CUSTOMIZATION TOOL

USER'S GUIDE

September 2018, Release 1.1

DialogueTech

Third Edition (September 2018)
This major revision obsoletes and replaces all previous versions. This edition applies to Release 2.0 of Ergo, and to all subsequent releases and modifications until otherwise indicated in new editions. Address any comments you may have on the document to:

Dialogue Technologies AB
Ankdammsgatan 20
S-171 43 Solna, Sweden, or
info@dialoguetech.com

CONTENTS

Preface

The objectives of this document are:

1. To document our knowledge of and experiences with Dialogue Technologies Ergo application customization to enable customers to have access to all the information they need to customize their application.

2. To provide a reference document to facilitate both customization and support efforts by demonstrating how to solve specific kinds of generic customization problems and by identifying relevant product limitations for minimizing customization time.

PART 1. Advanced Customization Guide

1. OVERVIEW OF ERGO

Dialogue Technologies Ergo is a query product, which uses advanced grammar-based Natural Language Processing (NLP) technologies. It provides end-users and business professionals using their own natural language, such as English, with the ability to access information stored in relational database management systems (RDBMSs) or to control applications.

The components of Ergo are:
i) Query Interface (QI) for interactions between the users and the database:
- QI for voice users
- QI for text input

ii) Natural Language Engine (NLE) with an API for processing the queries
- Natural language analyzer
- Natural language generator

iii) Customization Tool (CT) for customizing applications:
- Conceptual modeling facility
- Model transfer facilities

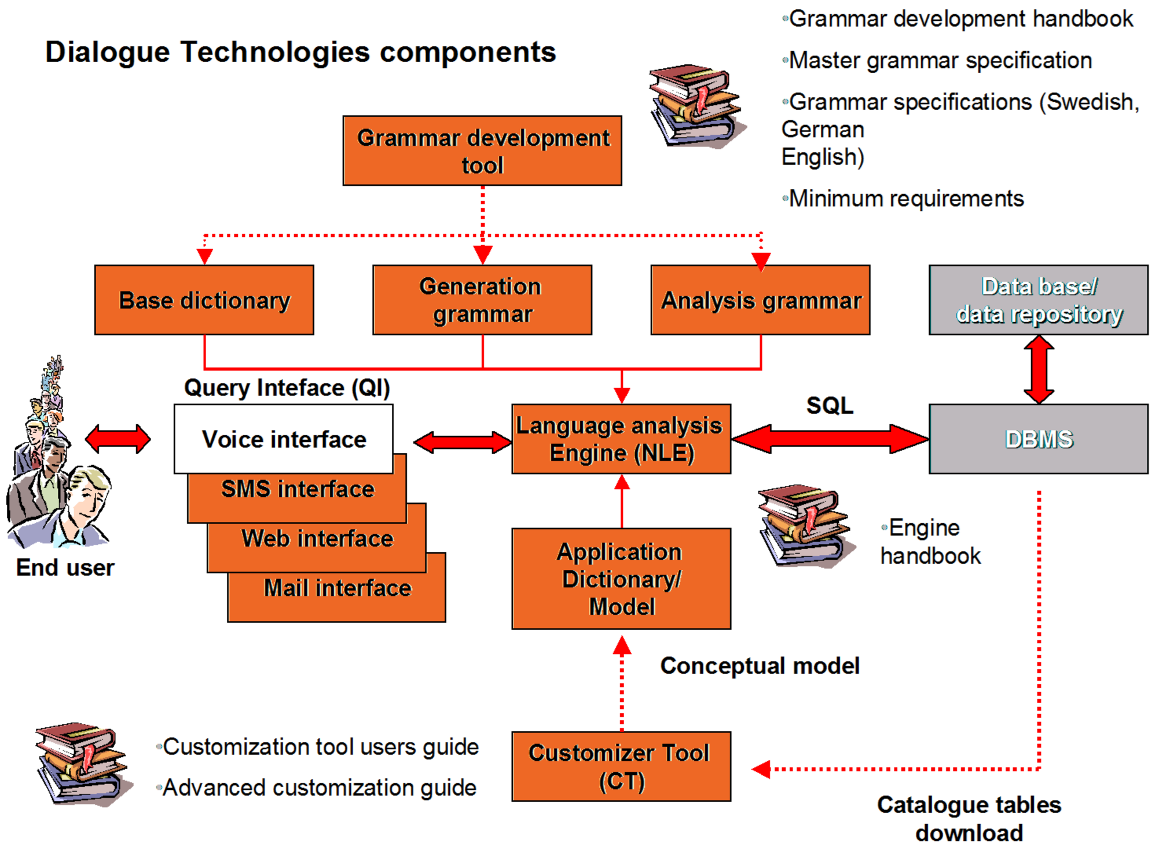

Dialogue Technologies components
•Grammar development handbook
•Master grammar specification
•Grammar specifications (Swedish, German English)
•Minimum requirements
Grammar development tool
Base dictionary
Generation grammar
Analysis grammar
Data base/ data repository
Query Inteface (QI)
Voice interface
SMS interface
Web interface
Mail interface
End user
Language analysis Engine (NLE)
SQL
DBMS
Application Dictionary/ Model
•Engine handbook
Conceptual model
•Customization tool users guide
•Advanced customization guide
Customizer Tool (CT)
Catalogue tables download

Figure 1 Components of an application

1.1 The Query Interface

The query interface handles communication between the user and the database. It can be text-oriented for applications with text input via e.g. the web, a special input window, SMS, e-mail, etc., or based on using a voice front-end for converting speech to text.

1.2 The Natural Language Engine

The Natural Language Engine (NLE) is the Ergo component that transforms queries into structured query language (SQL) statements (or extensive markup language, XML, etc.) that is sent to the database management system, which retrieves the answer. The input to this component is a natural language query. The output is the SQL command into which the input query has been translated as well as a set of paraphrases of the input query.

The NLE is designed with an Application Programming Interface (API) so it can be used as an embedded component in other products. In principle, the API can be used by any program. It is up to the calling program to handle user interactions, to use the generated SQL commands for retrieving data, and to answer users' questions based on the data and the information returned from the API.

When an end user submits a natural language query, it is parsed. Valid sentences are translated to an internal format using an analysis grammar for the language of the query. The vocabulary that is used for parsing consists of the application-independent words contained in the built-in dictionary and the application-specific words that have been added during the customization process. The query is translated back into natural language as a paraphrase to permit the user to confirm that the query has been understood correctly by the computer. In cases of ambiguity, alternative paraphrases are presented to the user for selection of the correct one. After confirmation, the translated query is sent to the database and the answer is displayed for the user. The query can be of the yes/no sort. For example, the user types in the following query:

Which senior manager works at the head office?

which results in the interpretation:

Find senior managers that work at departments named head office.

Ergo can handle natural language questions, which cannot be expressed in a single SQL query. The natural language query is translated into an answer set, which is beyond pure SQL. Besides SQL statements, the answer set contains information on the data representation (yes/no, report) derived from the input natural language question and, in cases when the data cannot be retrieved by a single SQL query. It also includes SQL statements for creating intermediate relations. Intermediate relations are temporary relations created as SQL tables containing data to be retrieved by a query. When the user selects an interpretation, the corresponding SQL is sent off to the Database Management System (DBMS).

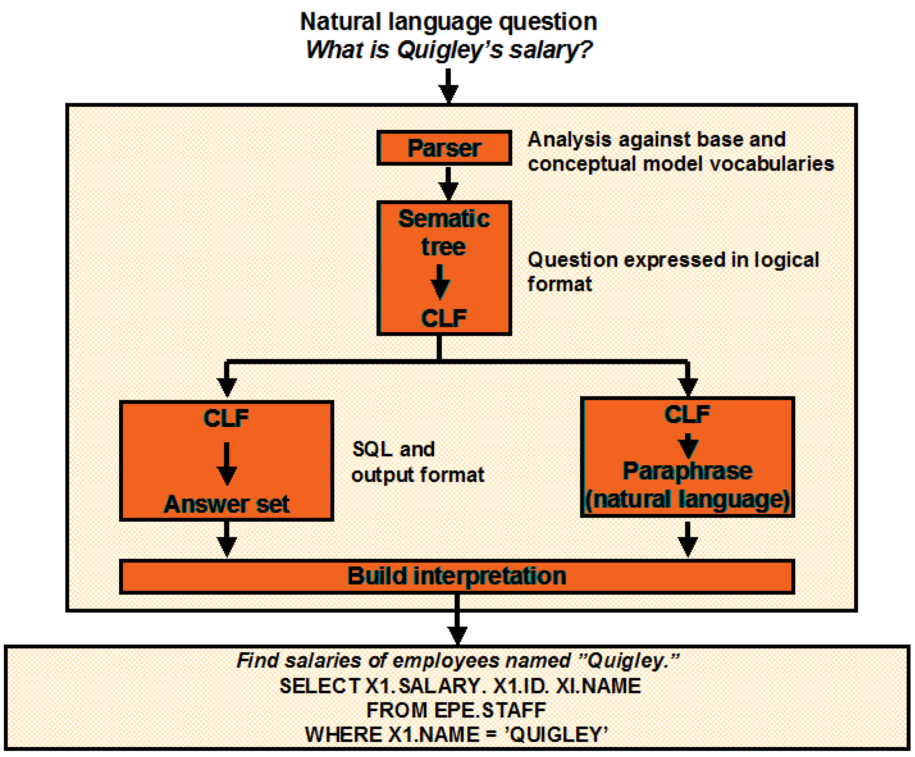

Natural language question
What is Quigley's salary?
Parser
Analysis against base and conceptual model vocabularies
Sematic tree
CLF
Question expressed in logical format
CLF
Answer set
SQL and output format
CLF
Paraphrase (natural language)
Build interpretation
Find salaries of employees named "Quigley."
SELECT X1.SALARY. X1.ID. XI.NAME
FROM EPE.STAFF
WHERE X1.NAME = 'QUIGLEY'

Figure 2 The Natural Language Engine

1.3 The Customization Tool

The Customization Tool (CT) is used to create a conceptual model by defining relations between concepts and the database tables and columns, specifying relevant natural language terms, and establishing relations between the words and the database elements. When selecting tables and columns for an application, and when processing the basic entities generated automatically by the CT, the customizer is prompted to enter information in a series of dialog boxes. The customizer selects the suggestions he wants or enters the information requested. Once this basic customization is complete, the customizer defines more complex relations in the conceptual model he or she builds.

An application in Ergo is a *conceptual model*. This conceptual model is a collection of prolog facts about the tables and columns in the database, including all information about corresponding entities, terms, and relations the NLE requires to process queries. When users ask queries, Ergo makes use of these facts to process the query and generate an answer. The task of providing Ergo with application-dependent information (i.e., an application conceptual model, which includes an application dictionary) is called customization. Applications are customized by a customizer (i.e., the one who customizes Ergo for a specific application).

