FIT2094-FIT3171 Databases
Week 6 Tutorial Activities
LOGICAL MODELLING

FIT Database Teaching Team

Complete the week 6 activities:

FIT2094-FIT3171 2021 S2
FIT2094-FIT3171 Databases
Author: FIT Database Teaching Team

Important

Remember before starting any lab activity which involves working with files (here we will be creating data models), first use SQL Developer to pull from the FITGitLab server so as to ensure your local files and the FITGitLab server files are in sync.

Learning Objective

- be able to map an ER diagram to a logical model for relational database
- be able to use SQL Developer - Data Modeler to draw a logical model, map it to the relational model and generate a schema file

After preparing a conceptual model the next stage is to select the type of database we will use (for us relational) and convert our conceptual model into an appropriate logical model. For logical modelling, we will make use of Oracle SQL Developer Data Modeler.

6.1 SQL Data Modeler Configuration

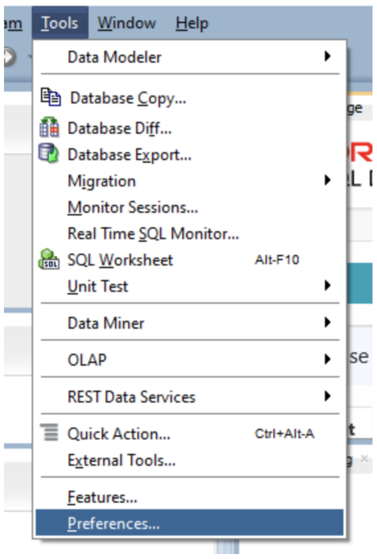
This software is a commercial level tool with an extensive set of features including support for GIT versioning and source control thus permitting teams of developers to work collectively on a design. Given the extensive range of features, *we are only going to be looking at a subset of these for our study*. SQL Developer Data Modeler begins with a (relational) LOGICAL model and then creates what it calls a RELATIONAL model from which the schema file can be generated. The relational model is essentially a graphical representation of the physical model.

6.1.1 Configuring Data Modeler

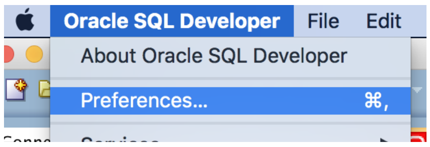
First, ensure that you have configured the required SQL Developer settings mentioned in section of 1.2.2 of the Week 1 Tutorial Activities document, especially the PL/SQL Scope Identifiers configuration.

A number of Data Modeler features must be configured using the SQL Developer preferences:

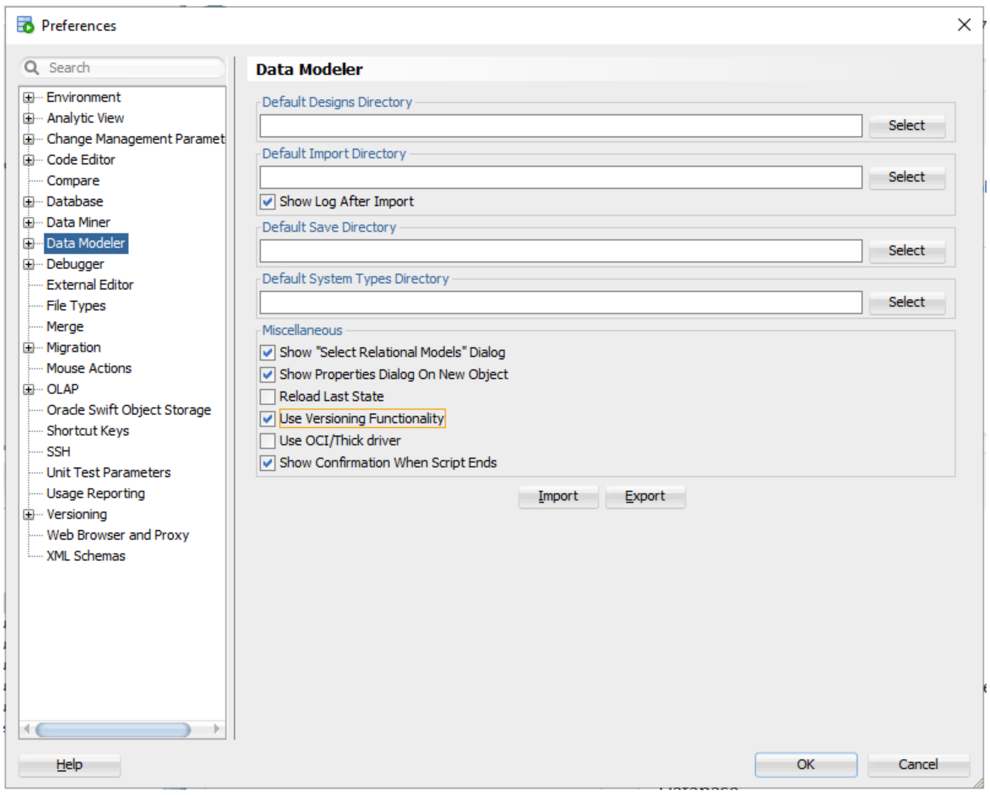
Windows/MoVE: Tools-Preferences



OSX: Oracle SQL Developer-Preference



In preferences, select the Data Modeler:



