracle Package is modeled as a UML Class with a stereotype of <<package>>. It has two operations:

- Specification
- Body

For each of these operations the complete SQL syntax is contained in the 'Initial Code' field.

Create an Oracle Package

Step	Action
1	Add a Class element to your data model.
2	Open the Properties window for the element and, in the 'Stereotype' field, type the value 'Package'.
3	Click on the element and press F10, to display the Features window at the 'Operations' page. For the Package specification, press Ctrl+N and create an operation with the name 'Specification' and with no return type.
4	The Properties window displays the properties of the operation; click on the 'Code' tab and type the entire Package specification into the text panel.
5	Return to the Features window at the 'Operations' page and, for the Package body, press Ctrl+N and create an operation with the name 'Body' and no return type.
6	On the Properties window, click on the 'Code' tab and type the entire Package body code into the text panel.

Database Connections

A Database Connection object provides a convenient way of storing the connection details of a live database. Enterprise Architect supports the definition of a number of different connection types:

- MS Access
- Firebird
- SQLite (introduced in Enterprise Architect v16)
- Native Connection (introduced in Enterprise Architect v16), and
- ODBC

For file based connections (MS Access, Firebird and SQLite) you only have to specify the full path to the database files. For Native connections you will be prompted for the connection details of a database server. For connections of type ODBC you are prompted to select from the list of pre-defined ODBC DSNs on your machine.

Create a Database Connection Element

A Database Connection element is represented in the UML Data Modeling Profile as an Artifact element with the stereotype <<database connection>>. You create these either:

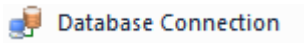
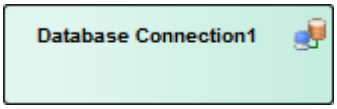
- Within the Database Builder or
- By dragging the 'Database Connection' icon from the 'Data Modeling' Toolbox pages onto a diagram

Add a Database Connection Using the Database Builder

Step	Action
1	Open the Database Builder.
2	Load or create a Data model.
3	Right-click on the Connections Package and select 'Add New DB Connection'.
4	Overtype the default name with the appropriate name for the Connection, and press the Enter key.
5	Double-click on the new Connection, or right-click on it and select 'DB Connection Properties'. The 'Database Connection Properties' dialog displays.

Add a Database Connection to a Diagram

Step	Action
1	Open your Data Modeling diagram and, if necessary, display the 'Data Modeling' page of the Diagram Toolbox (click on  to display the 'Find Toolbox Item' dialog and specify 'Data Modeling').

2	Drag the 'Database Connection' icon onto the diagram.  This generates the Database Connection element. 
3	Double-click on the new element. The 'Database Connection Properties' dialog displays.

Database Connection Properties

Option	Action
DBMS Type	Click on the radio button for the appropriate type: • MS Access file based database • Firebird file based database • SQLite file based database • Direct Native connection, or • ODBC based database The 'Save Password?' checkbox is only enabled for ODBC connection types, and indicates if Enterprise Architect should store the password for the selected ODBC DSN. The checkbox defaults to selected; that is, passwords are saved. While all connection passwords are encrypted before being saved, there can be occasions when data modelers want to restrict access to only users that have the required permissions.
Filename/DSN	If you have selected a 'DBMS Type' of MS Access or Firebird, type in or browse for the location and name of a physical file. If the file does not already exist it will be created. If you have selected a 'DBMS Type' of ODBC, type in or select a defined ODBC DSN. Depending on the DBMS, you might be prompted for other details such as server, connection user ID and password.
Other Schemas	This field acts as a schema filter to limit the number of objects returned by enquiries made against the ODBC connection. Entering a value in this field is particularly important for Oracle databases to reduce the time it takes for making connections to the database, due to the large number of system objects. If you need to enter multiple schemas to be filtered on, separate them with commas.
OK	Click on this button to save the changes you have made.

Delete Connection

If a connection is no longer required, you can delete it as for any other element from the Database Builder, the Browser

window or a diagram. Right-click on the element and select the corresponding 'Delete <element name>' option.

Notes

- It is advisable that when working in a team environment (that is, multiple users sharing a single Enterprise Architect repository) all ODBC based Database Connection objects are defined as 'DSN-less' so that the Database Connection object contains all necessary details and can therefore be shared between all users, although a Native Connection does this and is easier to setup
- The DBMS type of a Database Connection object cannot be changed once the initial selection has been saved

Manage DBMS Options

Using the 'Manage DBMS Options' dialog, you can quickly change the DBMS Type and/or Owner of an individual database object or several objects within an individual Package or Package hierarchy. You can also create bulk Foreign Key Indexes on all Foreign Keys that do not already have an index.

Access

Ribbon	Design > Package > Manage > DBMS Options Develop > Data Modeling > Database Builder > Right-click on the required database | Load | right-click on the root node | Manage DBMS Options

Options

Option	Action
Package	Displays the name of the Package in the Browser window that you are currently working on. If necessary, click on the  button and select a different Package using the Navigator window (a version of the 'Find Package' dialog).
Include Objects in Child Packages	Select this checkbox to include all the database objects in all sub-Packages. Selecting or deselecting this control will immediately refresh the list of objects.
List of Objects	This list control will display the names of all objects in the current Package (or Package hierarchy) along with its allocated DBMS and owner. By default every object has its checkbox selected whenever the list is loaded or refreshed.
All	Click on this button to select all deselected checkboxes in the 'List of Objects'.
None	Click on this button to deselect all selected checkboxes in the 'List of Objects'.
Change DBMS	Select this checkbox if you want to change the assigned DBMS of objects in the Package. Provide values for the 'Current DBMS' and 'New DBMS' fields in order to continue. The 'Current DBMS' drop-down list includes the option '<All>', which changes several different DBMS values all to the new value. Note: When performing this function, the data types of all Table columns are automatically converted to the closest match for the selected DBMS; therefore, you should perform a manual review of the data types after running the process.
Change Owner	Select this checkbox if you want to change the Owner of the selected objects in the 'List of Objects'. Specify the current Owner in the 'Current Owner' field in order to continue. Leaving the 'New Owner' field blank will remove the Owner property of all selected objects.
Create Indexes on Foreign Keys	Select this checkbox to create an index on all Foreign Keys in the Package, where one does not already exist.

OK	Click on this button to start the update process. The button is disabled unless at least one object in the list and one of the update options are selected.

Data Types

Every Table column that you define in your data model has a data type assigned that specifies the type of information that can be stored by the column. The available datatypes for a column are dependent on the selected DBMS for the Table, because each DBMS supports its own list of datatypes. Whilst each DBMS supports the same basic types, such as string, whole or decimal numbers, each DBMS calls them by different names and have different properties.

Each Enterprise Architect repository contains the definitions of the core datatypes for a number of standard DBMS products. However, since data types vary from one DBMS product to another, and from one version of a product to another, Enterprise Architect provides you with tools to:

- Define new data types for a new version of your DBMS product
- Define data types for a new, non-standard database product
- Automatically convert data types from one defined DBMS product to another
- Import and export datatypes between repositories

Map Data Types Between DBMS Products

Whilst modeling physical data models provides a great deal of detail about all Tables and their columns, this level of detail does make it harder to change the target technology or platform. For example, after reverse engineering your database into a physical data model, you must remap the data types before generating the schema for the new DBMS product.

Enterprise Architect provides a set of default mappings for standard, supported DBMS products, to help you automate the conversion process.

However, you might want to customize the default mappings to suit your specific project requirements, or where the mapping of one data type to another is not currently defined. For example, in your migration from one DBMS platform to another, one of the platforms might be non-standard or otherwise not supported by Enterprise Architect.

Access

Ribbon	Settings > Reference Data > Settings > Database Datatypes : Datatype Map

Database Data Types Mapping

Repeat this process for each of the data types to map.

Once you are satisfied with the data type mappings, you can convert either individual Tables or an entire Package of Tables to the new target DBMS product.

Field/Button	Action
From Product Name	Click on the drop-down arrow and select the DBMS product to map data types from.
Defined Datatypes for Databases	Displays all the defined data types for the product and, where appropriate, their sizes and values. Click on the data type to map - this must have a defined size unit and value. The 'Datatype' and 'Common Type' fields under the 'From Product Name' field display this data type.
To Product Name	Click on the drop-down arrow and select the DBMS product to map data types to. The 'Datatype' and 'Common Type' fields under this field display the values corresponding to those in the fields for the 'From' product.
Size	Click on the radio button for the appropriate size unit and type the default values in the corresponding data fields.
Save	Click on this button to save the mapping.

DBMS Product Conversion for a Package

Using the DBMS Package mapper, you can automatically convert a Package of database Tables from one supported DBMS type to another supported DBMS type. You can also change the DBMS type for individual Tables.

If one of the DBMS types is non-standard or otherwise not supported by Enterprise Architect, you should check that the mapping of datatypes from one DBMS type to the other has been defined.