A conceptual model in Ergo is used as a formal representation between the database and the language. It is connected to:

• A database through SQL-statements, and

• A language through natural language terms.

Using the Ergo CT the customizer creates a conceptual model. This model describes all the objects, which are of interest to the users. In other words, it is a model of the universe of discourse, which is selected portion of the real world or a postulated world dealt with in the application. The conceptual model is represented graphically to the customizer. The modeling technique used by Ergo is the binary ER (entity/relationship) model. This was chosen because the expressive power of the E/R approach and that it is simple and can easily address language concepts and the links between them.

The conceptual model is composed of concepts (entities) and relationships (links) between them. It is connected to the natural language terms from one side and to the database from the other side. These can either be nouns, verbs, and adjectives. Relationships are those that connect entities together which may denote possession, time, place, verb and prepositional relationships.

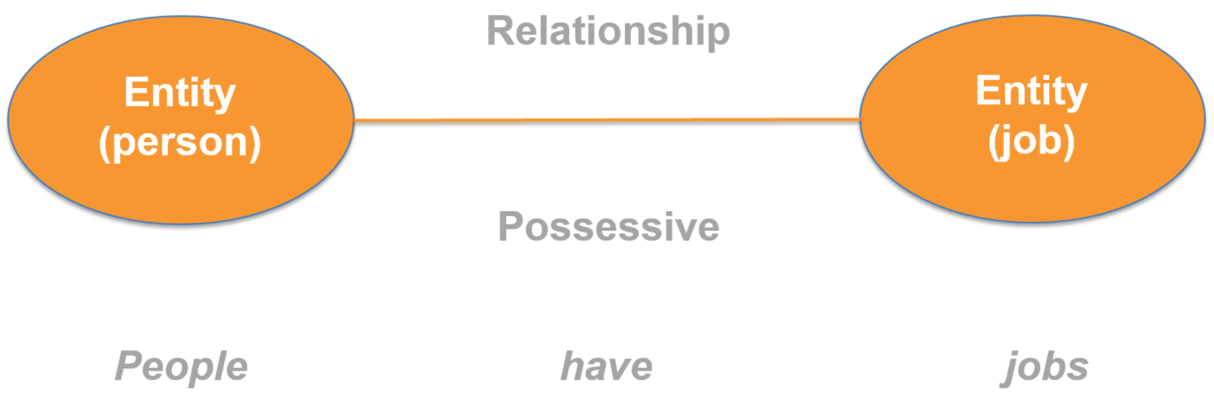


Figure 3 Binary Entity Relationship Model

In the customization process, words needed for a given application are specified in the model together with associated grammatical information. The relation of the words to each other and their relationship to information in the database is also defined. Many users can access the same customized application.

1.4 The need for Dialogue Technologies' Ergo

The need to use Ergo may vary. Ultimately there will be a host of applications and devices, which are controlled via voice or text commands. Initially two major application areas have been defined. These include:

i. Command applications: These are applications where a device, application, or service in controlled by issuing commands in natural language. This way advanced operations can be performed without the user having to learn a set of complicated commands. A special case is mobile services. Voice control using natural language is one of the strongest user interfaces for mobile applications.

ii. Customer support: Since Ergo supports asking questions in natural language the same Ergo application can service end-users both via a voice interface and a web-based query interface. The result is significant savings in customer support costs and an increased level of service. These applications include support and help-desk applications as well as information applications.

There are certainly more areas where language-based user interfaces will become a reality over time.

The examples in the rest of this document generally reflect a data-mining application in which various decision makers in a company would like to access business-critical data from the company database. A customer support application can in this context be understood as entering frequently asked questions and manuals into an SQL database and allow end-users to query that database for information.

1.5 Database-centric and corpus-centric applications

There are two main types of Ergo applications, database-centric and corpus-centric. In this context a corpus is a set of type questions and answers that describe the domain, or universe of discourse (UOD), that the application should cover. A type question is a formulation of a question requesting a specific piece of information. E.g. *How long is the ship?* represents a question about the length of a particular ship. The same question can be formulated in a number of different ways (e.g. *What is the length of the ship?, How long is the vessel?,* etc.). Each of these requests the same information and each formulation can serve as representation for the type question.

1.5.1 Database-centric applications

Ergo can be used for building database-centric applications, taking its starting point in the *answers*, i.e. the existing databases. A typical example is the Yellow Pages. The Yellow Pages contain information about names, addresses, telephone numbers, etc. usually stored in a huge database. The purpose is to make this, and only this, information available to users.

In this case the knowledge domain of the application is completely defined. In other words, the answers that the application can provide are known *a priori*. As a consequence, all different questions that can be asked to retrieve these answers are limited in number, and can, in principle, be listed. Database-centric Ergo applications can thus reach a coverage of the user questions approaching 100%. The pre-study of a database-centric application is more focused on the design of the databases and how this design can be represented in the domain model, than on how questions can be formulated. This is not surprising since the database content restricts how questions can be posed.

1.5.2 Corpus-centric applications

In corpus-centric applications Ergo takes its’ starting point in the users' *questions*, i.e. what the users actually want to know. The knowledge domain or UOD is more difficult to define in this type of applications since it is hard to anticipate *what* users would like to know and *how* they are going to ask about it. For corpus-driven applications pre-study mainly consists of:

i) Pin-pointing the knowledge domain (*what*) and

ii) Gathering as many authentic end-user formulations as possible about this domain (*how)*.

It is very important that the collected data is as authentic, and as large as possible, since it is almost impossible to cognitively foresee user questions. This data makes up the corpus.

Based on the corpus a suitable database structure is defined, including defining tables and columns in the tables.

1.5.3 Comparison between database-centric and corpus-centric applications

The main difference between the two types of applications is in how the UOD is determined and how the database structure is defined.

In the database-centric approach the database, including all data records, is given from the beginning, something which largely also defines the corpus.

In the corpus-centric approach the corpus, and the database structure, is refined in a series development stages, each involving testing on a group of users. The database is populated with data during the development process. This is described in more detail in Appendix B.

	Database centric	Corpus centric
User question	Induced from database	Empirical pre-study and study of question logs
Model	Governed by database design	Governed by user questions
Database	Given	Governed by model
Knowledge domain	Identical with content of database	Combination of initial hypothesis, pre-study and analysis of user data.

Table 1 Characteristics of the two types of applications

A particular strength of Ergo is that both types of applications can be combined.

For example, two knowledge domains belonging to database-centric and corpus-centric applications, respectively, were combined into a single customer service application. The database-centric knowledge domain contained area codes for telephony (e.g. allowing questions like *Where does 0611 go?* and *What's the area code for Härnösand?).* The underlying database contained one table with three columns with:

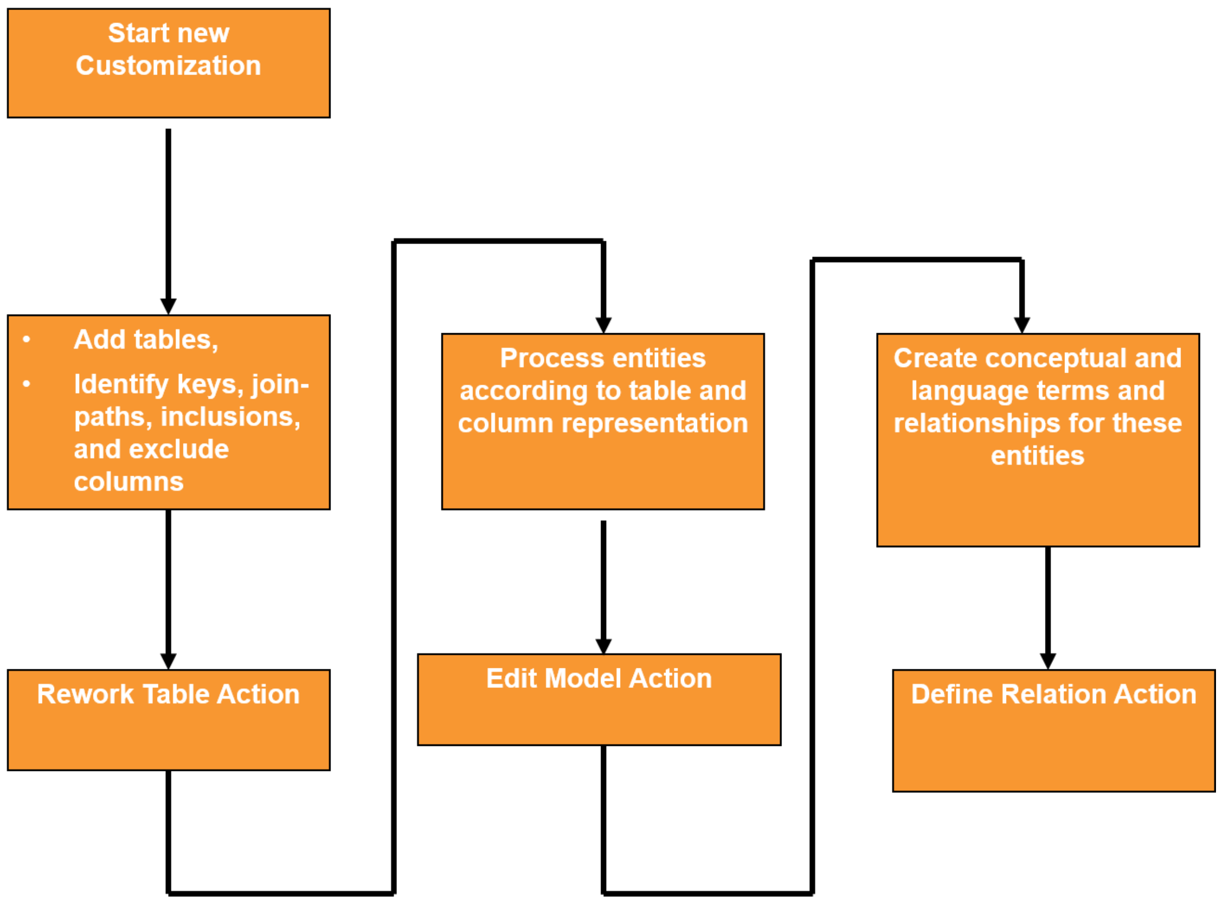
- Area code (e.g. 0611),
- Area (e.g. Härnösand) and
- Country (e.g. Sweden).

The corpus-centric knowledge domain covered user questions about telecommunication products and services (e.g. allowing questions like *Can I order*

ADSL 8.0 Mbit? and *How often does the phone bill come?)*. The structure of this database was determined by user queries.

In the combined application users could ask both about area codes and general questions about products and services, with the application querying two different databases.