Data Modeler

Default Designs Directory

Default Import Directory

☑ Show Log After Import

Default Save Directory

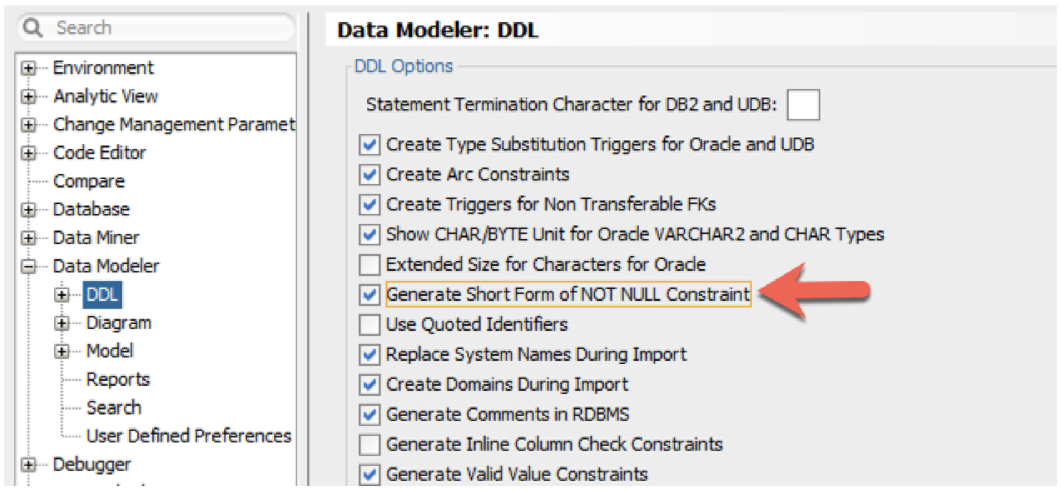
Default System Types Directory

Miscellaneous

☑ Show "Select Relational Models" Dialog
☑ Show Properties Dialog On New Object
☐ Reload Last State
☑ Use Versioning Functionality
☐ Use OCI/Thick driver
☑ Show Confirmation When Script Ends

We wish to modify three features:

(a) select DDL and select the option to "Generate Short Form of NOT NULL Constraint" - this will cause not null constraints to be not named



Data Modeler: DDL

DDL Options

Statement Termination Character for DB2 and UDB:

☑ Create Type Substitution Triggers for Oracle and UDB
☑ Create Arc Constraints
☑ Create Triggers for Non Transferable FKs
☑ Show CHAR/BYTE Unit for Oracle VARCHAR2 and CHAR Types
☐ Extended Size for Characters for Oracle
☑ Generate Short Form of NOT NULL Constraint
☐ Use Quoted Identifiers
☑ Replace System Names During Import
☑ Create Domains During Import
☑ Generate Comments in RDBMS
☐ Generate Inline Column Check Constraints
☑ Generate Valid Value Constraints

(b) Select Diagram, Relational Model and choose "Foreign Key Arrow Direction" as:

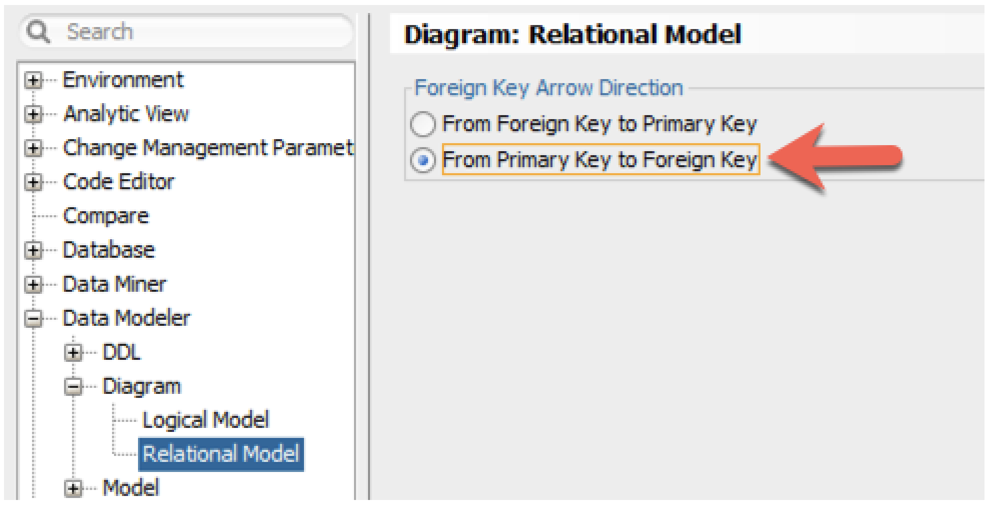
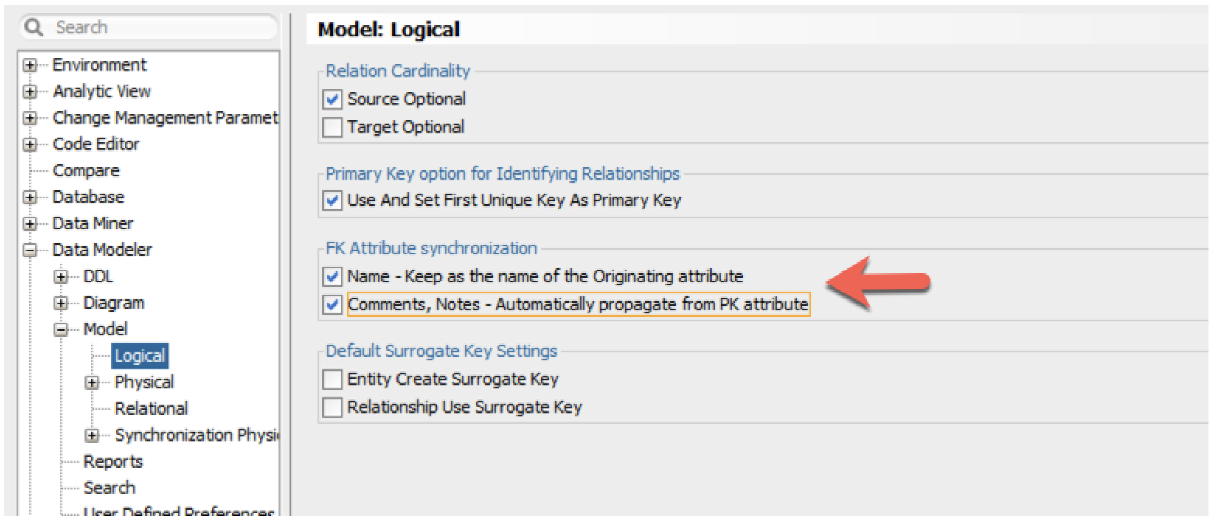