Access

Ribbon	Design > Package > Manage > DBMS Options Develop > Data Modeling > Database Builder > Right-click on the required database | Load | right-click on the root node | Manage DBMS Options

Map the DBMS data types of a Package to the data types of another DBMS

Field/Button	Action
Include Objects in Child Packages	If there are objects in child Packages that also require changing, select the checkbox.
Change DBMS	Select the checkbox.
Current DBMS	Click on the drop-down arrow and select the current DBMS.
New DBMS	Click on the drop-down arrow and select the target DBMS.
OK	Click on this button to map all Tables in the selected Packages to the new DBMS.

Data Type Conversion For a Table

Once a database schema has been set up on an Enterprise Architect diagram (either by importing through ODBC or manually setting up the Tables), the DBMS can be changed to another type and the column datatypes are mapped accordingly for each Table.

You might use this procedure if you have copied a small number of Tables into the project from elsewhere, but if you have many Tables you can also convert all of them at once within their parent Package.

If one of the DBMS types is non-standard or otherwise not supported by Enterprise Architect, you should check that the mapping of datatypes from one DBMS type to the other has been defined.

Map the DBMS type of a Table to another DBMS type

Step	Action
1	Double-click on the Table element in a diagram. The Table 'Properties' dialog displays, with the 'Database' field showing the current DBMS for this Table.
2	To map the datatypes to another DBMS, click on the 'Database' drop-down arrow and select the target DBMS.
3	Click on the Apply button. The datatypes are converted to match those of the new DBMS, and these are reflected in any DDL generated from this Table.

Database Datatypes

Using Enterprise Architect's 'Database Datatypes' dialog, you can add to the set of data types that are available for a particular DBMS. You can:

- Identify the DBMS in use and, if required, set this as the model default
- Include any new data types that are supported by later versions of the DBMS and not yet included with Enterprise Architect
- Remove any previously-added data types that are no longer relevant
- Add a new DBMS product and its built-in data types if, for example, you want to create a physical data model for a DBMS product that is not yet supported natively by Enterprise Architect

Access

Ribbon	Settings > Reference Data > Settings > Database Datatypes or Develop > Data Modeling > Datatypes

Manage Datatypes

You can transport these database data types between Enterprise Architect models using the 'Export Reference Data' and 'Import Reference Data' options.

Field/Button	Action
Product Name	Click on the drop-down arrow and select an existing DBMS. Once a product is selected, all defined data types will be shown in the 'Defined Datatypes for Databases' list.
Add Product	If your DBMS is not listed, click on this button to add it. An 'Input' prompt displays, in which you type the DBMS name; click on the OK button to add the name to the drop-down list.
Set as Default	Select the checkbox to set the selected DBMS as the default for your database engineering and modeling. Once you set the default database, when you create any new Table elements the database type is automatically pre-set to this default. You can also set the default database type in the second data entry field of the Code Generation toolbar.
New	Click on this button to clear the data type fields on the dialog so that you can define another data type.
Datatype	Type a name for the data type.
Size	Select the appropriate radio button for the required size and, if appropriate, specify the default and maximum values: • None – for data types without a size component, such as INT

	• Length – for data types that require a single size that defines the Length, such as VARCHAR(10) • Precision & Scale – for data types that require two numeric values, such as DECIMAL(18,2)
Common Type	Click on the drop-down arrow and select the generic name of each data type. This is used when a Table's DBMS is changed.
Save	Click on the button to immediately save your data type to the repository (and add it to the 'Defined Datatypes for Databases' list).
Defined Datatypes for Databases	This panel lists the data types currently defined for the selected DBMS, either system-supplied or user-defined.
Delete	Select a data type in the 'Defined Datatypes for Databases' list and click on this button to remove the data type.
Datatype Map	If you have changed the DBMS or technology for which you have defined the data types from or to an unsupported DBMS type, click on this button to define how to automatically remap the data types to your new DBMS or technology.

MySQL Data Types

MySQL supports the ENUM and SET data types, which must be added to your Enterprise Architect model before you can use them as the types for columns.

Access

Ribbon	Settings > Reference Data > Settings > Database Datatypes

Add the ENUM and SET data types for MySQL

When using these data types later in a column's 'Initial' field, type the values as a comma-separated list, in the format:

('one','two','three')

If one value is the default, use the format:

('one','two','three') default 'three'

Step	Action
1	The 'Database Datatypes' dialog displays.
2	In the 'Product Name' field select 'MySQL'.
3	Add the data types ENUM and SET.

Oracle Data Types

The Oracle data types NUMBER and VARCHAR have additional properties that you can model.

Access

Ribbon	Settings > Reference Data > Settings > Database Datatypes

Data Types

Data Type	Detail
NUMBER	The NUMBER data type requires precision and scale properties. The 'Precision' and 'Scale' fields are displayed on the 'Attributes' page of the Features window when the data type is set to NUMBER; if you enter information into these fields, it is displayed on your diagrams. For example: create NUMBER by setting 'Precision' = 0 and 'Scale' = 0 create NUMBER(8) by setting 'Precision' = 8 and 'Scale' = 0 create NUMBER(8,2) by setting 'Precision' = 8 and 'Scale' = 2
VARCHAR	Oracle VARCHAR2(15 CHAR) and VARCHAR2(50 BYTE) data types can be created by adding the Tagged Value LengthType with the value CHAR or BYTE.

Data Modeling Settings

Enterprise Architect provides data modeling settings that can be used to configure the way database systems are modeled in Enterprise Architect. These include the ability to define the data modeling language, which determines the way that connectors are displayed, and settings to configure the naming of Primary Keys, Foreign Keys and Indexes. The settings are global and will affect any Enterprise Architect repository.

Access

Ribbon	Start > Appearance > Preferences > Preferences > Source Code Engineering > Code Editors > DDL

DDL Editor

In this field you browse for the full execution file path and name of an external program that Enterprise Architect should use to open files that are created by its Generate DDL functionality. If you leave this field empty, Enterprise Architect uses the default code editor.

Default Database

In this field you select the DBMS that will be automatically assigned to database objects that are created outside a Data Model workspace (see the *Create a Data Model from Model Pattern* Help topic).

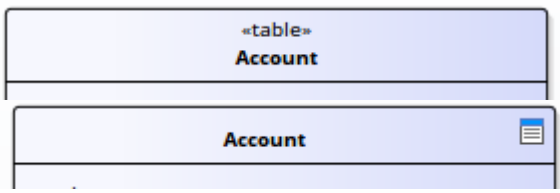
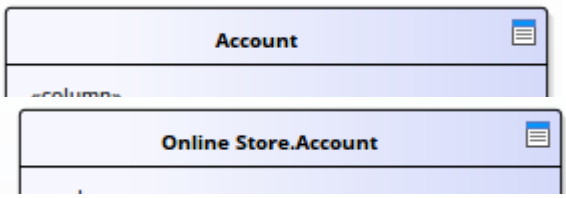
MySQL Storage

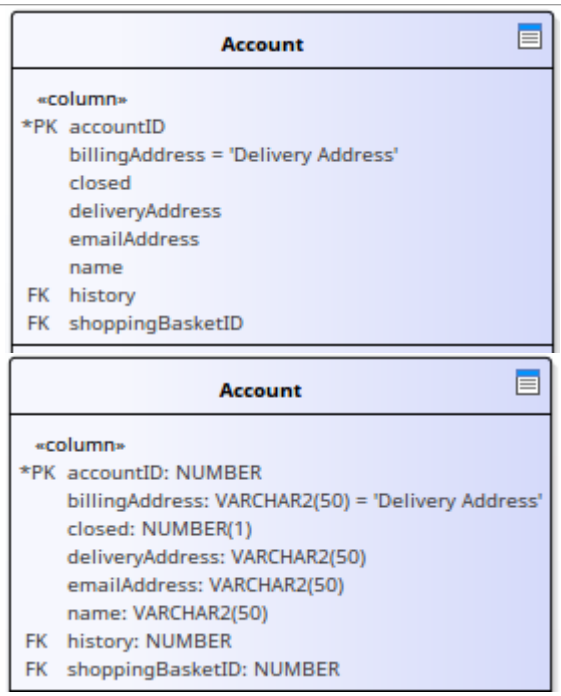
In this field you select the default storage engine to be assigned to MySQL Tables; from MySQL v 5.5 onwards the default value is InnoDB.

Data Modeling Notations

Enterprise Architect supports numerous settings related to data modeling that can influence how database objects are represented on diagrams. These settings, and how they can affect the representation of database objects, are described here.

Settings

Setting	Detail
Stereotype Icons	Access: 'Design > Diagram > Manage > Properties > Elements : Use Stereotype Icons' Default Value: True Enterprise Architect provides a diagram-level setting for the display of stereotyped objects. When the checkbox is selected, database objects on the diagram are displayed with an icon representing their stereotype instead of the stereotype name. 
Show Data Model Owner	Access: 'Design > Diagram > Manage > Properties > Elements : Show Data Model Owner' Default Value: True The system provides a diagram-level setting for the display of Owners. When the checkbox is selected, database objects on the current diagram will be displayed with their full name '{Owner.}ObjectName'. 
Show Column Details	Access: 'Design > Diagram > Manage > Properties > Features : Show Attribute Detail' Default Value: Name Only The system provides a diagram-level setting for the display of Table column names and datatypes. The available options are: 'Name Only' or 'Name and Type'.