2. USING THE CUSTOMIZATION TOOL


Start new Customization
• Add tables,
• Identify keys, join-paths, inclusions, and exclude columns
Rework Table Action
Process entities according to table and column representation
Edit Model Action
Create conceptual and language terms and relationships for these entities
Define Relation Action

In rework tables you specify what database structure the model has. You should specify the JOIN-PATHS and INCLUSION DEPENDENCIES that you have found in the pre-customization phase. You should specify the primary key and columns and exclude columns you do not need. You do all this using the *Mode* ➔ *DB* and *Action* ➔ *Add tables, Add columns*.	Customization links words that you are going to use in your queries to the database tables and column. You have to define these words as entities. This is done by assigning a term, classifying, defining the syntax and maybe adding SQL statements to the entity. Additional entities may need to be created if there is no data base representation for the entity *Mode* ➔ *Lang* and *Action* ➔ *Add entity*	As with any entity, additional entities must be related to other entities. Relationships are formed based on the entities classification. There are both Language and Conceptual relationships. You do this by dragging/dropping one entity on top of another in the graphical interface.

Figure 4 Customization process

2.1 Enter DB information

In the CT you enter information about the database tables and columns – this becomes part of the domain model. The engine needs to know the name of the database to be used as well as information about tables and columns.

2.1.1 Enter table information

To enter table information press *Mode* ➔ *DB* and *Action* ➔ *Add tables*.

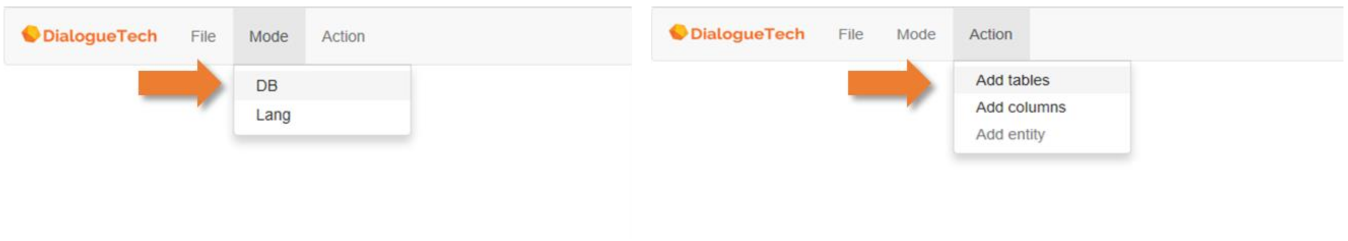


Figure 5 Opening of the "Add tables" window

When you press *Add tables* a screen appears that prompts you to enter the name of the database and tables.

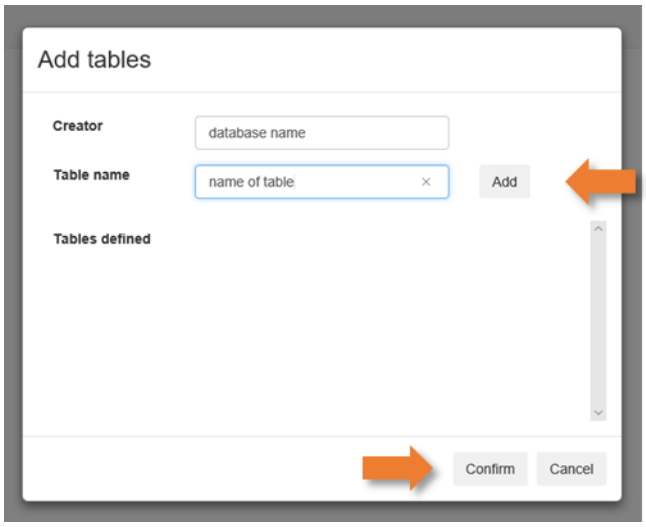


Figure 6 The "Add tables" window

Press *Add* and *Confirm* to complete the list of database tables

Once you have entered all tables you can add the columns of each table. To add the columns, you press *Action* ➔ *Add columns.* Enter the name of each column for each table – press *Add* for each new column.

Add additional information for each column – define the type of data you have in each column by double-clicking on each column name, and select the data type in the Edit column screen:

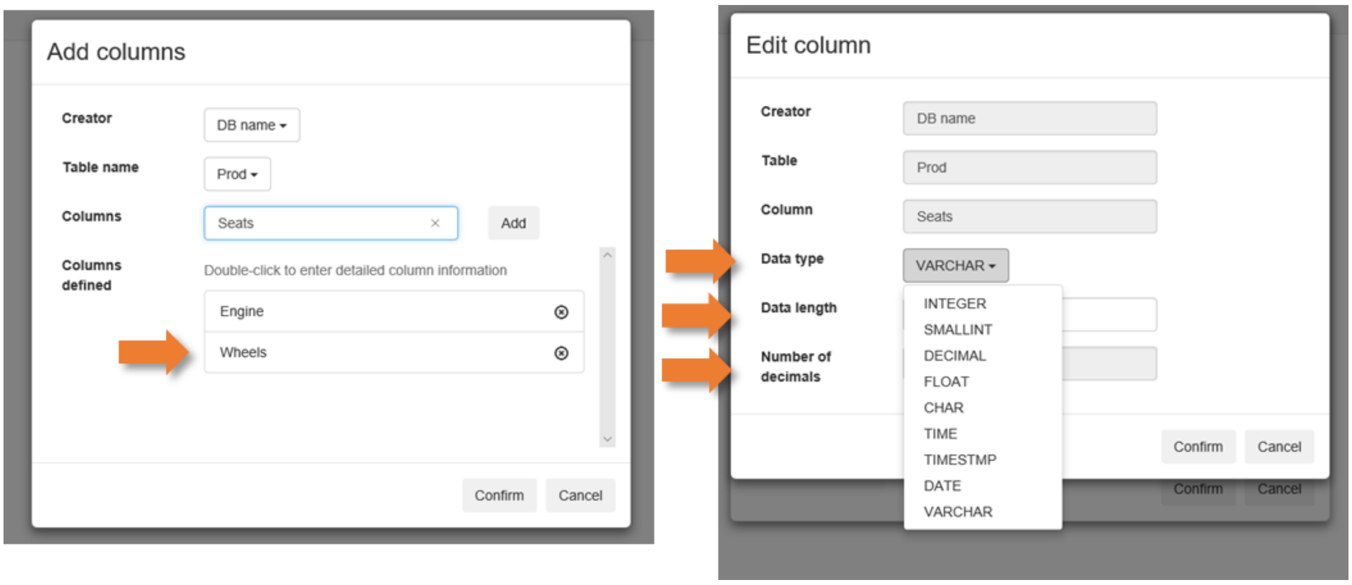


Figure 7 "Add colums"and "Edit columns" windows

The result from the example above look like:

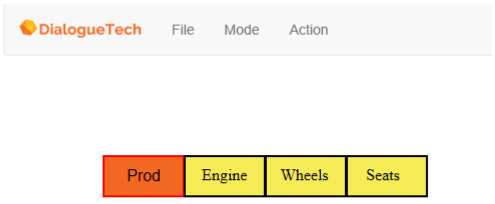


Figure 8 Graph DB mode

You can now create keys, join-paths, inclusion dependencies and exclude columns that you decided on in the previous phase. You should also exclude foreign keys and any other columns that you do not want to include in your model. Open the “Properties of columns” window by double-clicking on the column in the graph:

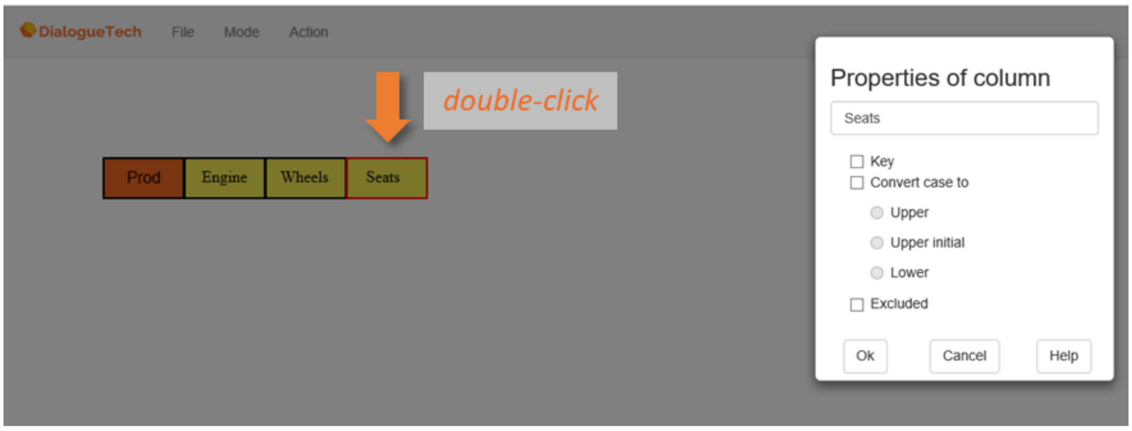


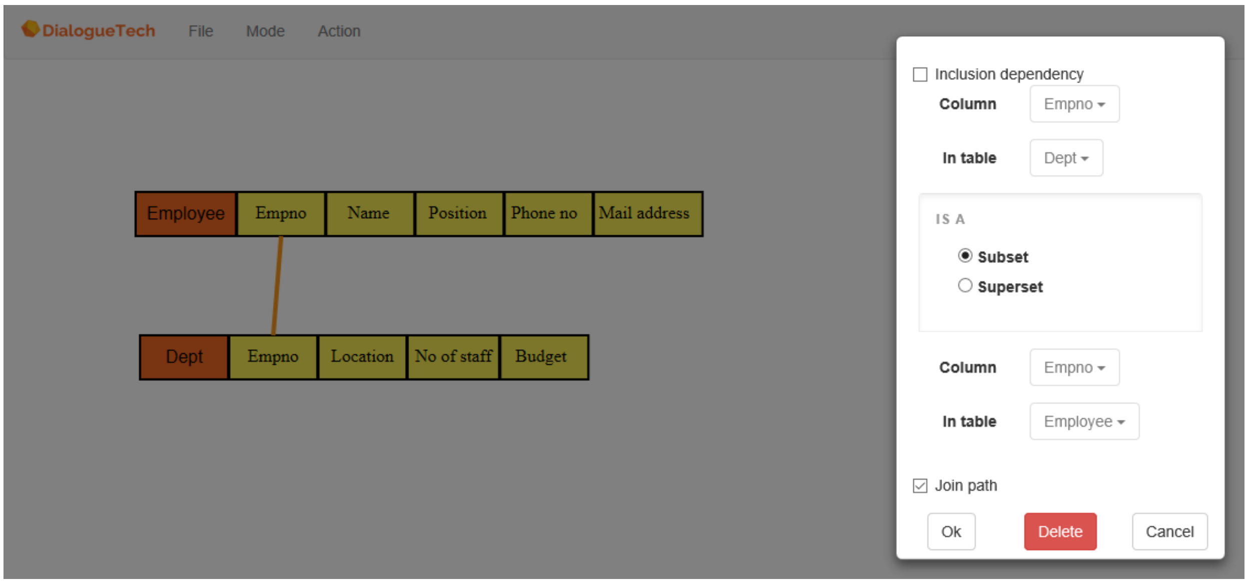
Figure 9 Defining columns and keys

DialogueTech

2.1.2 Joins

A join-path is established between the primary key of one table and its foreign key in another table. This enables the Natural Language Engine to select information from different tables in a single query. More importantly, if a concept from one table contains an attribute associated with another table, a join-path enable a relationship to be established, thus linking entities across tables. For instance, if you have the EMPNO column in two tables, e.g., EMPLOYEE and DEPT tables, then you could connect the two tables with a join-path. If an employee has a department phone number, e.g., DEPT PHONE, then this relationship can be defined.