Diagram: Relational Model

Foreign Key Arrow Direction
- ( ) From Foreign Key to Primary Key
- (●) From Primary Key to Foreign Key

(c) select Model, Logical and check both items in "FK Attribute synchronization"



Model: Logical

Relation Cardinality
- [✓] Source Optional
- [ ] Target Optional

Primary Key option for Identifying Relationships
- [✓] Use And Set First Unique Key As Primary Key

FK Attribute synchronization
- [✓] Name - Keep as the name of the Originating attribute
- [✓] Comments, Notes - Automatically propagate from PK attribute

Default Surrogate Key Settings
- [ ] Entity Create Surrogate Key
- [ ] Relationship Use Surrogate Key

There are a large number of other settings available to configure, you might like to investigate these if you are using your own copy of SQL Developer, for the tutorials we will leave the remaining settings at the default values.

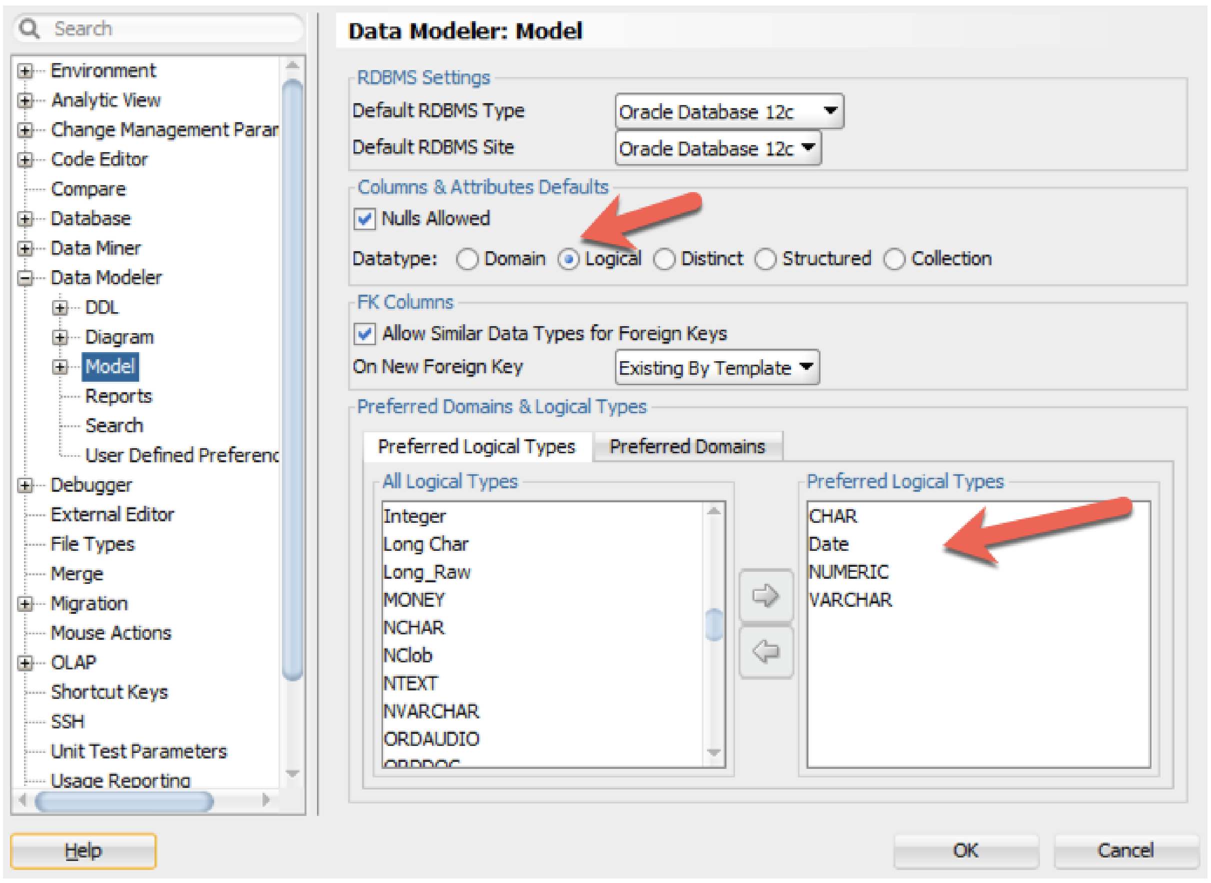
6.1.2 Configuring Data Types

Attributes on the logical model have a number of possible data types, the main ones for us being "Domain" and "Logical":

- Domain types are domains that you create via the menu items - Tools - Data Modeler - Domains Administration. No domain types are supplied, you must create any you need
- Logical types are not actual data types - they are names that are mapped to native types at a later stage. These logical types are pre-populated with several Oracle types

For our work we will *not* make use of domain types, instead, you should **always use logical types**. You can speed up the entry of attributes by restricting Data Modeler to logical types and a preferred range of types.