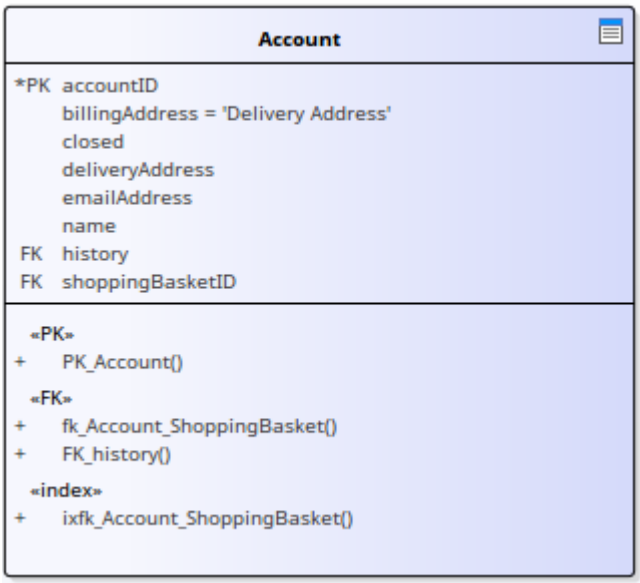
Show Involved Column Details

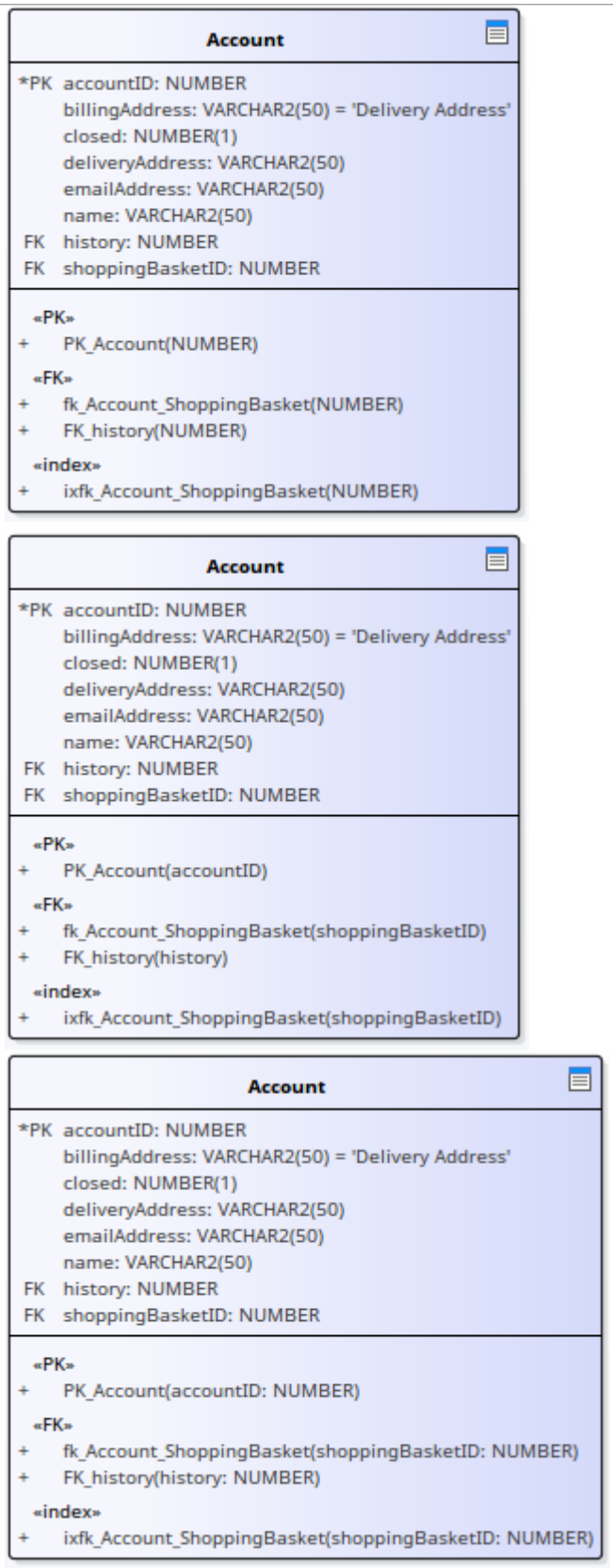
Access: 'Design > Diagram > Manage > Properties > Features | Show Parameter Detail'

Default Value: Type Only

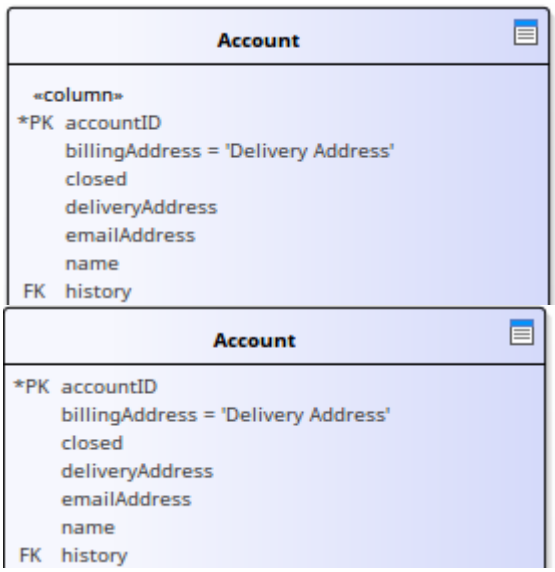
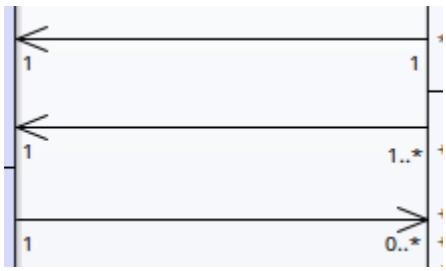
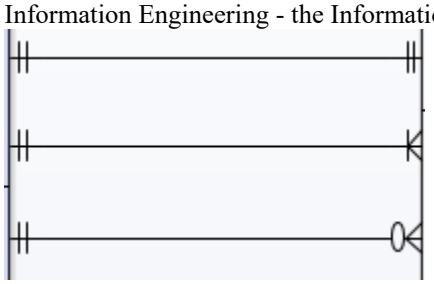
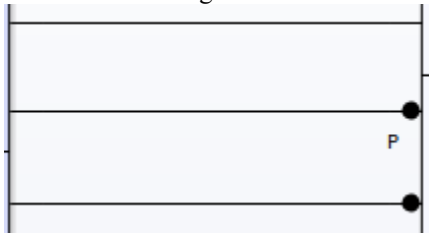
The system provides a diagram-level setting for the display of columns involved in a Table constraint. The available options are: 'None', 'Type Only', 'Name Only' and 'Full Details'.

In these examples, the Primary Key (PK) constraint 'PK_account' involves the column 'accountID'.





Show Column Stereotype	Access: Start > Application > Preferences > Preferences > Objects: Show <<column>> stereotype Default Value: True Enterprise Architect provides a global-level setting that controls whether or not the

	<<column>> stereotype is displayed above each Table's columns. You can therefore hide the stereotype if you prefer, considering that attributes with a stereotype of <<column>> are the only valid option for Tables. 
Connector Notation	Access: 'Design > Diagram > Manage > Properties > Connectors : Connector Notation' Default Value: UML 2.1 Enterprise Architect supports three diagram notations for data modeling: • UML 2.1 - the standard UML 2.1 notation for connectors  • Information Engineering - the Information Engineering (IE) connection style  • IDEF1X - the Integrated Definition Methods IDEF1X connection style  (These are the same three connectors using the different notations.) The default notation for the Data Modeling diagram is 'Information Engineering', whilst the default notation for models created from Database Engineering Patterns is 'IDEF1X'.

DDL Name Templates

At various times during the process of data modeling, Enterprise Architect is required to automatically generate Table constraints. The naming standard for these generated constraints is defined in and applied by the DDL Name Templates, which you are free to change at any time. These Name templates are defined at repository level, so whenever they are changed all users of the repository will use the new templates.

Access

Ribbon	Settings > Model > Options > Source Code Engineering : DDL Name Templates 

DDL Name Templates

Option	Action
Primary Key	Define the name template used when Primary Key constraints are created.
Unique Constraint	Define the name template used when Unique Constraints are created.
Foreign Key	Define the name template used when Foreign Key constraints are created.
Foreign Key Index	Define the name template used when Foreign Key indexes are created.
Save	Click on this button to save the name template(s) you have defined.

Template Macros

These recognized macros will be replaced by name text during the creation of a constraint name.