To define a join path, click on one of the tables and drag it on top of another table – this will open the window below:



☐ Inclusion dependency

Column: Empno

In table: Dept

IS A

● Subset

○ Superset

Column: Empno

In table: Employee

☑ Join path

Figure 10 Defining joins and inclusions

Tables must only be joined with one join-path.

Except for multi-column primary key whose components are multi-column foreign keys, only one single-column join-path should be made between the two tables.

So, if these two tables, EMPLOYEE and DEPT also have a column called DEPTNO, or 'department number', you can choose which join-path to make: either the join-path between the EMPNO columns or between the DEPTNO columns, but NOT BOTH.

If you define more than one direct join-path between two tables, Ergo assumes at least one table has multiple-column primary keys. If there are more than one join-path links between a pair of tables Ergo interprets this as a multi-column join-path. For example if a table has a two-column key that can be found as a foreign key in another table with, say, a three-column primary key, you may join both columns to the corresponding columns in the second table.

Join-paths cannot be circular. If you have connected the employee table and the department table with a join-path and you have connected the department table with the address table then the employee table and the address table automatically has a connection between them, and you should not make a join-path between them. What order to connect the tables is somewhat arbitrary, but connections should be made to the most important table. So, if you are dealing with an employee application then you could connect the employee table to all the other tables like spokes in a bicycle wheel.

Concerning designing what join-paths should be the best paths let us take this hypothetical example.

Table: Insurance Policy
COL1: POLNO(Primary key)
COL2: REGION_CD
COL3: DISTRICT_CD
COL4: BRANCH_CD

Table: REGION
COL1: REGION_CD (Primary Key)

Table: DISTRICT
COL1: REGION-CD(Primary Key)
COL2: DISTRICT CD(primary Key)

Table: Branch Office:
COIL: REGION_CD (Primary key)
COL2: DISTRICT_CD(primary Key)
COL3: BRANCH CD(primary Key)

Given these insurance tables, we could link Region, District and Branch Office to Insurance Policy individually. We can go from Insurance Policy to Branch Office to District to region etc. There are various possibilities to join these tables. However, what decides the best join-path? Usually, the kind of questions and the complexity of the SQL generated dictate the choice of a join-path. If the application mostly concerns insurance policies and agents in branch offices, we might link Insurance_Policy via Branch Office. If the application concerns policies and regions, then we may link via regions. If all three are equally important, then we will link all these to Insurance_Policy.

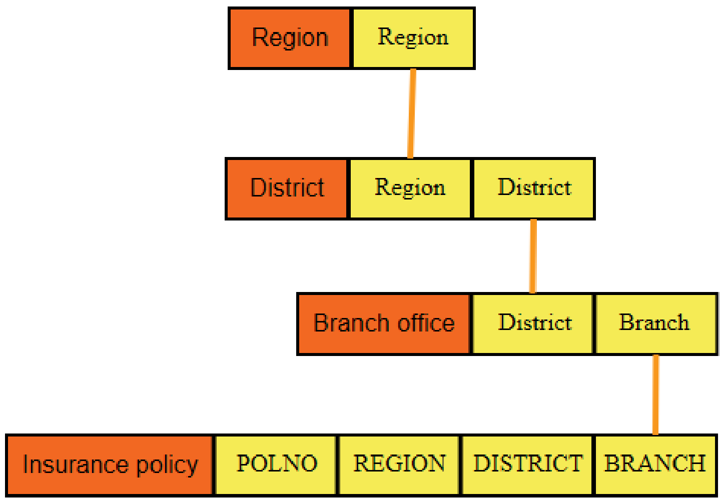


Figure 11 Example of join-path and inclusion dependencies

2.1.3 Inclusion dependencies

Primary-Key/Foreign-key relations should be scrutinized.

If the foreign-key column represents a different concept than the primary-key concept (e.g. Tab 1. employee/Tab 2. manager), then an Ergo inclusion dependency is usually specified, and the conceptual relationships of the super-type entity will be inherited by the subtype entity.

If a foreign key column represents the same concept as the primary key (e.g., Tabl_dept/Tab1dept) then customization is straightforward, and usually a join-path is set up between the tables to permit the specification of relationships between entities representing concepts related to different tables.

2.1.4 Excluding columns

In table mode, those columns that are excluded, are those, which are not part of the application, which are foreign keys columns representing the SAME concept as the primary key. An example of this is an EMPLOYEE_DEPARTMENT and DEPARTMENT_NO representing the same concept so EMPLOYEE_DEPARTMENT can be excluded. However the EMPLOYEE.EMPMNO and DEPARTMENT.MANAGER cannot be excluded because it is a subtype (subclass) of employee representing a different concept.

To exclude a column, double-click on the column graph and select “Excluded” from the “Properties of column” window.

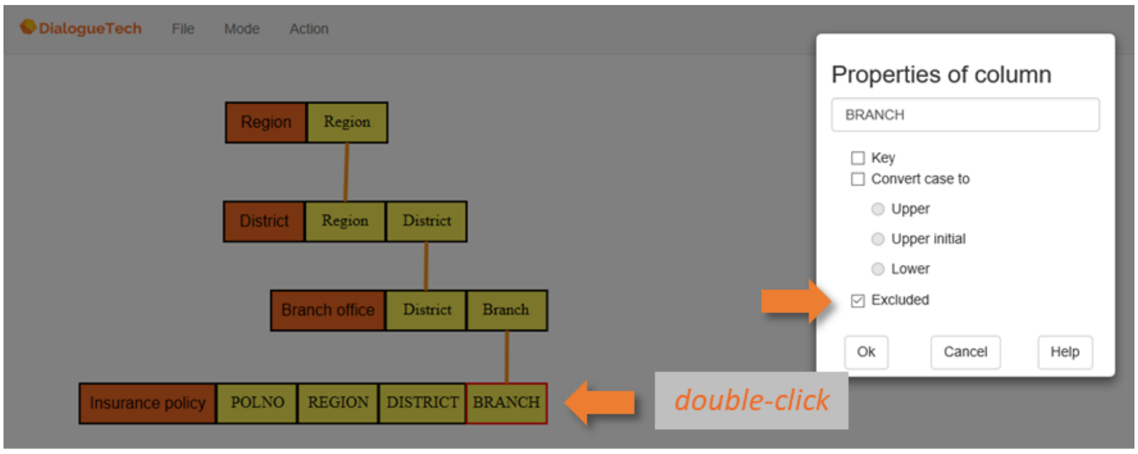


Figure 12 Exclude column

Constituents of a multi-column primary key should not be excluded.

All the work in the pre-customization phase and in rework tables should be checked before you go on. It is important that this phase is correct before you move on to edit the model. See Case 1. Joins and Inclusion Dependencies, Fine Tuning the Model.

2.2 Create entities

Use the Lang (language) Mode to create and define entities in the conceptual model. This is what you must do when creating/defining a new entity:

1. Create an entity by giving it a name
2. Classify the entity
3. Give it a language term if you want to use it in a query
4. Define the morphology of the term if you defined one.

These are the tasks you can use when you edit the conceptual model.

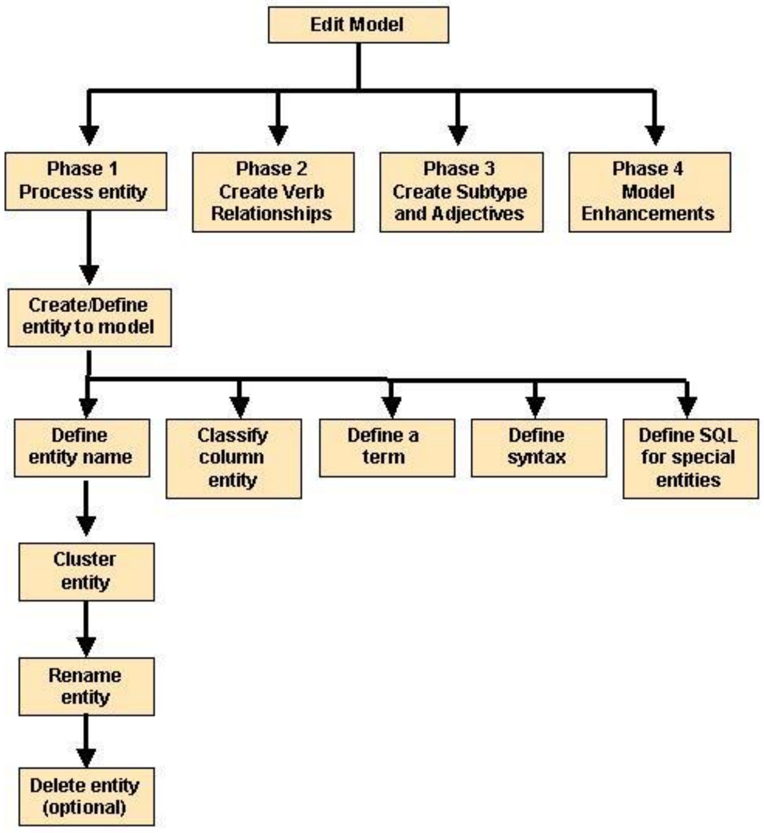

Edit Model
Phase 1 Process entity
Phase 2 Create Verb Relationships
Phase 3 Create Subtype and Adjectives
Phase 4 Model Enhancements
Create/Define entity to model
Define entity name
Classify column entity
Define a term
Define syntax
Define SQL for special entities
Cluster entity
Rename entity
Delete entity (optional)

Figure 13 Process for entity creation

2.2.1 Creating a new entity

You create new entities when defining subclasses, instances, verbs and adjectives, to enhance the conceptual model. To create a new entity, select:

Mode ➔ *Lang*
Action ➔ *Add entity*

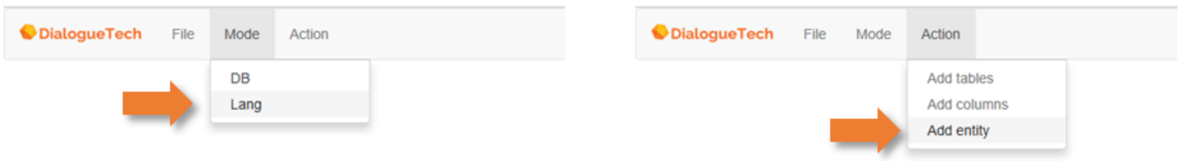


Figure 14 Opening of the "Add entity" window

Once you have opened the *Add entity* window you can enter Name of entity, classify it, add terms, syntax and SQL

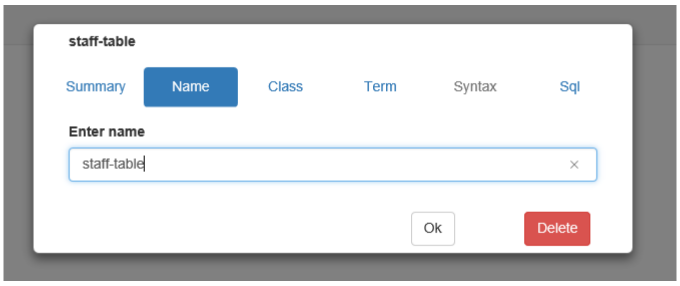


Figure 15 Add entity window

Once an entity is defined and saved it will appear in the DB-mode graph. To edit the entity you double-click on the icon.