In your SQL Developer preferences set:



Data Modeler: Model

RDBMS Settings

Default RDBMS Type: Oracle Database 12c

Default RDBMS Site: Oracle Database 12c

Columns & Attributes Defaults

[✓] Nulls Allowed

Datatype: Domain [✓ Logical] Distinct Structured Collection

FK Columns

[✓] Allow Similar Data Types for Foreign Keys

On New Foreign Key: Existing By Template

Preferred Domains & Logical Types

Preferred Logical Types | Preferred Domains

All Logical Types	Preferred Logical Types
Integer	CHAR
Long Char	Date
Long_Raw	NUMERIC
MONEY	VARCHAR
NCHAR	
NClob	
NTEXT	
NVARCHAR	
ORDAUDIO	

The 'Preferred Logical Types' are populated by selecting the item in the 'All Logical Types" and clicking on the right-pointing arrow between All and Preferred.

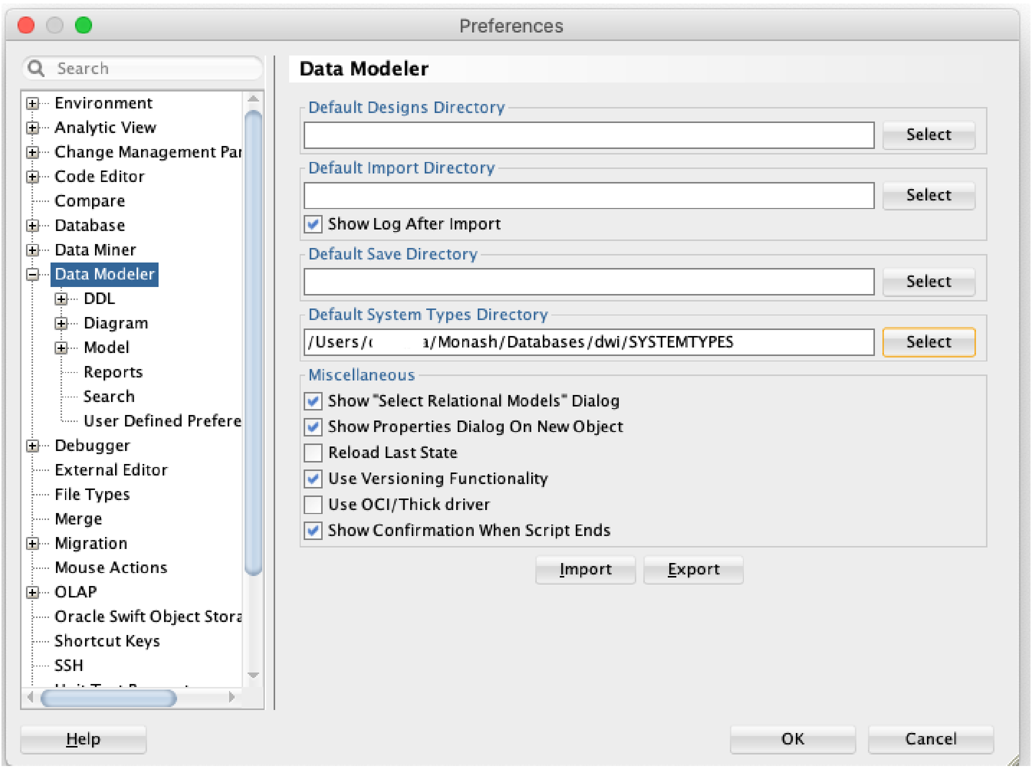
6.1.3 Configuring System Types Directory

To set up the System Types Directory:

- In file explorer/finder, locate the folder called SYSTEMTYPES which was created when you cloned your local repo. This folder will be located under:
    - **C:\Monash\**UNITS\Database\yourLocalRepo\ on Windows
    - **/users/**username/**Monash/**UNITS/Database/yourLocalRepo/ on Mac
    - \Documents\UNITS\Database\yourLocalRepo\ on MoVE

- In the Data Modeler preferences point the "Default System Types Directory" to this folder by clicking "Select" and navigating to your System Types folder (as above), then click Select.

  Double-check that the path listed is the correct path to your SYSTEMTYPES folder. If it is correct click OK


Preferences

Data Modeler

Default Designs Directory:

Default Import Directory:

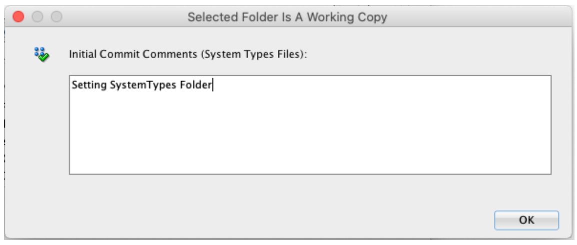
☑ Show Log After Import

Default Save Directory:

Default System Types Directory: /Users/[illegible]/Monash/Databases/dwi/SYSTEMTYPES

Miscellaneous
- ☑ Show "Select Relational Models" Dialog
- ☑ Show Properties Dialog On New Object
- ☐ Reload Last State
- ☑ Use Versioning Functionality
- ☐ Use OCI/Thick driver
- ☑ Show Confirmation When Script Ends

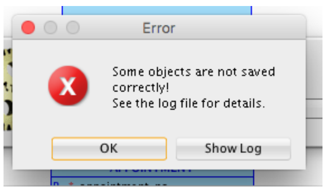
SQL Developer will prompt with a commit window. Key in a proper commit message, such as show below, and click OK.