Macro	Applies to
%tablename%	Primary Key Unique Constraint Description: The string that is replaced by the Table's name.
%columnname%	The string that is replaced by the constraint's column name(s).
%primarytablename%	Foreign Key Description: The string that is replaced by the primary (parent) Table's name.
%foreigntablename%	The string that is replaced by the foreign (child) Table's name.

%foreignkeyname%	Foreign Key Index Description: The string that is replaced by the Foreign Key name.

Import Database Schema

The power of model-based engineering is the ability to visualize, analyze and design all aspects of a system. Being able to view the database schemas alongside other models of a system provides great clarity and reduces the chance of error. Enterprise Architect can reverse engineer a DBMS schema and its objects into a model under a number of different standards, including UML, Information Engineering and IDEF 1X. A wide range of database objects are supported including Tables, Views, Procedures, Functions and Sequences. Enterprise Architect achieves this by interrogating the DBMS's information schema and importing the definition into a UML objects. As modifications are made to the Live database the changes can be synchronized into the model.

Once the schema is in Enterprise Architect, the database objects can be traced to other elements, ensuring the integrity of design and architecture. When systems target multiple DBMSs, these can all be reverse engineered into a model and elements and datatypes can be compared between these models. The sophisticated reporting engine can produce high quality documentation, including data dictionaries, diagrams and relationships back to other models such as architecture and information requirements, and ultimately to business goals and drivers.

Database schema information can be imported via the Database Builder (recommended) or from the 'Develop' ribbon.

Import Database Schema

Step	Action
1	Open the Database Builder (Develop > Data Modeling > Database Builder)
2	Load or create a Data Model.
3	• Right-click on the loaded Data Model in the Database Builder and select 'Import DB schema' or • From the ribbon select 'Develop > Data Modeling > Import' The 'Import DB Schema' dialog displays, showing the details of the current active database connection.

The Import DB Schema dialog

Option	Description
Database	This field shows a description of the current Live connection, in the format: dbms.database_server.database_name If necessary, click on the  button and, select an alternative connection.
Import to	This field shows the target Package that the new objects will be saved to. If you want to specify a different Package, click on the  button and select an alternative Package.
Only include objects from Schema(s)	If the database type supports multiple schemas (such as SQL Server, Oracle, PostgreSQL and DB2 Express) you can filter objects to be retrieved from the database by schema. The available schemas are automatically listed in this panel. Select the checkbox against each schema to include in the import.

	(You can click on the All button to select all the schemas, or the None button to clear all selected checkboxes.) If you suspect that the schema list might have changed since you loaded them, you can refresh the list by clicking on the Reload Schemas button.
Name Filter	The 'Name Filter:' field allows filtering of objects using SQL wildcards appropriate to the DBMS of the schema being imported. For example, for Oracle: • LIKE 'A%' - list objects with a name starting with the letter 'A' • NOT LIKE '%_%' ESCAPE '\' - list objects with a name that does not include an underscore (_) • IN ('TABLE1','TABLE2') - list objects with names that are included in the parentheses • NOT IN ('TABLE1','TABLE2') - list objects with names that are not included in the parentheses Note that only one filter can be entered. You cannot add a second filter using the AND clause. Filtering is not available for MS Access
Filter Options	The 'Filter Options' panel controls what object types and properties are read in from the database schema. Values changed on this screen are saved to the registry so that they are re-applied in the next work session. The available options are briefly described here; select the checkbox against an option to activate it. **Tables** • Tables - Select to import Tables • Table Primary Keys - Select to import Primary Key definitions on Tables • Table Foreign Keys - Select to import Foreign Key definitions on Tables • Table Indexes - Select to import Table Indexes • Unique Constraints - Select to import Unique Constraint definitions on Tables • Check Constraints - Select to import Check Constraint definitions on Tables • Table Triggers - Select to import Trigger definitions on Tables • Table Properties - Select to import extended Table properties • Constraint Properties - Select to import Constraint Properties for Tables • Length Semantics - Select to import length semantic definitions on Oracle string columns **Objects** • Views - Select to import Views • Procedures - Select to import Procedures - As Operations - Select to import Procedures as operations (methods) of a single Class; you can view and edit them through the Database object container 'Properties' dialog (the option defaults to unselected, where the selected items are imported as separate Classes) • Functions - Select to import Functions - As Operations - Select to import Functions as operations (defaults to unselected) • Sequences - Select to import Sequences - As Operations - Select to import Sequences as operations (defaults to unselected) • Package - Select to import Oracle Packages

	Advanced • System Objects - Select to import system Tables, Views and other system objects **Warning**: With the 'As Operations' option for Procedures, Functions and Sequences, if objects have been imported under one setting (selected or unselected) and then you change the setting and import further objects, the objects imported under the first setting are removed.
Synchronization	Select the appropriate radio button to indicate whether the existing Classes are to be updated, or the database objects imported as new objects. If you select the 'Synchronize existing classes' option, also select the appropriate checkboxes to determine whether model comments, column default values and/or Table constraints are to be retained or overwritten with the comments, values and constraints of the imported objects.
Import To	Select the appropriate radio button to indicate whether to update the Package and currently-open data model diagrams, or just the Package. If no diagrams are open, the 'Package Only' radio button defaults to selected and the options are disabled; if the open diagrams are in the selected Package, you can select either option.
Import	Click on this button to start the import. The 'Select Database Objects to Import' dialog displays, listing all the database objects found that match the selection criteria. Select the checkbox against each schema (or object type) to automatically select all objects in that group or to import each object individually. Click on the All button to select all types and objects, or on the None button to clear all selected checkboxes. When you have selected all the objects to import, click on the OK button to continue the import.

Notes

- Within Windows, ODBC DSN can be defined for either 32 or 64 bit applications, therefore care must be taken to ensure that all ODBC DSNs for Enterprise Architect's use are defined sharing the same architecture. This is particular important from Enterprise Architect version 16 onwards because it is now available in both 32 and 64 bit versions. An alternative solution (and what Sparx Systems recommend) is to make use of Native connections, since they work for both architectures.
- The ODBC connection should use the ODBC driver available from the DBMS vendor, such as MySQL's ODBC driver for MySQL, and Oracle's ODBC driver for Oracle; drivers provided by third-party vendors are not supported, including the Microsoft ODBC driver for Oracle
- You can import a suitable ODBC driver for SQLite from http://www.ch-werner.de/sqliteodbc/
- Due to the limitations of SQLite, round tripping of SQLite Table and column comments is not possible; to retain comments entered in an SQLite data model when importing from ODBC, deselect the 'Overwrite Object Comments' checkbox in the 'Synchronization' section of the 'Import DB Schema from ODBC Source' dialog
- If setting up an ODBC connection for reverse engineering, the default settings are sufficient
- The list of Data Modeling Data types is defined as static data (in each repository), so depending on the age of your repository, there could be additional data types available from the 'Data Modeling Data Types' section of the 'Resources' page on the Sparx Systems website

Generate Database Definition Language (DDL)

Once a physical model has been defined and the objects modeled, Enterprise Architect can generate Database Definition Language (DDL) for a variety of objects including database Tables, Views, Functions, Sequences and Procedures. This is a time saving mechanism and reduces the errors that can be introduced by doing this by hand in other tools. Forward engineering is governed by a set of templates that define how UML constructs are converted to the objects in the targeted DBMS. Standard templates are provided for all supported DBMSs, and these can be edited to customize the way the DDL is generated. In the case that a DBMS is not supported out-of-the-box, a new set of templates can be created using the existing ones as a starting point and reference.

When forward engineering DDL, the output can be directed to a file (or a series of files, one for each object) or to the DDL execution engine. The execution engine allows you to execute the DDL immediately, targeting a live database through the active connection. If you direct the output to a file you can execute the DDL against a live database later, at your convenience. The generated files can be opened using the code editor, by selecting F12, Ctrl+E or Alt+7, allowing you to view the DDL inside Enterprise Architect.

Generate DDL For Objects

As you create your database model, you can generate the DDL for an individual object, a Package of objects or the complete data model. The only difference is how you invoke the generate DDL process.

Access

Open the Database Builder window, then use the context menu and select 'Generate DDL'.

Ribbon	Develop > Data Modeling > Database Builder > Click on an object, Package or Data Model node : Generate DDL

Generate Tab

Field/Button	Action
Package	Click on the  button and browse for the Package for which you want to generate DDL, using the Navigator window (a version of the 'Find Package' dialog). (Note: This field might not be displayed in all situations.)
Include All Child Packages	Select this checkbox to include the objects in sub-Packages in the 'Select Objects to Generate' list.
Delete Target Files	When objects are generated to single files, the full filename is stored with the object, and displayed in the 'Target File' column of the 'Select Objects to Generate' list. Click on this button to remove all the existing filenames and prompt for new ones.
Select Objects to Generate	This field displays the list of objects that DDL will be generated for, in the displayed order. If you need to change this order to resolve object dependencies, click on an object to move and click on the  buttons to move that object one position up or down in the sequence. Select each object for which to generate DDL. Click on: • The All button to select every item • The None button to clear all selections • Each of several objects while you press Ctrl, to select a number of individual objects • The first and last objects in a block while you press Shift, to select every object in the block
Save Generated Order	If you have changed the order in which the objects are listed, select the checkbox to save the new sequence when you click on the Generate button.
Refresh	Reload the list of objects, restoring each object to their previous positions (if object positions have been changed).