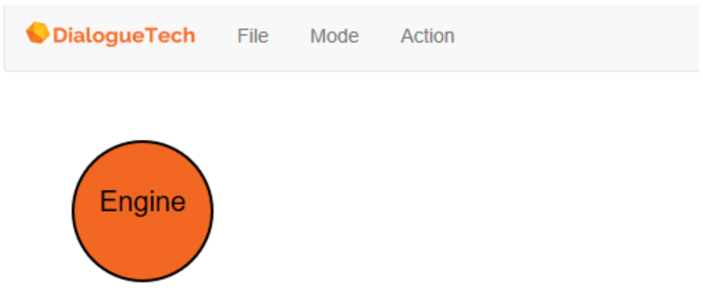


Figure 16 Graph of an (undefined) entity

2.2.2 Naming an entity

The entity window contains an empty entry field, in which you enter a name for the entity.

To change the name of an existing entity, select *Name*. The existing name appears in the entry field. Each entity name must be unique. It can contain up to 25 characters.

1. Type the entity name in the entry field.
2. Press *OK* to enter the input and remove the window or press *Class* to add classification

DialogueTech

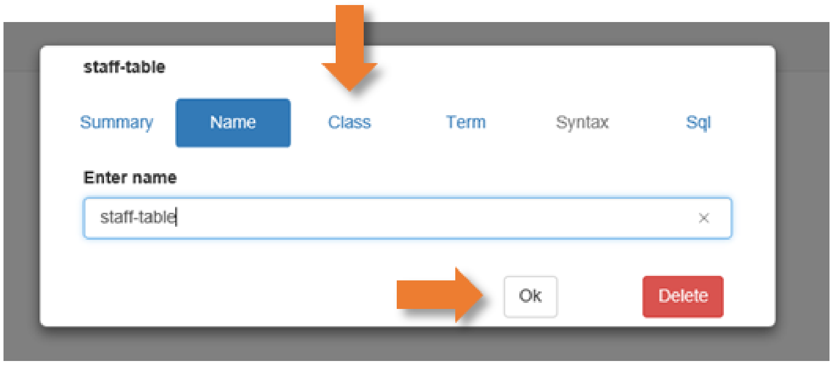


Figure 17 Specifying an entity name

2.2.3 Classifying an entity

Classify each entity in your conceptual model as a subclass or instance of at least one other class. The define classifications window shows existing classifications which are shown as a tree. The tree can be expanded and contracted.

2.2.4 Subclass and instance entities

1. Press the *Class* button to display the classification window:

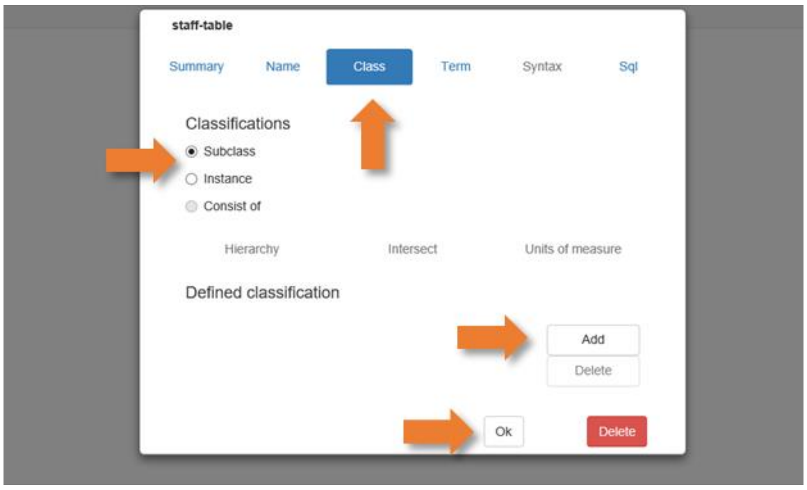
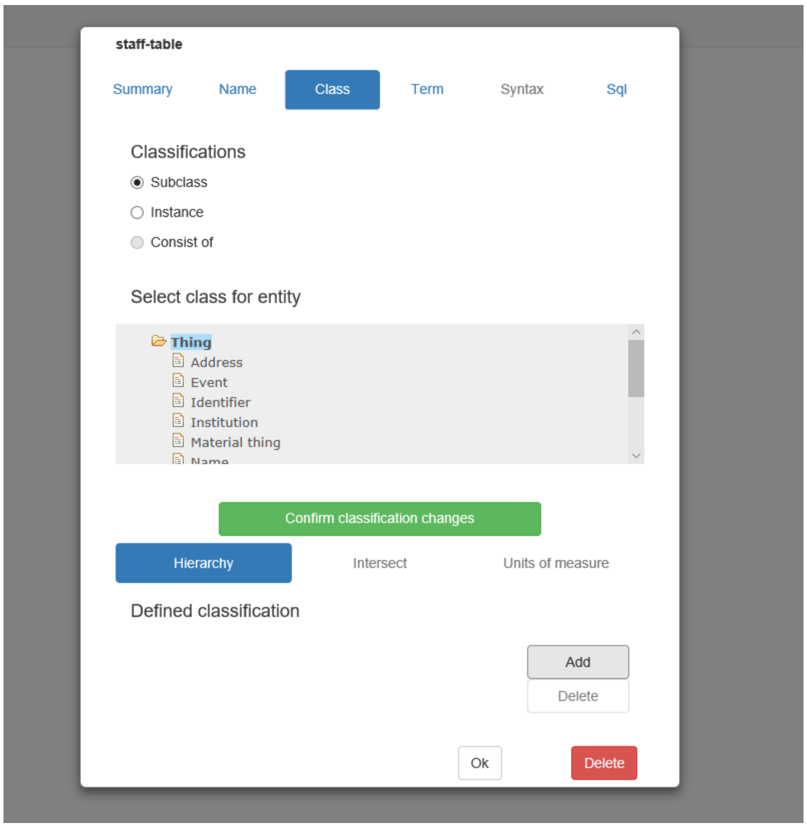


Figure 18 Entity classification window

2. Select *Subclass* or *Instance*.

3. Press *Add* ➔ *Thing* to display the hierarchy of classes:

DialogueTech


staff-table

Summary | Name | Class | Term | Syntax | Sql

Classifications

- [✓ Subclass]
- Instance
- Consist of

Select class for entity

- Thing
  - Address
  - Event
  - Identifier
  - Institution
  - Material thing
  - Name

Defined classification

Figure 19 Class hierarchy

Some classes in the hierarchy already have subclasses. This is marked by a ■ symbol. To expand the hierarchy and view the subclasses, click on the ■ symbol.

4. Select the class under which you want to add your entity.

5. Press *Confirm classification changes* to add the entity.

If the class you select is collapsed, a ■ appears in front of it to show that you can now expand that class. If you click on it, you see how your entity has been added to the hierarchy:

DialogueTech

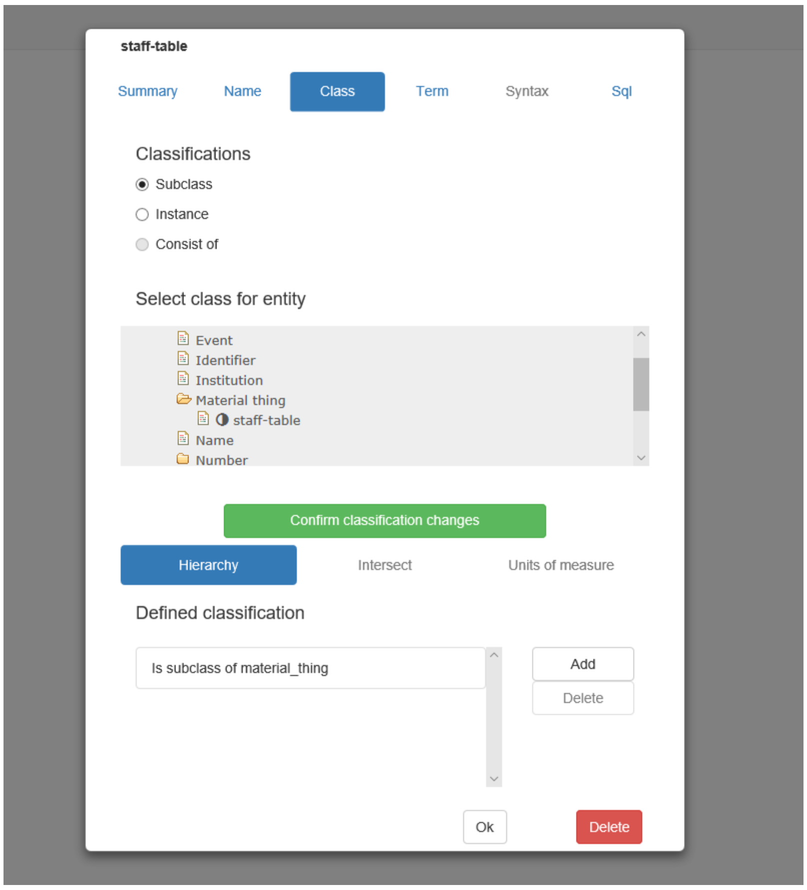


Figure 20 Adding a subclass to the hierarchy

2.2.5 Consists of

Use the *Consists of* selection if you have to define a structured entity that consists of a combination of two or more entities that have an SQL statement. As an illustration you may have an entity car and an entity fast (connected to the terms fast, quick, etc.) and define a structured entity fast car.

To classify a structured entity,
1. Press Class.
2. Add the entity as a subclass of one of the classes in the hierarchy.
3. Press OK in the Define classification window.
4. Select Consists of in the Define Entity window, and press Add.
5. Select the constituent entities in the same order as products users will use them in their queries when they enter data values of the corresponding entities. Avoid more than 3 or 4 constituents for each composite entities Otherwise questions involving composite entities may cause memory problems. To remove a selected entity, click on it once in the list in the upper part of the window. The highlighting then disappears, and the entity is removed from the *Ordered selection* list.
6. Press OK.