Selected Folder Is A Working Copy

Initial Commit Comments (System Types Files):

Setting SystemTypes Folder

Later on, if you see this error message when you save your model:



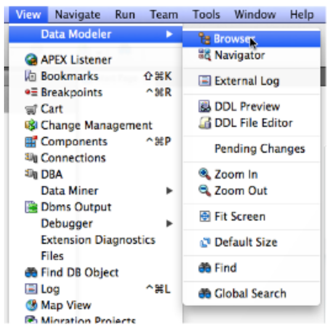
Some objects are not saved correctly!
See the log file for details.

you have not correctly set up the Default System Types Directory. Data Modeler **must be able to write to the logical data types located in the Default System Types Directory you defined above when it saves a project**. The error message appears if the default System Types Directory is a location Data Modeler cannot write to (e.g. c:\program files). Please follow the steps listed above and reset your Default System Types Directory.

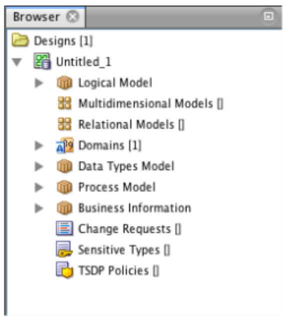
6.2 Using SQL Developer Data Modeler

6.2.1 Accessing Data Modeler

Data Modeler is installed as part of the standard SQL Developer installation. To work with Data Modeler it is **important to always have the Data Modeler browser open** in the left panel of SQL Developer. To open this panel select View - Data Modeler - Browser:

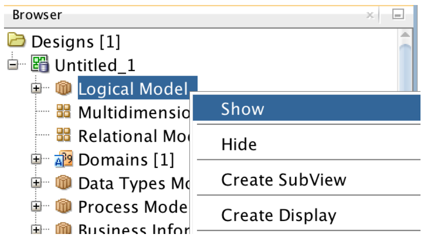


The browser opens with a new (unnamed) model:

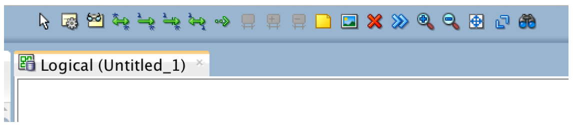


If the browser does not open, exit SQL Developer (File → Exit), then reopen SQL Developer and immediately click View - Data Modeler - Browser.

To begin creating a logical model, right-click Logical Model and select Show.



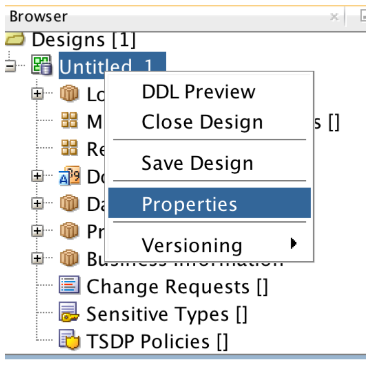
This will open a blank model in the main working panel of SQL Developer and add a range of new icons to the main toolbar. Hover over each of these new icons to become familiar with what they represent.



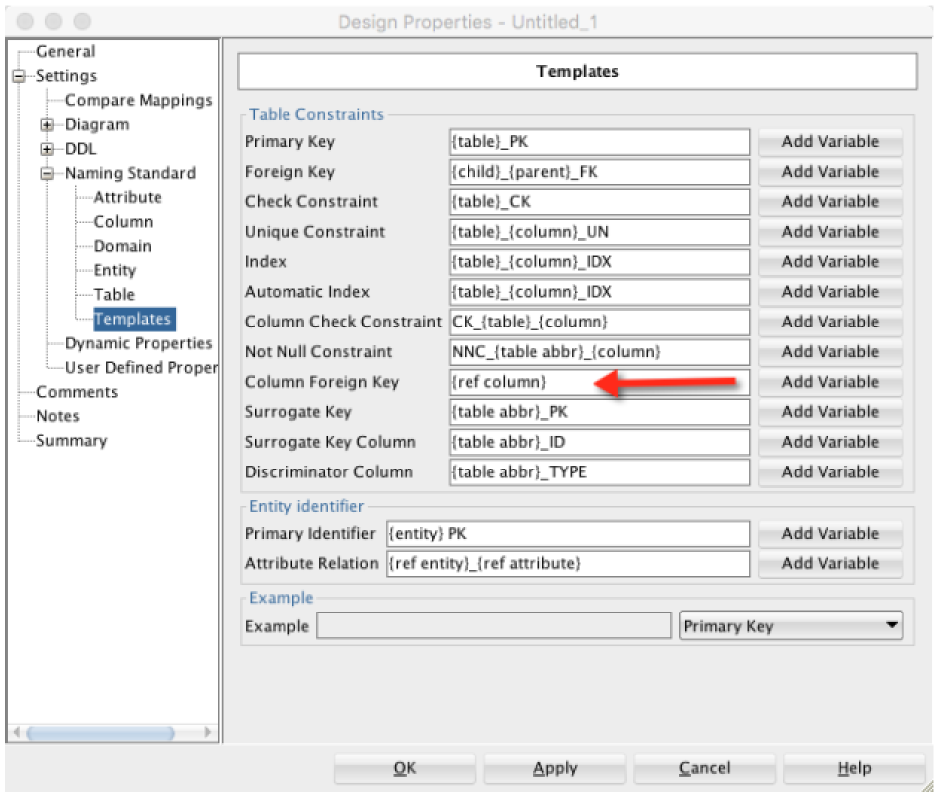
When using Data Modeler it is very important that you save and close your model before exiting SQL Developer or shutting your laptop. Failure to do so may result in loss of parts of your model.