Single File	Select this radio button if you want to save the generated DDL to a single file. Click on the  button to browse for the file path and file name.
Individual file for each table	Select this radio button if you want to save the DDL generated for each object to a separate file. When you click on the Generate button, the system prompts you for the target file name for each object in turn (if it is not specified already).
Generate to DDL Execution Engine	Select this radio button if you want to save the DDL to the execution engine (the 'Execute DDL' tab of the Database Builder). The DDL Execution Engine provides the facilities for executing the generated SQL script and responding to errors in execution immediately, without having to create an external file and load it into another tool. 'Generate to DDL Execution Engine' is the default option if the Database Builder is open.
Generate	Click on this button to run the Generate DDL process with the options you have selected.
View	If you have generated the DDL to a single external file, click on this button to view the output. By default Enterprise Architect uses the default code editor. However, you can define an alternative default DDL editor on the 'Preferences' dialog ('Start > Application > Preferences > Preferences > Source Code Engineering > Code Editors > DDL').
Close	Click on this button to close the dialog. If you did not generate the DDL, this button also abandons DDL generation for the object.

Options Tab

Set any of these flags to False if you do not want to take the action they initiate.

Group	Options
Table Generation Options	**Tables** - indicates that DDL for Table elements should be generated (*) **Primary Keys** - indicates that DDL for Primary Keys should be generated ($) **Foreign Keys** - indicates that DDL for Foreign Keys should be generated ($) **Indexes** - indicates that DDL for Indexes should be generated ($) **Unique Constraints** - indicates that DDL for Unique Constraints should be generated ($) **Check Constraints** - indicates that DDL for Check Constraints should be generated ($) **Table Triggers** - indicates that DDL for Table Triggers should be generated ($) **Table properties** - indicates that DDL for extended table properties should be generated ($) **Length Semantics** - indicates that DDL for Oracle Length Semantic should be generated ($)

Object Generation Options	**Views** - indicates that DDL for View elements should be generated (*) **Procedures** - indicates that DDL for Procedure elements should be generated (*) **Functions** - indicates that DDL for Function elements should be generated (*) **Sequences** - indicates that DDL for Sequence elements should be generated (*) **Packages** - indicates that DDL for Oracle Packages elements should be generated (*)
Formatting	**Include pre/post queries** - indicates that the generated DDL should include the SQL statements defined in the '_PreStatements' and '_PostStatements' SQL Queries **Include Owners** - indicates that the generated DDL should include the schema/owner of all elements **Include Comments** - indicates that the generated DDL should include any comments **Include Header Comments** - indicates that the generated DDL should include any header comments (#) **Include Object Comments** - indicates that the generated DDL should include any object (such as Table or View) comments (#) **Include Column Comments** - indicates that the generated DDL should include any columns comments (#) **Generate DROP statements** - indicates that the generated DDL should include the DROP statement for objects **Use Database** - indicates that the generated DDL should include a USE Database statement **Use Alias** - indicates that the generated DDL makes use of any object or column aliases **Separate Constraint from Table** - indicates that the generated DDL should define the creation of constraints as separate statements from the Table definition **Include NULL in column definitions** - indicates that the generated DDL should apply the NULL keyword to each column definition that is defined as nullable; that is, columns with their 'NOT NULL' flag unchecked (this option only applies to the DBMSs that support the 'NULL' syntax)

Notes

- (*) - options with this mark will be automatically set to True if you have specified to generate DDL for an individual element of that type; that is, if you select a Table and your 'Generate Table' option is False, Enterprise Architect will change the option to True
- ($) - options with this mark will be disabled if the 'Tables' option is set to False
- (#) - options with this mark will be disabled if the 'Include Comments' option is set to False

- In the Corporate, Unified and Ultimate Editions of Enterprise Architect, if security is enabled you must have 'Generate Source Code and DDL' permission to generate DDL
- For a PostgreSQL database, you must set the 'Sequences' option to True to enable auto increment columns to be created
- If generating Oracle sequences, you must always set the 'Table Triggers' and 'Sequences' options to True, so that a pre-insert trigger is generated to select the next sequence value to populate the column; also, in the column properties, set the 'AutoNum' property to True
- You can edit the DDL templates that the system uses to generate the DDL; these are stored at the repository level so

that all other users of the same repository will automatically use the updated templates

Edit DDL Templates

The DDL Template Editor provides the ability to change the templates that the system uses while generating DDL from a data model. It applies the facilities of the Common Code Editor, including Intelli-sense for the various macros. For more information on Intelli-sense and the Common Code Editor, see the *Editing Source Code* Help topic.

Access

Ribbon	Develop > Data Modeling > Templates

Select and Edit Templates

Option	Action
Language	Click on the drop-down arrow and select the database type (Database Management System).
New Database	Click on this button to create a new set of templates for a non-standard DBMS. The 'Input' dialog displays, on which you type the name of the new DBMS for which you are creating templates. This updates the 'Language' field.
Template	Displays the contents of the selected template, and provides the editor for modifying these contents.
Templates	Lists the base DDL templates, Click on a template name to display and edit the template contents; the name of the selected template is highlighted. The 'Modified' field indicates whether you have modified the default template originally supplied with the system.
Stereotype Overrides	Lists any stereotyped templates that exist for the currently-selected base template. The 'Modified' field indicates whether you have modified a default stereotyped template.
Add New Custom Template	Click on this button to display the 'Create New Custom Template' dialog, on which you select the template type from a drop-down list, and type in a name for the template. The template type becomes a prefix for the name; for example: Namespace_MyDDLTemplate
Add New Stereotyped Override	Select a base template and click on this button to display the 'New Template Override' dialog for adding a stereotyped template for the selected template. From the drop-down lists, select the Class and/or Feature stereotype for which to apply the override template.
Get Default Template	Click on this button to refresh the editor display with the default version of the

	selected template. (This does not delete the changed version of the template.)
Save	Click on this button to overwrite the selected template with the updated contents of the Template panel.
Delete	If you have overridden the selected template, click on this button to delete the overridden template and replace it with the corresponding default DDL template.

Notes

- User-modified and user-defined DDL Templates can be imported and exported as Reference Data (see the *Sharing Reference Data* topic)
- Any user-defined templates for a database type are listed in the 'Export Reference Data' dialog in the 'Code, DDL, Transformation & CSV Templates' table, identified by the DBMS name with the suffix _DDL_Template - if no user-defined templates exist for a DBMS, there is no entry for the DBMS in the dialog
- You must also define any appropriate data types for the DBMS and, if exporting the templates as Reference Data, you must export the 'Model Data Types - Code and DDL' table as well

DDL Template Syntax

DDL Templates are written using Enterprise Architect's Code Template Framework, but they have been extended to support DDL generation.

DDL Template Development

These aspects of DDL Template development are discussed in this section.

Aspect	See also
DDL Templates	DDL Templates
DDL Macros	DDL Macros
DDL Function Macros	DDL Function Macros
DDL Property Macros	DDL Property Macros
DDL Options in Templates	DDL Options in Templates

DDL Templates

The DDL Template Editor operates in the same way as the Code Template Editor, except that the DDL Template Editor displays templates for DDL Generation and templates for Alter DDL Generation at the same time. The Alter DDL Generation templates are shown at the bottom of the list, prefixed by 'DDL Diff'.

Base Templates for DDL Generation

The DDL Template Framework consists of a number of base templates for DDL Generation. Each base template generates a DDL statement (or a partial statement) for a particular aspect of the UML data model.

Templates

This table lists and briefly describes the base templates used for DDL generation.

Template	Description
DDL Check Constraint	Invoked by the DDL Table Constraint template to generate the Check Constraint statements for a Table object.
DDL Column Comment	Normally invoked by the DDL Create Table Extras template to generate COMMENT ON statements (or equivalent) for each Table column.
DDL Column Definition	Invoked by numerous templates to build the statement to create a single Table column, as it appears in the CREATE TABLE statement.
DDL Column Extras	Normally invoked by the DDL Create Table Extras template to generate any extended column properties for each Table column.
DDL Constraint Column Name	Invoked by each of the constraint templates to retrieve the correctly formatted column names involved in the current constraint.
DDL Constraint Comment	Normally invoked by the DDL Create Table Extras template to generate COMMENT ON statements (or equivalent) for each Table constraint.
DDL Create Foreign Keys	Invoked by the DDL Create Table Constraints template to generate Foreign Key constraints for a Table object.
DDL Create Function	Invoked by the DDL Script File template to generate the CREATE FUNCTION statement for a Function object.
DDL Create Package	Invoked by the DDL Script File template to generate the CREATE PACKAGE statements for a Package object (Oracle only).
DDL Create Procedure	Invoked by the DDL Script File template to generate the CREATE PROCEDURE statement for a Procedure object.
DDL Create Schema	Currently not used.
DDL Create Sequence	Invoked by the DDL Script File template to generate the CREATE SEQUENCE statement for a Sequence object.
DDL Create Table	Invoked by the DDL Script File template to generate the CREATE TABLE statement for a Table object.
DDL Create Table Constraints	Invoked by the DDL Script File template to generate Table constraints and Indexes for a Table object.