2.2.6 Units of measure

When you classify an entity as a subclass of *quantified_property* (or a subclass of a subclass of *quantified_property*), you can specify its unit of measure. The customization tool contains units of measure for age, area, currency duration, length, volume, and weight. When you specify a unit of measure, Ergo users can use that unit of measure in their questions, for example:

How many dollars does a chisel cost?

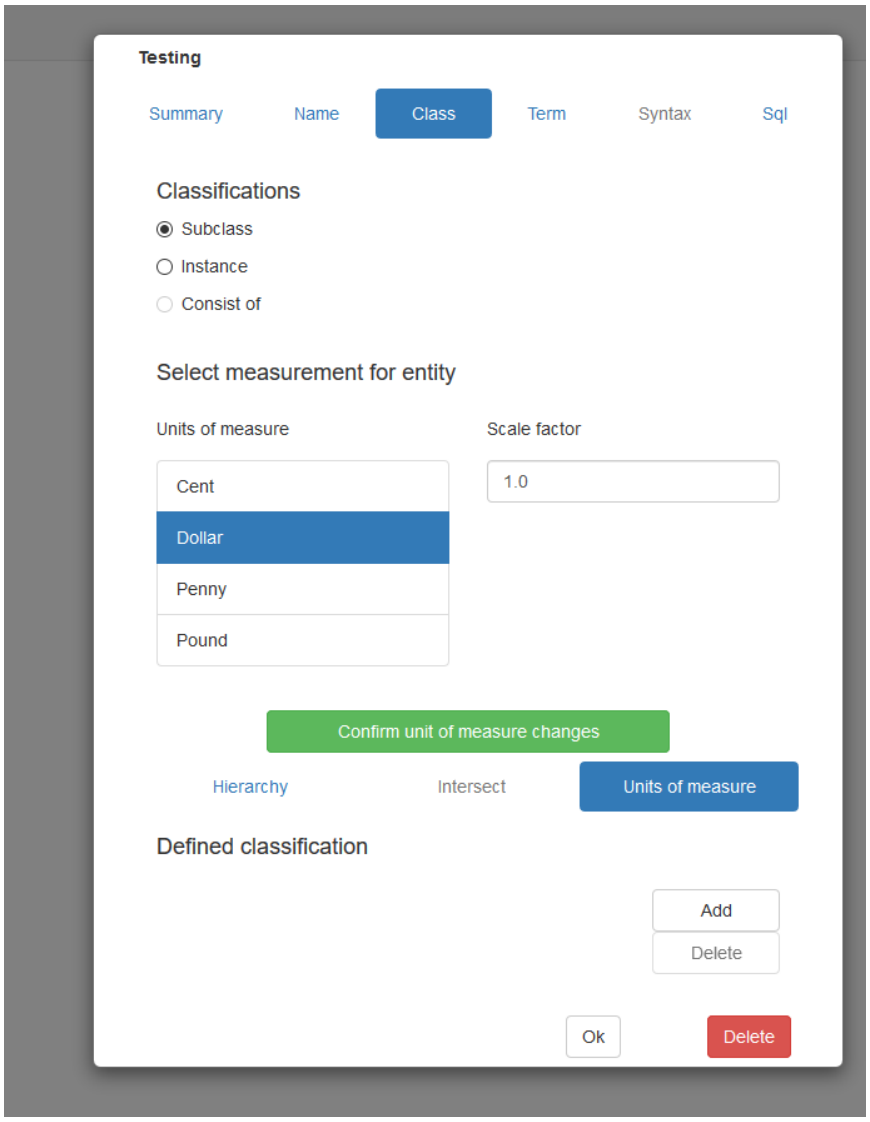
Selecting an existing unit of measure

When you classify an entity as a subclass of *quantified_property* (or a subclass of a subclass of *quantified_property*), you can specify its unit of measure. The customization tool contains units of measure for age, area, currency duration, length, volume, and weight. When you specify a unit of measure, Ergo users can use that unit of measure in their questions, for example:
How many dollars does a chisel cost?

Selecting an existing unit of measure

To select an existing unit of measure:

1. Classify the entity under *quantified_property* (for example, as a subclass of *price*).
2. Press *Confirm classification changes*.
3. Press *Unit of measure* in the *Class* window.
4. Select the unit of measure that you want from the list (for example, *dollar*).
5. Press *Confirm unit of measure changes*.


Testing

Summary | Name | Class | Term | Syntax | Sql

Classifications

(●) Subclass
( ) Instance
( ) Consist of

Select measurement for entity

Units of measure: Cent, [✓ Dollar], Penny, Pound

Scale factor: 1.0

Defined classification

Figure 21 Specifying units of measure

You can also specify a scale factor. For example, if your database contains weights measured in pounds, but users instead want to talk about ounces in their queries, select *ounce* as the unit of measure and specify *16* as the scale factor because 16 ounces = 1 pound.

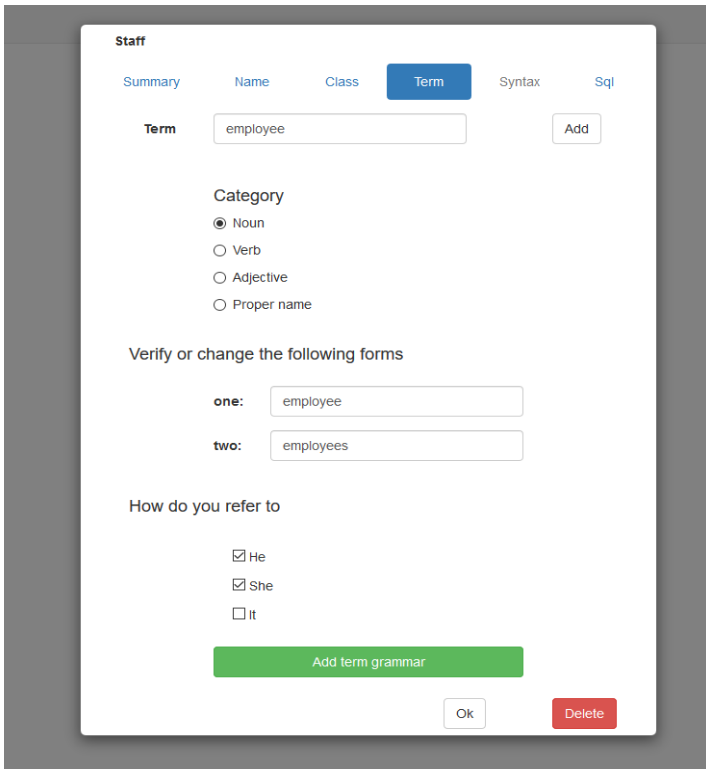
Note that the scale factor is used only to convert the unit of measure in a question so that it corresponds to the unit of measure used for data values in the column. The answer to a question is not converted.

2.2.7 Creating terms

Create one or more terms for each entity that users want to refer to. The terms you create can be either nouns, verbs, adjectives, or proper names. Each term that refers to the same entity must have the same category; if you give an entity a term that is a noun, the additional terms you create for that entity are automatically nouns. Do not confuse terms with entity names. The name is used to distinguish the entity icon in the diagram. Terms are used in questions. To create terms:

1. Press *Term* in the *Add entity* window.
2. Enter your term: for example, *employee*.
3. Select a category: *Noun*, *Verb, Adjective*, or *Proper name*.
4. Press *Add*.

If your term is a noun, verb or adjective, a window appears where you specify the grammar of the term.



Staff

Term: employee

Category
- [✓ Noun]
- Verb
- Adjective
- Proper name

Verify or change the following forms

one: employee
two: employees

How do you refer to

- [✓ He]
- [✓ She]
- It

Figure 22 Creating a term

2.2.8 Specifying syntax

In the customization tool, syntax refers to how terms are used in a question. You must specify syntax for each verb entities. Syntax is optional for noun and adjective entities.

Syntax for verbs, nouns and adjectives
In the customization tool, syntax refers to how terms are used in a question. You must specify syntax for each verb entities. Syntax is optional for noun and adjective entities.

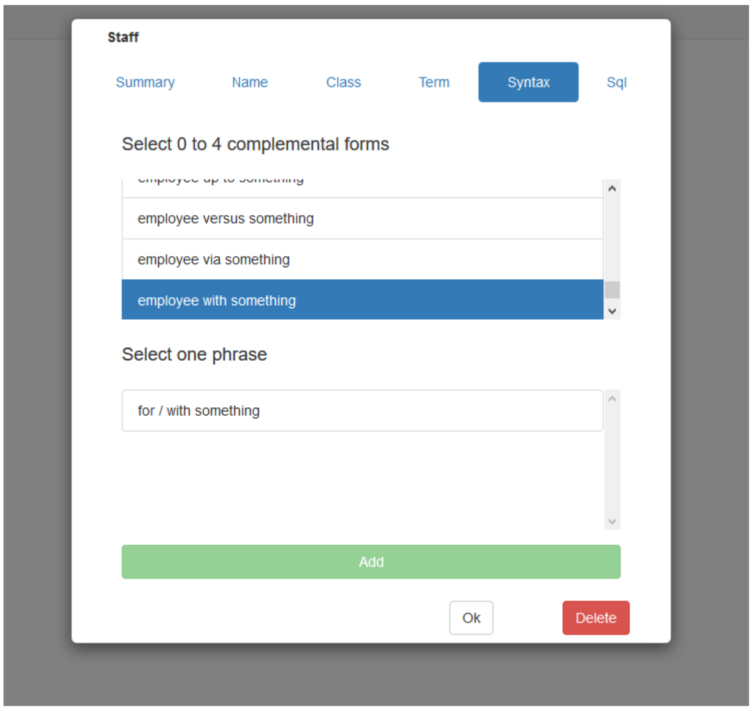
Syntax for verbs, nouns and adjectives
You can optionally specify syntax for adjectives and nouns.
1. Press *Syntax* in the Add entity window.

The Preposition complement window appears.

2. Select the prepositions that you want to use with your entity. Verb syntax is discussed in detail in **Fel! Hittar inte referenskälla. Fel! Hittar inte referenskälla.**.

3. Press *Add*

DialogueTech



Select 0 to 4 complemental forms

employee up to something
employee versus something
employee via something
employee with something

Select one phrase

for / with something

Figure 23 Adding syntax

4. If you select more than one preposition, the Syntax window appears with proposed phrases using your term.
5. Select the phrase that suits your term.
6. Press *OK*.