6.2.2 Configuring Logical Model Project

Every time you work on a new project you must set the new project property such that the foreign key name is being named based on the PK attribute they were copied from. First right-click the project in the Data Modeler browser and select properties.



Within the Design Properties sheet select Settings - Naming Standard - Templates and modify the "Column Foreign Key" setting from {ref table}_{ref column} to {ref column}:



Templates

Table Constraints

Primary Key	{table}_PK
Foreign Key	{child}_{parent}_FK
Check Constraint	{table}_CK
Unique Constraint	{table}_{column}_UN
Index	{table}_{column}_IDX
Automatic Index	{table}_{column}_IDX
Column Check Constraint	CK_{table}_{column}
Not Null Constraint	NNC_{table abbr}_{column}
Column Foreign Key	{ref column}
Surrogate Key	{table abbr}_PK
Surrogate Key Column	{table abbr}_ID
Discriminator Column	{table abbr}_TYPE

Entity identifier

Primary Identifier	{entity} PK
Attribute Relation	{ref entity}_{ref attribute}

Example

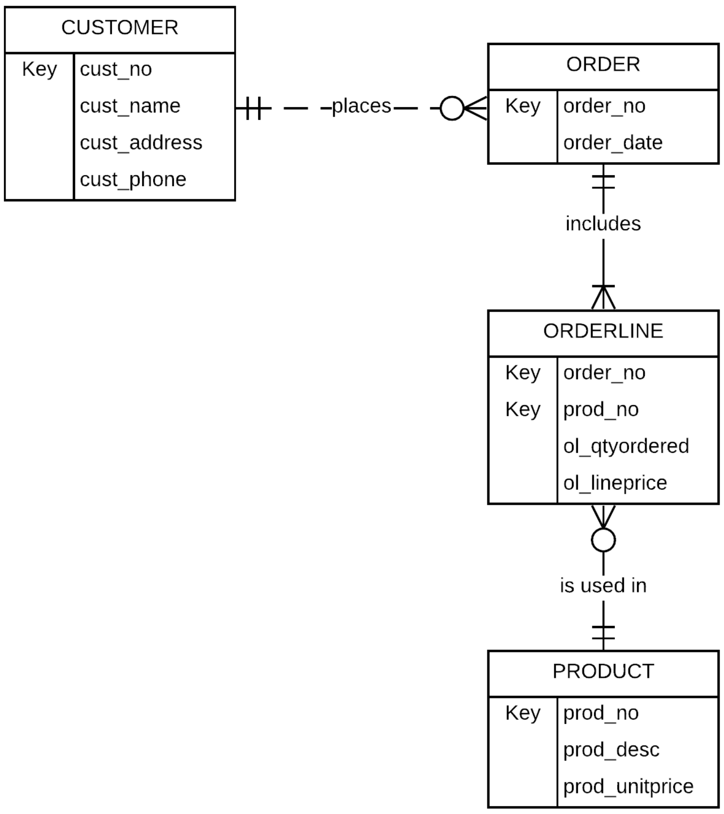
Example		Primary Key

This will result in FK's being named based on the PK attribute they were copied from. If this is not set the FK names will have the form TABLE_attribute - e.g. CUSTOMER_custno, we wish the FK to be simply custno. Note that you must do this for every new project you create.

6.2.3 Develop a model - Stage 1 The Logical Model

For your first model, we will implement a Customer-Orders system, represented by the following entities, where customers place orders for products. This is basically the conceptual model you created in the week 3 tutorial and normalisation done in week 4 workshop translated to a logical model.

Customer-Order conceptual model (taken from Week 3 Tutorial Sample Solution)



Customer-Order 3NF (taken from Week 4 Workshop)

ORDER (<u>order_no</u>, order_date, cust_no)
ORDER_PRODUCT (<u>order_no</u>, <u>prod_no</u>, ol_qtyordered, ol_lineprice)
PRODUCT (<u>prod_no</u>, prod_desc)
CUSTOMER (<u>cust_no</u>, cust_name, cust_address)

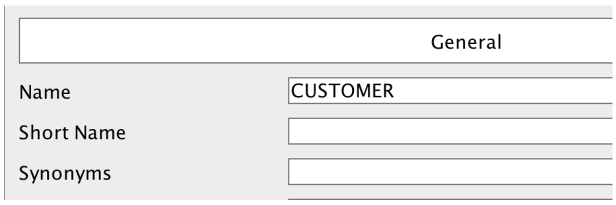
Further discussion with the client has revealed the following:

- CUSTOMERs are grouped into three different levels based on the total value of the orders they have placed in the previous financial year - Gold, Silver and Bronze
- PRODUCTs are categorised into Dairy, Household, Cleaning, Frozen, Fruit and Vegetables, and Bakery. Flexibility is required in the design so as to be able to add new and remove current product categories as circumstances change
- The client wishes to keep the name and address as simple attributes.

a. Adding Entity/Relation

On your logical model add an entity, click the entity icon and click anywhere in the main working panel.