DDL Create Table Extras	Invoked by the DDL Script File template to generate extended Table properties for a Table object.
DDL Create View	Invoked by the DDL Script File template to generate the CREATE VIEW statement for a View object.
DDL Data Type	Invoked by the DDL Column Definition template to generate the correctly formatted data type statement for a Table column.
DDL Drop Column Extras	Invoked by the DDL Drop Table Extras template to generate any specialized drop statements for column extended properties.
DDL Drop Foreign Keys	Invoked by the DDL Script File template to generate the statements to DROP all Foreign Keys for a Table object.
DDL Drop Function	Invoked by the DDL Script File template to generate the DROP FUNCTION statement for a Function object.
DDL Drop Procedure	Invoked by the DDL Script File template to generate the DROP PROCEDURE statement for a Procedure object.
DDL Drop Sequence	Invoked by the DDL Script File template to generate the DROP SEQUENCE statement for a Sequence object.
DDL Drop Table	Invoked by the DDL Script File template to generate the DROP TABLE statement for a Table object.
DDL Drop Table Extras	Invoked by the DDL Script File template to generate the statements to DROP all extended properties for a Table object.
DDL Drop View	Invoked by the DDL Script File template to generate the DROP VIEW statement for a View object.
DDL Foreign Constraint	Invoked by the DDL Table Constraint template to generate the ADD FOREIGN KEY CONSTRAINT statements for a Table object.
DDL Grant	Invoked by the DDL Create Table Extras template to generate the GRANT statement for the current object (Oracle only).
DDL Index	Invoked by the DDL Table Constraint template to generate the CREATE INDEX statements for a Table object.
DDL Left Surround	Used to define the character (or characters) used as the left-hand delimiter on the name of an object (or object component).
DDL Name	Used by most templates to provide a common way of formatting the name of an object (or object feature). This template accepts four parameters: • Object Location (values: EA or LIVE) • Object Type (values: OWNER, TABLE, VIEW, PROCEDURE, FUNCTION, SEQUENCE, PACKAGE, COLUMN, CONSTRAINT, CONSTRAINT_COLUMN, REFERENCE_TABLE, REFERENCE_COLUMN) • Include Owner flag; controls if the name should be prefixed by the Owner name (values: INCLUDE_OWNER or {blank})

	• Include Surround flag; controls if the name should be delimited by the left and right surround characters (values: INCLUDE_SURROUND or {blank})
DDL Primary Constraint	Invoked by the DDL Table Constraint template to generate the ADD PRIMARY KEY CONSTRAINT statement for a Table object.
DDL Reference Column Name	Normally invoked by the DDL Name templates to retrieve the correctly formatted reference column names involved in a Foreign Key.
DDL Reference Definition	Invoked by the DDL Foreign Constraint template to generate the ON DELETE/ON UPDATE statements for a Foreign Key constraint.
DDL Right Surround	Used to define the character (or characters) used as the right-hand delimiter on the name of an object (or object component).
DDL Script File	A top-level template to generate DDL; all other templates are invoked from this one.
DDL Script Header	Invoked by the DDL Script File template to add a header comment at the start of each DDL file.
DDL Script Separator	Used by all templates that must include a statement separator in the generated DDL.
DDL Statement Term	Used to define the character (or characters) used as the statement terminator. For example, semi-colon (';') for most DBMSs.
DDL Statement Term Alt	Used to define the character (or characters) used as the alternative statement terminator. For example, some DBMSs must have the statement terminator changed in order to not cause problems with DDL statements generated for SQL-based objects, such as Views and Procedures.
DDL Synonym	Invoked by the DDL Create Table Extras template to generate the CREATE SYNONYMS statement (Oracle only).
DDL Table Constraint	Invoked by the DDL Create Table Constraints template to generate the Table constraints and Indexes for each Table object, taking into account the generation options.
DDL Table Level Comment	Invoked by the DDL Create Table Extras template to generate COMMENT ON statements (or the equivalent) for an object.
DDL Trigger	Invoked by the DDL Table Constraint template to generate the CREATE TRIGGER statements for a Table object.
DDL Unique Constraint	Invoked by the DDL Table Constraint template to generate the ADD UNIQUE CONSTRAINT statements for a Table object.
DDL Use Database	Invoked by the DDL Script File template to include a USE DATABASE statement at the start of each DDL file.

Base Templates for Alter DDL Generation

The DDL Template Framework consists of a number of base templates for Alter DDL generation. Each base template generates DDL statement(s) based on the detected *Action* that must be undertaken to synchronize the data model and live database.

Templates

This table lists and briefly describes the base templates used for Alter DDL generation.

Template	Description
DDL Diff Column	Invoked directly by Enterprise Architect for each Table Column difference that was detected.
DDL Diff Constraint	Invoked directly by Enterprise Architect for each Table Constraint difference that was detected.
DDL Diff Table	Invoked directly by Enterprise Architect for each Table difference that was detected.
DDL Diff View	Invoked directly by Enterprise Architect for each View difference that was detected.
DDL Diff Procedure	Invoked directly by Enterprise Architect for each Stored Procedure difference that was detected.
DDL Diff Function	Invoked directly by Enterprise Architect for each Function difference that was detected.
DDL Diff Sequence	Invoked directly by Enterprise Architect for each Sequence difference that was detected.

DDL Macros

Field substitution macros provide access to data from your model. In particular, they are used to access data fields from:

- Database objects (such as Tables and Views)
- Columns
- Constraints
- Constraint Columns

Field substitution macros are named according to Camel casing. By convention, all DDL macros are prefixed with 'ddl'.

Macros that represent checkboxes or Boolean values return a string value of 'T' if the checkbox/boolean is true. Otherwise an empty string is returned.

Internal Field Macro - ddlAction

The ddlAction macro is an internal macro available in the 'Alter DDL' templates, providing direct access to Enterprise Architect's internal fields; it has no direct mapping to any stored data.

ddlAction represents the action that must be undertaken to synchronize the live database with the current repository. For example, 'Create Table', 'Drop Table' or 'Change Owner'.

Element Field Macros

This list identifies the macros that are available in DDL templates to access element-level fields, where (in Enterprise Architect) the fields are editable, such as 'Table Name' and 'Table Alias'.

ddlFunctionAlias

Function 'Properties' dialog: 'Main' tab: 'Alias' text field.

ddlFunctionName

Function 'Properties' dialog: 'Name' text field.

ddlOwner

{Table element} 'Properties' dialog: {element} 'Table Detail' tab: 'Owner' text field.

ddlPackageAlias

Package 'Properties' dialog: 'Main' tab: 'Alias' text field.

ddlPackageName

Package 'Properties' dialog: 'Name' text field.

ddlProcedureAlias

Procedure 'Properties' dialog: 'Main' tab: 'Alias' text field.

ddlProcedureName

Procedure 'Properties' dialog: 'Name' text field.

ddlSchemaFunctionName

The name of the Function element's definition read in from the live database.

ddlSchemaOwner

The 'Owner' property of the element's definition read in from the live database.

ddlSchemaProcedureName

The name of the Procedure element's definition read in from the live database.

ddlSchemaSequenceName

The name of the Sequence element's definition read in from the live database.

ddlSchemaTableName

The 'Table Name' property read in from the live database.

ddlSchemaViewName

The name of the View element's definition read in from the live database.

ddlSequenceAlias

Sequence 'Properties' dialog: 'Main' tab: 'Alias' text field.

ddlSequenceName

Sequence 'Properties' dialog: 'Name' text field.

ddlTableAlias

Table 'Properties' dialog: 'Main' tab: 'Alias' text field.

ddlTableDBMS

Table 'Properties' dialog: 'Main' tab: 'Database' drop down list field.

ddlTableLevelComment

Table 'Properties' dialog: 'Notes' text field.

ddlTableName

Table 'Properties' dialog: 'Name' text field.

ddlViewAlias

View 'Properties' dialog: 'Main' tab: 'Alias' text field.

ddlViewName

View 'Properties' dialog: 'Name' text field.

Column Field Macros

This list identifies the macros that are available in DDL templates to access column-related fields, where (in Enterprise Architect) the fields are editable, such as 'Column Name' and 'Column Alias'.

ddlColumnName

'Columns and Constraints' dialog: 'Column' tab: 'Name' cell.

ddlColumnAlias

'Columns and Constraints' dialog: 'Column' tab: 'Alias' cell.

ddlColumnComment

'Columns and Constraints' dialog: 'Column' tab: 'Notes' text field.

ddlSchemaColumnName

The Column **Name** property read in from the live database.

Note: This field is not editable directly in Enterprise Architect.