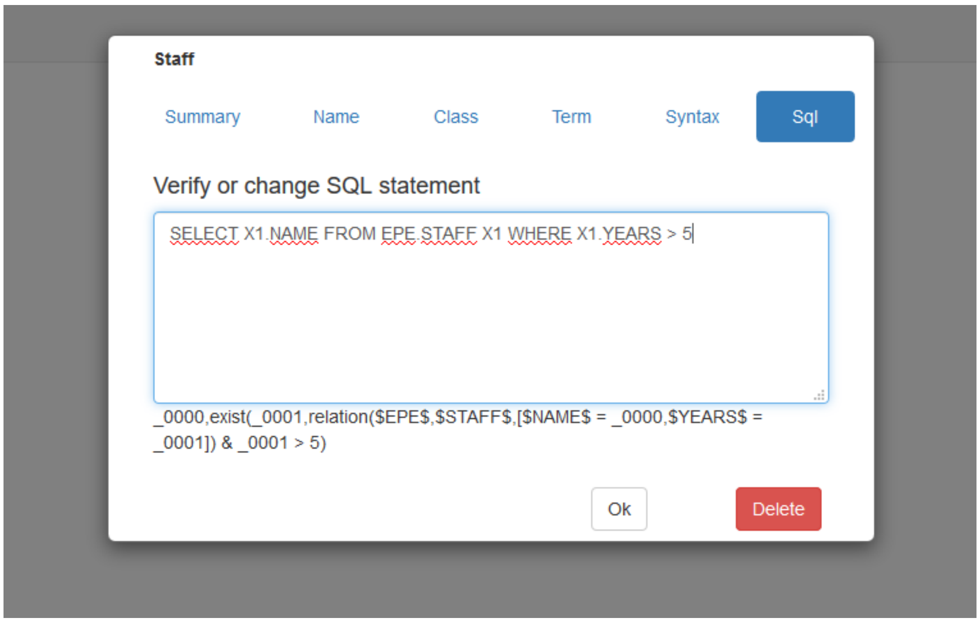
2.2.9 Specifying an SQL statement

We use SQL to define the SELECT statement for an instance, adjective, or noun entity that you create. If the column contains NULL values, you can change the SELECT statement so that the NULL values are disregarded in the answer to a query:

SELECT X1.COMM
FROM EPE.STAFF X1
WHERE X1.COMM IS NOT NULL

To specify an SQL SELECT statement:
1. Press *SQL* in the *Add entity* window.
2. Type the SELECT statement in the entry field, or copy text into the window from, for example, an application program.
3. Press *OK* to enter the input and remove the window.

DialogueTech



Verify or change SQL statement

```
SELECT X1.NAME FROM EPE.STAFF X1 WHERE X1.YEARS > 5
```

```
_0000,exist(_0001,relation($EPE$,$STAFF$,[$NAME$ = _0000,$YEARS$ =
_0001]) & _0001 > 5)
```

Figure 24 Specifying SQL

Note that if you modify the SQL statement for a column entity go to table mode, then return to entity mode, the customization tool generates a new entity with the original SQL statement. The column entity with a modified SQL statement remains unaffected in the conceptual model.

See **Fel! Hittar inte referenskälla. Fel! Hittar inte referenskälla.** on SQL considerations for entities.

2.2.10 Creating verbs

Verbs are used to describe an action and link nouns together. In creating a verb, one should perform the following:

1. Select *Add entity* from the *Action* bar
2. Give the entity a name
3. Classify the verb as event
4. Define the term and choose the verb grammar
5. Choose *Verb Syntax*

- Choose verb complements
- Choose prepositional complements
- Select the final syntax

To use the same verb with many different nouns, you can:

- Link the verb to a noun with many subclasses. (Because relationships are inherited, the verb can also be used by any noun subclasses of that noun.)
- Create two or more verb entities with the same terms. You must do this if:

- The nouns are not all in the same table or tables that are joined together (directly or indirectly).
- The verb has a different syntax or different meaning when used with different nouns. For example, if programmers *write* programs and writers *write* manuals about programs, define two verb entities with the term *write.*

DialogueTech

2.2.11 Subclasses

Subclasses define data that are a subset of the data in the database.

- A subclass is one that is wholly contained within another class already defined.

Examples:
Clerk is defined as a subclass of staff, where job is clerk.
Tools are defined as a subclass of product, where the product group is tool.

- Subclasses use SQL to select a defined subset from the database.

Defining a subclass

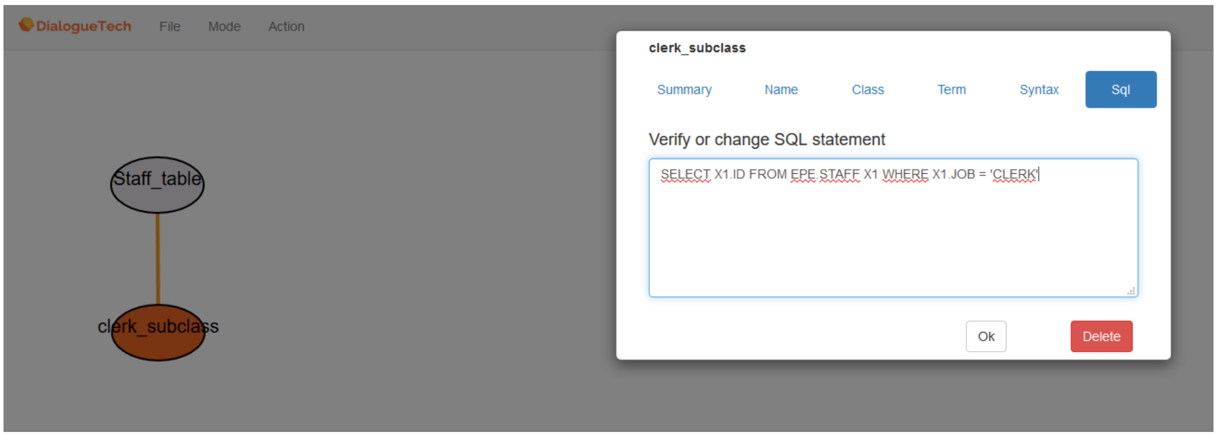
1. Create and name a new entity. You could give the name an extension of_subclass.
2. Classify the entity under the entity to which it belongs.

Examples:
Classify clerk_subclass under staff_table.
Classify tools_subclass under product_table.

3. In the *Terms* window, make the entity a noun and give it suitable terms and grammar.
4. Select any prepositional syntax required.
5. Give the entity an SQL SELECT statement that will retrieve the required data. The column selected is normally the identifier (primary key) of the concept.

Example: The SQL for clerk subclass is:
SELECT X1.ID FROM EPE.STAFF X1
WHERE X1.JOB = 'CLERK'

If the column is in another table, you need not change the default SQL for the column entity. Appropriate SQL is derived from the inclusion dependency defined in *Table* mode.



Verify or change SQL statement

SELECT X1.ID FROM EPE.STAFF X1 WHERE X1.JOB = 'CLERK'

Figure 25 Subclass example

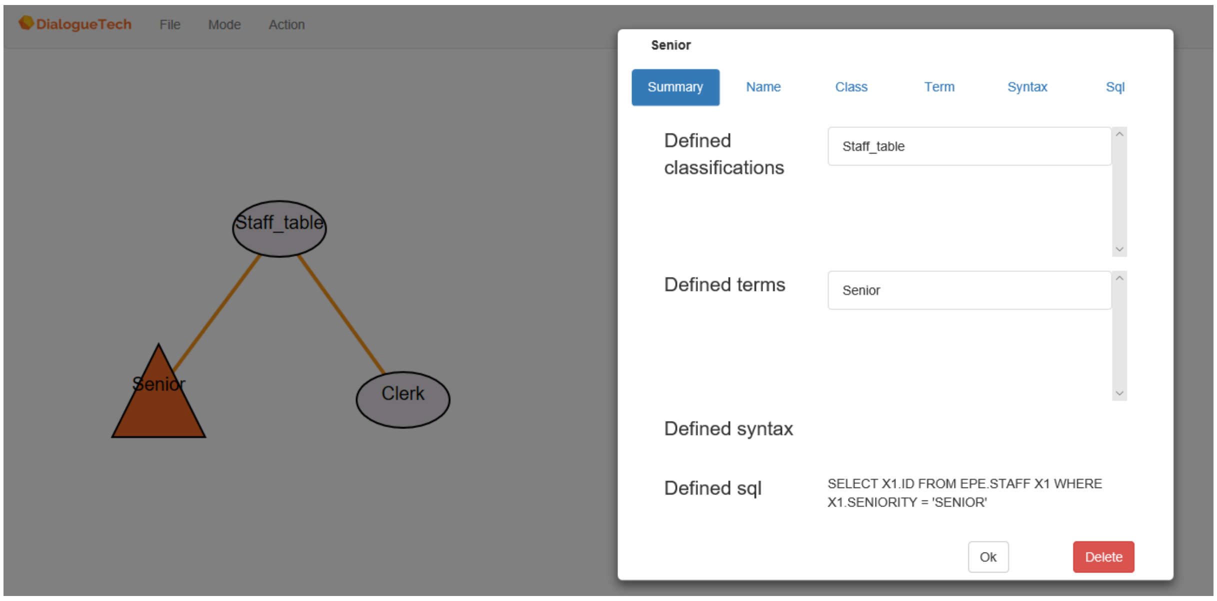
2.2.12 Creating adjectives

Adjectives in Ergo are used to select part of the data in a database. As with subclasses, this selection is done with an SQL statement.

Specific adjectives

1. Create and name an entity. No extension is needed.
2. Make the entity a subclass of the noun it qualifies.

3. In the *Terms* window, make the entity an adjective and choose suitable adjective terms and grammar.
4. Select any prepositional complements that may be required.
5. Enter the SQL that selects the required set of data.



Defined classifications: Staff_table

Defined terms: Senior

Defined syntax

Defined sql: SELECT X1.ID FROM EPE.STAFF X1 WHERE X1.SENIORITY = 'SENIOR'

Figure 26 Adjective

Subtype adjectives

If you create a compound term and define it as adjective + noun, Ergo creates a subtype adjective. Example:

If you define electrical product as adjective + noun, but without creating the adjective electrical, you can use the subtype adjective to ask questions such as *List products that are electrical.*
If you define the compound term *electrical product* as a single noun, you can only ask questions such as *List electrical products.*

For Each Subtype Adjective
1. Create a subclass entity
2. Classify the entity under the noun to be qualified
3. Give the entity a compound term (Adjective + noun)
4. Give the entity the SQL that selects the required subset.