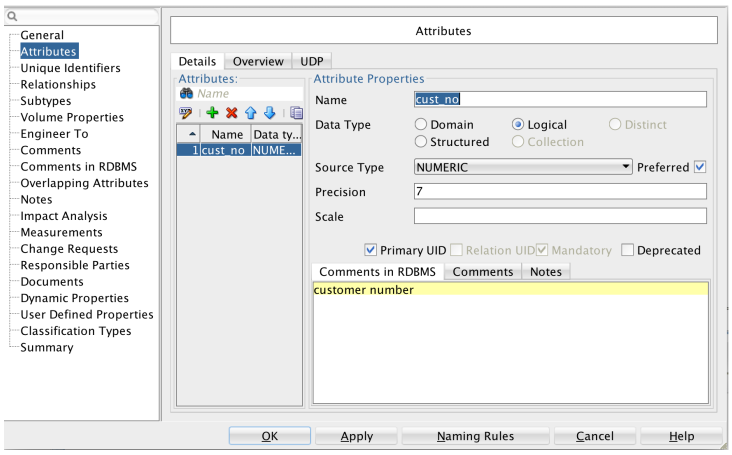
Named CUSTOMER (name the entity under the General features in the Entity Properties dialogue box) and then select the attributes feature.


General

Name	CUSTOMER
Short Name	
Synonyms	

Add the following attributes (click green + button to add another attribute):

- cust_no - Logical type: Numeric Precision 7, Scale 0, Primary UID
- cust_name - Logical type: VARCHAR(50), Mandatory
- cust_address - Logical type: VARCHAR(50), Mandatory
- cust_phone - Logical type: CHAR(10)


General
Attributes
Unique Identifiers
Relationships
Subtypes
Volume Properties
Engineer To
Comments
Comments in RDBMS
Overlapping Attributes
Notes
Impact Analysis
Measurements
Change Requests
Responsible Parties
Documents
Dynamic Properties
User Defined Properties
Classification Types
Summary

Attributes

Details | Overview | UDP

Attributes:

	Name	Data ty...
1	cust_no	NUME...

Attribute Properties

Name	cust_no
Data Type	○ Domain [✓ Logical] ○ Distinct ○ Structured ○ Collection
Source Type	NUMERIC — Preferred ✓
Precision	7
Scale	

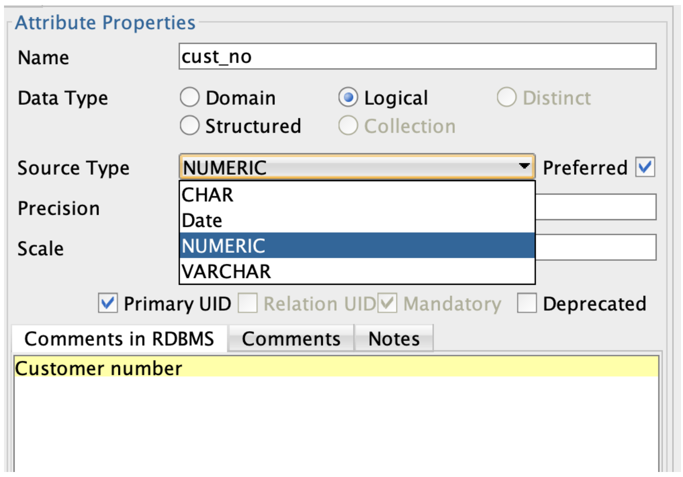
✓ Primary UID ☐ Relation UID ✓ Mandatory ☐ Deprecated

Comments in RDBMS | Comments | Notes

customer number

OK | Apply | Naming Rules | Cancel | Help

Note 1: you must check the 'Preferred' tick box so that you only see your selected range of logical types that you set on page 5:


Attribute Properties

Name	cust_no
Data Type	○ Domain [✓ Logical] ○ Distinct ○ Structured ○ Collection
Source Type	NUMERIC — Preferred ✓
Precision	
Scale	

CHAR
Date
NUMERIC
VARCHAR

✓ Primary UID ☐ Relation UID ✓ Mandatory ☐ Deprecated

Comments in RDBMS | Comments | Notes

Customer number

Note 2: for each attribute add a meaningful description of the attribute under the attribute option - "Comments in the RDBMS". **Comments in the RDBMS are required for ALL data models in this unit**.

Then add an ORDERS entity with attributes:

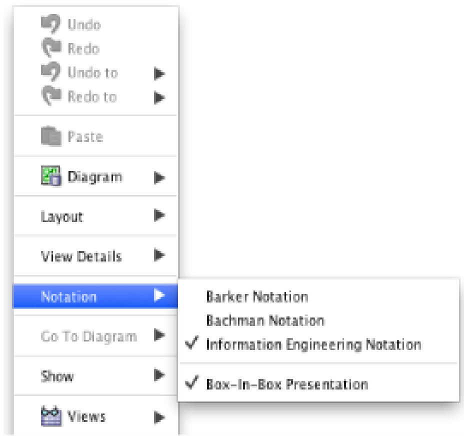
- order_no - Logical type: Numeric Precision 7, Scale 0, Primary UID
- order_date - Logical type: DATE, Mandatory

Note here that the entity is being named as ORDERS since ORDER is a reserved SQL word - remember that as a general rule entity names must not be pluralised.

b. Configuring Logical Model Notation

For all the logical models you create you MUST set the following (right-click on the **blank area** in the logical model and then select these options):

- Change Notation to "Information Engineering Notation":