Constraint Field Macros

This table lists the macros that are available in DDL templates to access constraint-related fields, where (in Enterprise Architect) the fields are editable, such as 'Constraint Name' and 'Constraint Type'.

ddlConstraintAlias

'Columns and Constraints' dialog: 'Constraints' tab: 'Alias' cell.

ddlConstraintColumnAlias

'Columns and Constraints' dialog: 'Constraints' tab: 'Involved Columns: Assigned' list.

ddlConstraintColumnName

'Columns and Constraints' dialog: 'Constraints' tab: 'Involved Columns: Assigned' list.

ddlConstraintComment

'Columns and Constraints' dialog: 'Constraints' tab: 'Notes' text field.

ddlConstraintName

'Columns and Constraints' dialog: 'Constraints' tab: 'Name' cell.

ddlPKColumnCount

Only relevant if the current constraint has a type of Primary Key, this macro will return a count of assigned columns to the Primary Key.

'Columns and Constraints' dialog: 'Constraints' tab: 'Involved Columns: Assigned' list.

ddlReferenceColumnAlias

Only relevant if the current constraint has a type of Foreign Key, this macro will return the column alias from the reference table.

'Columns and Constraints' dialog: 'Constraints' tab: 'Alias' cell.

ddlReferenceColumnName

Only relevant if the current constraint has a type of Foreign Key, this macro will return the column name from the reference table.

Foreign Key 'Constraint' dialog: 'Involved Columns' list: 'Parent' column.

ddlReferenceTableAlias

Only relevant if the current constraint has a type of Foreign Key, this macro will return the reference table's alias.

Table 'Properties' dialog: 'Main' tab: 'Alias' text field.

ddlReferenceTableName

Only relevant if the current constraint has a type of Foreign Key, this macro will return the reference table's name.

Foreign Key 'Constraint' dialog: 'Involved Columns' list: 'Parent' column header.

ddlReferenceTableOwner

Only relevant if the current constraint has a type of Foreign Key, this macro will return the reference table's owner.

Foreign Key 'Constraint' dialog: 'Involved Columns' list: 'Parent' column header.

ddlSchemaConstraintColumnName

The column names involved in the current constraint read in from the live database.

Note: this field is not editable directly in Enterprise Architect.

ddlSchemaConstraintName

The Constraint **Name** property read in from the live database.

Note: this field is not editable directly in Enterprise Architect.

ddlSchemaConstraintType

The Constraint **Type** property read in from the live database.

Note: this field is not editable directly in Enterprise Architect.

DDL Function Macros

The DDL Function macros provide a convenient way of manipulating, retrieving or formatting element data relevant to DDL generation. These macros, along with the code function macros, are available to the DDL templates. Each Function macro returns a result string and is used in the same manner as a Code Template Function macro.

The available function macros are described here. All parameters have a type of String and are denoted by square brackets; that is: FUNCTION_NAME([param]).

DDL_DATATYPE_SIZE ([productName], [datatype])

Returns the fully formatted datatype of the current column in DDL syntax.

Parameters

- productName - the current Table's assigned DBMS, such as SQL Server 2012, Oracle or PostgreSQL
- datatype - the current column's datatype name, such as VARCHAR or INT

Remarks

Within an Enterprise Architect Table column, datatypes are defined with a Length Type (0, 1 or 2) property that influences the DDL syntax; this function macro takes the Length Type (and other factors) into consideration when building the return value.

DDL_GET_DEFINITION_PARAS ([definition])

Returns a string representation of the parameters from the supplied function/procedure definition.

Parameters

- definition - the complete SQL definition of the procedure/function

Remarks

Some DBMSs (such as PostgreSQL) support multiple definitions of the same procedure/function name. The definitions differ only in their parameter list, therefore to manipulate such objects the DDL must specify the name and parameters. This function macro gives the DDL templates the ability to extract the parameters so that they can then be used to identify individual objects.

DDL_INCLUDE_SQLQUERY([objectName])

Returns the SQL statement defined in the SQLQuery object.

Parameters

- objectName - the name of the SQL Query object defined in the current data model

Remarks

None.

DDL_INDEX_SORT([product],[columns])

Returns the sort order of a given index.

Parameters

- product - the DBMS (currently, Firebird)
- columns - a CSV of column names involved in the index

Remarks

This macro currently only applies to Firebird indexes.

DDL_RESOLVE_NAME ([productName], [name], [leftSurround], [rightSurround])

Returns the supplied name delimited (with the supplied left and right characters) if the name is a reserved word for the current DBMS.

Parameters

- productName - the current Table's assigned DBMS, such as SQL Server 2012, Oracle or PostgreSQL
- name - the object/column name
- leftSurround - the left character of the pair used to surround the name; for example, single quote {'}
- rightSurround - the right character of the pair used to surround the name; for example, single quote {'}

Remarks

The DDL syntax of some DBMSs requires names that are reserved words to be delimited in a different manner; this function macro can be used to safely format all names for DB2 and Firebird.

DDL_TABLE_TAGVALUE ([tagName])

Returns the value for the supplied tag name in the repository's version of the current Table.

Parameters

- tagName - the tag item's name that is to be retrieved

Remarks

None.

EXECUTE_CURRENT ([objectName], [actionName], [priority])

Adds the return string from the current template to the Execution Engine's execution queue.

Parameters

- objectName - the value that will be shown in the 'Object' column of the execution queue, which indicates the name of the object being updated
- actionName - the value that will be shown in the 'Action' column of the execution queue, which indicates the action that resulted in the generation of this statement
- priority - a numeric value that represents the priority of the statement; the higher the number, the lower in the queue the statement is placed

Remarks

This function macro can be called at any point throughout the template, but will not execute until the end. Once the template is complete, the DDL it has generated is sent to the execution queue.

This function macro has no effect if the user has elected to generate DDL to a file.

EXECUTE_STRING ([objectName], [actionName], [priority], [ddlStatement])

Adds the supplied DDL statement to the Execution Engine's execution queue.

Parameters

- objectName - the value that will be shown in the 'Object' column of the execution queue, which indicates the name of the object being updated
- actionName - the value that will be shown in the 'Action' column of the execution queue, which indicates the action that resulted in the generation of this statement
- priority - a numeric value that represents the priority of the statement; the higher the number, the lower in the queue the statement is placed
- ddlStatement - a single DDL statement that performs the required action

Remarks

This function macro has no effect if the user has elected to generate DDL to a file.

EXIST_STRING ([ddlStatement])

Searches the Execution Engine's execution queue for the supplied DDL Statement and returns 'T' if the statement is found.

Parameters

- ddlStatement - a single DDL statement

Remarks

None.

GET_FIRST_SQL_KEYWORD([statement])

Returns the first keyword of the provided SQL statement.

Parameters

- statement - the SQL statement

Remarks

None.

ODBC_TABLE_TAGVALUE ([tagName])

Returns the value for the supplied tag name in the live database's version of the current table.

Parameters

- tagName - the tag item's name that is to be retrieved

Remarks

None.

PROCESS_DDL_SCRIPT ([type], [parameter2], [parameter3], [parameter4])

A generic function macro that returns a formatted string for a specific purpose.

Parameters

- type - specifies the special action to be undertaken
- parameter2 - generic parameter 2, will have a different purpose for each type
- parameter3 - generic parameter 3, will have a different purpose for each type
- parameter4 - generic parameter 4, will have a different purpose for each type

Remarks

For Oracle Synonyms use these parameters:

- type = "SYNONYMS"
- parameter2 = the table name; for example, TBL_EMPLOYEES
- parameter3 = a delimited string of values, separated by semi-colons, specifying the synonym owner and name with full colon between; for example, OE:EMPLOYEES;PUBLIC:PUB_EMPLOYEES;
- parameter4 = the statement terminator

Return Result

Of the format:

```
CREATE SYNONYM OE.EMPLOYEES FOR TBL_EMPLOYEES;
CREATE PUBLIC SYNONYM PUB_EMPLOYEES FOR TBL_EMPLOYEES;
```

REMOVE_LAST_SEPARATOR ([ddlStatement], [separator])

Returns the supplied DDL statement with the last separator removed (if it exists).

Parameters

- ddlStatement - a partial DDL statement
- separator - the separator character that should be removed

Remarks

When building a string that represents a DDL statement, it is common practice to append the separator character after each item; however, the separator is not required after the last item, so this function macro is provided to remove the trailing separator.

REMOVE_STRING ([ddlStatement])

Removes the supplied DDL statement from the Execution Engine's execution queue.

Parameters

- ddlStatement - a single DDL statement

Remarks

None.

SUPPRESS_EXECUTE_CURRENT ([boolean])

A function macro to enable/disable subsequent calls to EXECUTE_CURRENT.

Parameters

- boolean - True or False

Remarks

The default state for this flag is False; that is, calls to EXECUTE_CURRENT are not ignored.

DDL Property Macros

The DDL Property macros provide a convenient way of retrieving element property values (that is, Tagged Values). In the scope of data modeling there are two groups of properties:

- Internal properties (those that Enterprise Architect recognizes and uses in its compares) and
- User-defined properties

These property macros provide access to properties defined against the various elements. All property macros have the same syntax, return a string and require the name of the property to be specified.