2.2.13 Intersect

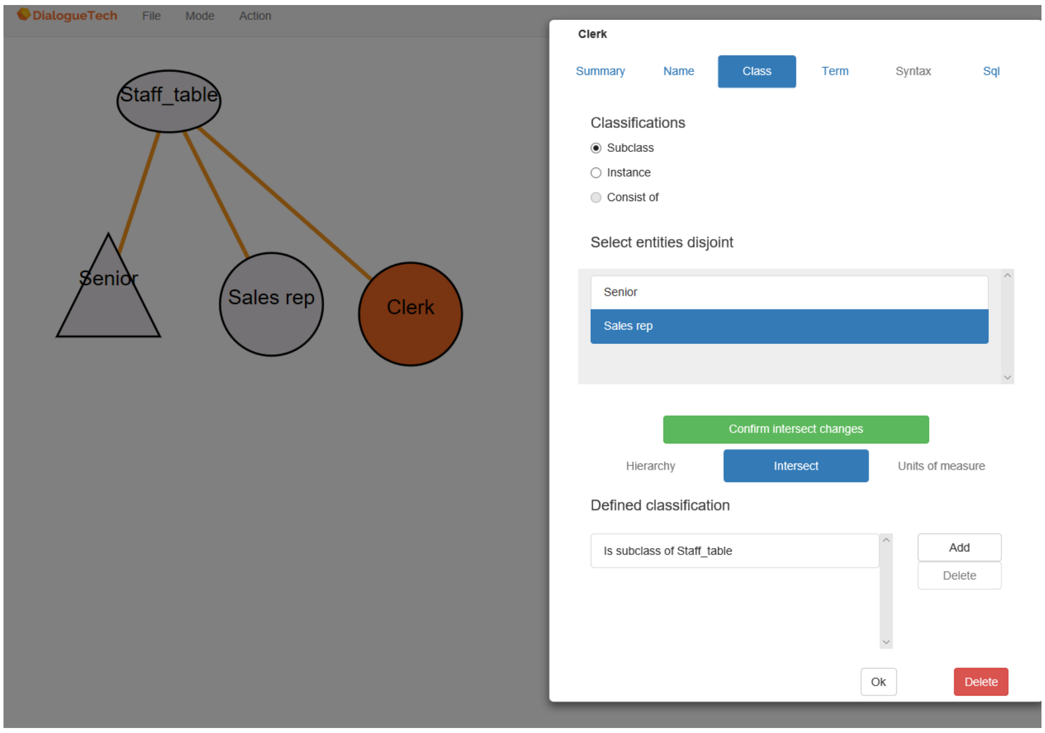
The classification window has an Intersect button that lets you select whether an entity intersects with other entities at the same classification level or is disjoint from them. Intersecting entities represent concepts that can be combined.
Example:
Senior is intersect with clerk_subclass because a person can be both a clerk and senior.

Disjoint entities represent concepts that are mutually exclusive.

Example:
Clerk_subclass is disjoint with manager_subclass because a person cannot be both a manager and a clerk.

Correct selection of intersect/disjoint improves performance.

Defining intersect entities
1. After classifying the entity, press *Apply*.
2. Press *Intersect*. A list of appropriate entities appears.
3. Select entities that are to be disjoint.
4. You only need to do this for one of the two entities involved.



Clerk

Classifications
(●) Subclass
( ) Instance
( ) Consist of

Select entities disjoint
Senior
Sales rep

Defined classification
Is subclass of Staff_table

Figure 27 The intersect window

2.2.14 Instances
Use instances to identify a specific row in a table. You may want to do this if an item in the database has more than one term.

Examples:
Head Office is also known as HQ by the users.
A generator can also be called a dynamo.
The state of Texas is referred to as TX in the database.

You need to inflect the name of a database item.
Example:
You need to refer to cam and to cams, when CAM is the product name in the database.

Creating instances

1. Create and name an entity. You could give the name an_inst extension.
2. Make it an instance of the concept to which it belongs.

Example:
Make cam_inst an instance of product_table.

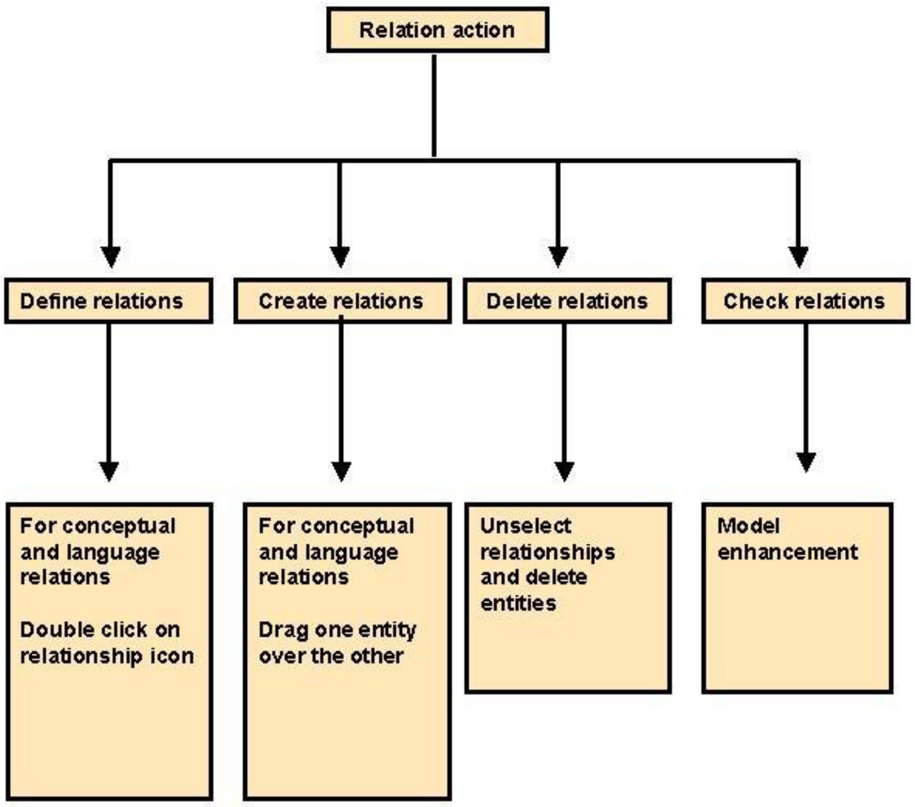
3. Select suitable terms and grammar.
4. Select prepositional complements if required.
5. Enter SQL to select the specific row, using the primary key.

Example:
To select cams, use

```
SELECT X1.PRODNUM
FROM EPE.PRODUCTS X1
WHERE X1.PRODNUM = 150
```

2.3 Creating relationships between entities

You can create language and conceptual relations. You also can delete relations. This is the way it works:


Relation action
Define relations
Create relations
Delete relations
Check relations
For conceptual and language relations
Double click on relationship icon
For conceptual and language relations
Drag one entity over the other
Unselect relationships and delete entities
Model enhancement

2.3.1 Defining relationships

A relationship is formed between two entities. Only one type of relationship can exist between two entities, however, an entity can have as many relationships as conceptually and grammatically needed. There are two types of relationships:

Conceptual relationships

1. Identifies
Used between an entity classified as an identifier and the entity it uniquely identifies. Example:
What uniquely identifies staff? employee_id
2. Names
Used between an entity classified as name and the entity it names. The name should be the column by which the row is generally described. Example:
What is the name of Staff? employee_name

Language relationships
There are additional relationships that denote possession, place, time and use of prepositions.
1. Possessive relationships - denotes possession, for example, between the manager and employee entities you define:
What does manager have? Employees

2. Place relationships - denotes location, for example, between the manager and department entities you define:
Where is the manager? Department

3. Time relationships - denotes a duration of time, for example, between project and end date entities you define:
Till when is the project? end date

4. Prepositional relationships - denotes usage of a preposition between two entities (i.e. like a prepositional phrase). For example, between the verb entity works and the entity clerk, a preposition as is defined.
Who works as a clerk

Relationships are represented by lines in the entity diagram. A solid line is drawn between two entities if

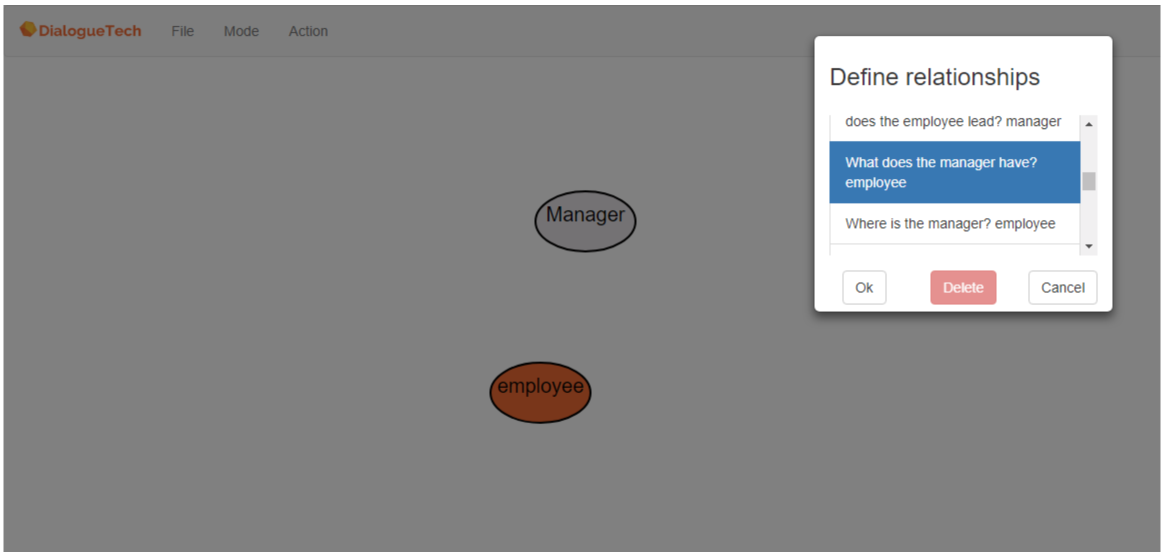
- You define a relationship between the entities
- One entity is a subclass or instance of the other
- One entity consists of several entities, including the other.

2.3.2 Creating or changing a relationship

To define a new relationship or change a relationship that you have already defined:

1. Specify that you want to define a relationship, in either of these ways:
- Hold down the right mouse button and drag one entity icon onto the other.
- If a line (dotted or solid) already exists between the two entities, double click on the line. To define a relationship between cluster icons, first expand the clusters, then double click on the relationship line.

The *Define relationships* window appears.



Define relationships

does the employee lead? manager

What does the manager have? employee

Where is the manager? employee

Figure 28 Selecting a relationship

2. The possible relationships are listed. Select the relationships that you want. The possible relationships depend upon the:
- Class of each entity
- Type of terms for each entity (noun, adjective, or verb)
- Syntax of each entity.

See Chapter 6 on Defining Relationships for more details

3. Press OK. A solid line is drawn between the two entities in the diagram.

Relationships need not be chosen from the Select relationships window when you have defined a subclass, instance or structured entities using the "consists of" function. If no relationships appear in the select relationships window, then you may have forgotten to classify or give a term to the entity.

2.3.3 Removing a relationship

To remove a relationship, double click on the line representing the relationship and deselect the relationships that are no longer valid in the Define relationships window. Alternatively – press Delete and the relationship will be erased.

2.3.4 Correcting invalid relationships.

Relationships that you previously defined may become invalid if you change the:

- Class of either entity
- Type of terms (noun, adjective, or verb) for either entity
- Syntax of either entity.

This is how you check and correct an invalid relationship:

1. Double click on the relationship line to display the Define relationships window. A selected but invalid relationships appears within brackets:

DialogueTech

[What is the name of department? location]

2. Click once on the invalid relationships to deselect it. The highlighting disappears.
3. Select any of the possible relationships that you want.
4. Press OK.