Syntax: propertyMacroName:"propertyName"

INTERNAL PROPERTIES

tableBoolProperty:"propertyName"

Returns a Boolean representation ("T" or "") of the value for the internal property in the repository's version of the current Table.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

tableProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current Table.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

columnProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current Column.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

columnBoolProperty:"propertyName"

Returns a Boolean representation ("T" or "") of the value for the internal property in the repository's version of the current Column.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

constraintProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current Constraint.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

constraintBoolProperty:"propertyName"

Returns a Boolean representation ("T" or "") of the value for the internal property in the repository's version of the current Constraint.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

constraintColumnProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current Constraint Column.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

constraintColumnBoolProperty:"propertyName"

Returns a Boolean representation ("T" or "") of the value for the internal property in the repository's version of the current Constraint Column.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

viewProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current View.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

procedureProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current Procedure.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

functionProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current Function.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

sequenceProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current Sequence.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

packageProperty:"propertyName"

Returns the value for the internal property in the repository's version of the current database Package.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

odbcTableProperty:"propertyName"

Returns the value for the internal property in the ODBC's version of the current Table.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

odbcConstraintProperty:"propertyName"

Returns the value for the internal property in the ODBC's version of the current Constraint.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

USER DEFINED PROPERTIES

tableUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current Table.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

columnUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current Column.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

constraintUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current Constraint.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

constraintColumnUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current Constraint Column.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

viewUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current View.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

procedureUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current Procedure.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

functionUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current Function.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

sequenceUserProperty:"propertyName"

Returns the value for the user-defined property in the repository's version of the current Sequence.

Parameters

- propertyName - the property name that is to be retrieved

Remarks

None.

DDL Options in Templates

The DDL Generation Options macros provide a convenient way for the DDL templates to access the generation options.

This list identifies and briefly describes each of the available option macros. Each option has a value of either 'T' for true or an empty string for false.

ddlGenerateToExecuteEngine

Directs the generated DDL to the Execution Engine.

ddlOptionColumnComments

Include column comments in the generated DDL.

ddlOptionGenerateCheck

Include Check constraints in the generated DDL.

ddlOptionGenerateDrop

Include DROP statements in the generated DDL.

ddlOptionGenerateForeign

Include Foreign Keys in the generated DDL.

ddlOptionGenerateFunction

Include Functions in the generated DDL.

ddlOptionGenerateIndex

Include Indexes in the generated DDL.

ddlOptionGenerateLengthSemantic

(Oracle only) Include length semantics syntax on text columns in the generated DDL.

ddlOptionGenerateNullable

Include the keyword NULL against each column if it hasn't been flagged as a NOT NULL column in the generated DDL.

ddlOptionGeneratePackage

(Oracle only) Include Packages in the generated DDL.

ddlOptionGeneratePrimary

Include Primary Key constraints in the generated DDL.

ddlOptionGenerateProcedure

Include Procedures in the generated DDL.

ddlOptionGenerateSeparateConstraint

Generate Table constraints separately to the CREATE TABLE statement; that is, using an ALTER TABLE statement.
Note: Some DBMSs do not support separate constraints in all conditions.

ddlOptionGenerateSequence

Include Sequences in the generated DDL.

ddlOptionGenerateTable

Include Tables in the generated DDL.

ddlOptionGenerateTableProperty

Include extended properties on Tables in the generated DDL.

ddlOptionGenerateTrigger

Include Table Triggers in the generated DDL.

ddlOptionGenerateUnique

Include Unique Constraints in the generated DDL.

ddlOptionGenerateView

Include Views in the generated DDL.

ddlOptionHeaderComments

Include header comments in the generated DDL.

ddlOptionTableComments

Include Table comments in the generated DDL.

ddlOptionUseAlias

Use Aliases instead of Names for all objects (object components) as specified on the Generate DDL screen.

ddlOptionUseDatabaseName

Include the USE DATABASE statement at the beginning of each generated file.

ddlUseAlias

Use Aliases instead of Names for all objects (object components) as specified on the Database Builder 'Database Compare' tab.

DDL Limitations

A fundamental feature of a Database Management System (DBMS) is to allow the definition of database objects via a structured language; this language is called DDL (for data definition language, or data description language). The DDL syntax of each DBMS is unique. While there are common DDL statements and keywords across all DBMSs, there are differences that require each DBMS to have its own set of DDL templates within Enterprise Architect.

This page summarizes the main limitations for each of the supported Database Management Systems.

MS Access

- Comments cannot be applied to (or changed in) Tables, Table Columns, Table Constraints or Views, therefore Enterprise Architect ignores these differences
- The CREATE TABLE statement does not support the definition of column defaults, therefore Enterprise Architect excludes the Default definition from all generated DDL; however, it does highlight a Default difference in the comparison logic
- Generally object names in DDL can be enclosed in square brackets ([ ]) so that they can include spaces and other non standard characters, however the CREATE VIEW DDL statement does not support the square bracket notation; the 'Create View' DDL template replaces all spaces with underscore ('_') characters

MySQL

- Comments can only be applied to Indexes and Unique Constraints, when the MySQL version is greater than 5.5.3
- Comments can only be applied to Indexes and Unique Constraints when they are created, therefore changing an Index or Unique Constraint's comment causes the constraint to be dropped and recreated
- Check Constraints are not supported; whilst the MySQL DDL engine can parse such statements, it simply ignores them
- Comments cannot be applied to (or changed in) Views, Procedures or Functions, therefore Enterprise Architect ignores these differences

Oracle

- Comments cannot be applied to (or changed in) Procedures, Sequences or Functions, therefore Enterprise Architect ignores these differences

PostgreSQL

- Currently Enterprise Architect does not support function parameters, therefore any statements (COMMENT ON or DROP) that refer to a function by name will fail because they must use a combination of function name and parameters

SQL Lite

- Constraints cannot be added to an existing Table; the Table must be dropped and created (including the new Constraint in the Create statement)

- Comments are not supported on any object type, therefore Enterprise Architect ignores all remark differences

Import DDL Script

This feature allows you to import DDL scripts from a specified directory in your file system, to create Database Model objects in your Enterprise Architect model. All of the scripts in the directory, whose filename extensions match with those specified will be imported.

The script files will be imported into the currently selected Enterprise Architect package, creating Tables, Views, Columns, Constraints, Procedures, Functions, Sequences and so forth, as defined by the DDL scripts.

Access

Ribbon	Develop > Data Modeling > Import DDL

Import DDL Scripts dialog

Directory	Type in or browse for the name of the directory to import.
Process Subdirectories	Select this check box to include the contents of the subdirectories as well.
DBMS	Select from the drop-down list the DBMS type for which the DDL scripts are applicable.
File Extensions	Type in or select from the drop-down list, the filename extensions to be included in the import. Use a ";" to separate values.
Import	Click the '**Import**' button to begin the import.

Supported Database Management Systems

Enterprise Architect has built in support for a comprehensive range of database management systems, but it also provides the flexibility to extend the product to support other DBMSs. The DDL template editor can be used to define how to generate DDL for an unsupported DBMS, the transformation templates can be used to define a new transformation to a physical model for an unsupported DBMS, and new datatypes can be defined for an existing or new DBMS.

Enterprise Architect provides the modeling constructs and the ability to forward and reverse engineer a database schema for these Database Management Systems:

- DB2 (*)
- Firebird
- MS Access 97, 2000, 2003, 2007, 2013
- MS SQL Server from 2005, all editions including Express and Azure SQL Database
- MariaDB
- MySQL v4, v5
- Oracle from 9i (all editions)
- PostgreSQL (including version 12)
- SQLite
- Informix (#)
- Ingres (#)
- InterBase (#)
- Sybase Adaptive Server Anywhere (Sybase ASA) (#)
- Sybase Adaptive Server Enterprise (Sybase ASE) (#)

(*) - Only compatible for DB2 when hosted in Windows and Linux environments.

(#) - No further development will be undertaken on these DBMSs, as these products are not commonly used by the Enterprise Architect user base. This will allow Sparx Systems to concentrate its efforts on the other areas of Database modeling which are used extensively.

Notes

- To perform data modeling for a particular DBMS, you must have the appropriate data types for that DBMS in your repository; you can download the most up-to-date data definitions from the 'Resources' page of the Sparx Systems web site

More Information

Sparx Systems Enterprise Architect provides information modelers, data modelers, and architects practical tools for creating models that span abstraction levels within an organization: conceptual, logical, and physical.

Conceptual Models: These are technology-independent and aid in discussions with business and domain experts to represent and agree on basic domain concepts.

Logical Models: These add detail and precision to conceptual models while remaining technology-neutral, facilitating discussions among information analysts on logical structures.

Physical Models: These apply technology-specific data to models, aiding engineers in making technology decisions for deployment in target environments like database management systems.

Edition Information

The Database Builder is available in the Corporate, Unified and Ultimate Editions of Enterprise Architect.