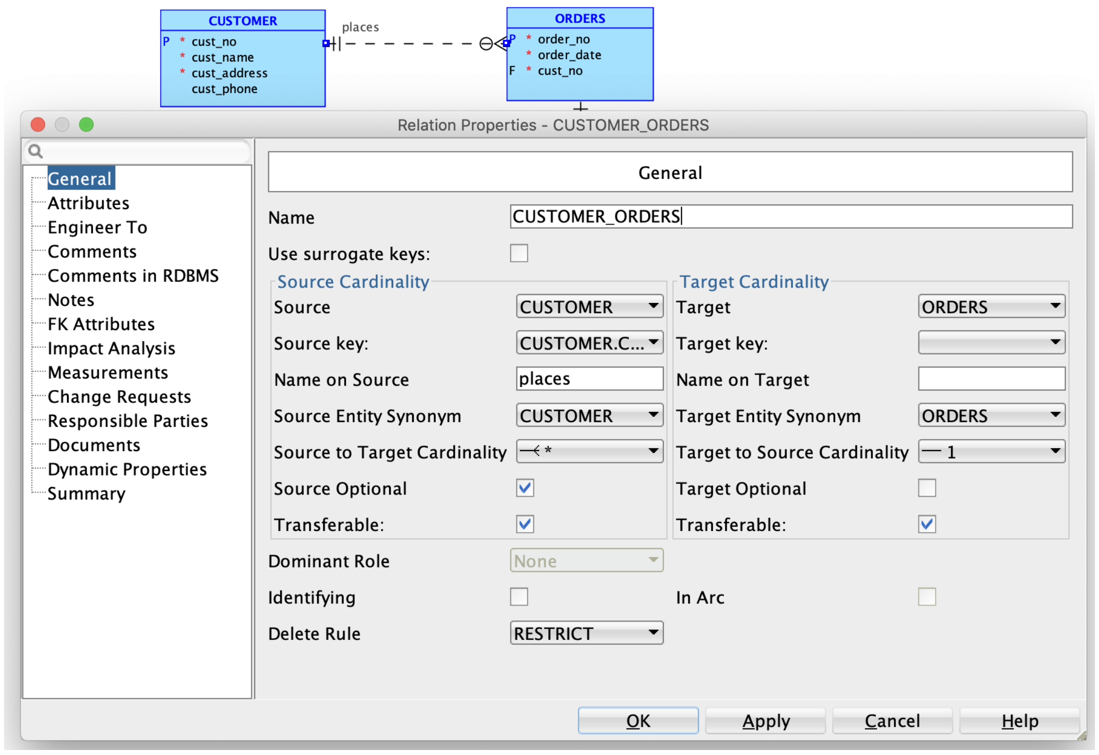


- View Details to "Attributes", and
- Show to "Labels" and "Legend"

When modelling you are **required to include the "Legend" on all models** (the panel may be moved to fit your model layout).

c. Adding Relationships

Now add a 1:N Relationship between CUSTOMER and ORDERS - click in CUSTOMER (the parent) and then in ORDERS. In the General Relations Properties name the relationship "CUSTOMER_ORDERS". For this unit, we name relationships in the one to many direction using the entities at each end - if the combined name is over 30 characters in length you should use an abbreviation of each entity name, such as CUST_ORD. Enter a name for the source (this will be our relationship label), enter a white space for the target, and set up the participation (Customer - optional, Order - mandatory i.e. not optional):



Relation Properties - CUSTOMER_ORDERS

General | Attributes | Engineer To | Comments | Comments in RDBMS | Notes | FK Attributes | Impact Analysis | Measurements | Change Requests | Responsible Parties | Documents | Dynamic Properties | Summary

General

Name: CUSTOMER_ORDERS
Use surrogate keys: ☐

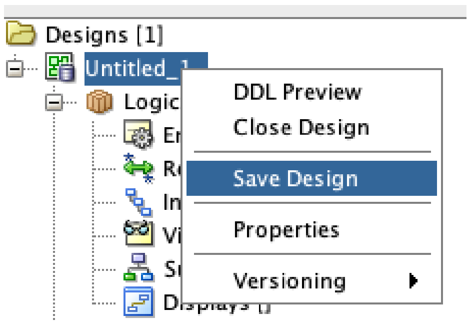
Source Cardinality		Target Cardinality	
Source	CUSTOMER	Target	ORDERS
Source key:	CUSTOMER.C...	Target key:	
Name on Source	places	Name on Target	
Source Entity Synonym	CUSTOMER	Target Entity Synonym	ORDERS
Source to Target Cardinality	—< *	Target to Source Cardinality	— 1
Source Optional	✓	Target Optional	☐
Transferable:	✓	Transferable:	✓

Dominant Role: None
Identifying: ☐ In Arc: ☐
Delete Rule: RESTRICT

OK | Apply | Cancel | Help

d. Saving Logical Model

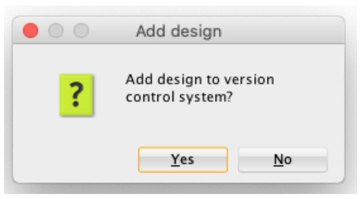
You must regularly save your model as you proceed through the development. To save your model, right-click the model in the browser panel and select "Save Design"



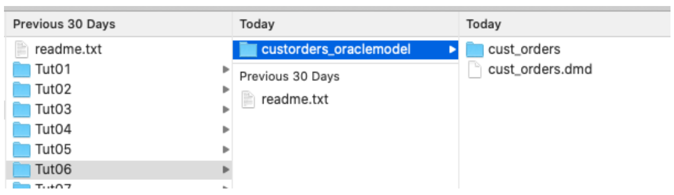
For your first save your design will be titled "Untitled" - before making this first save, create a folder (eg. custorders_oraclemodel) in the Tut 6 folder of your local repo to save your model in, then navigate and save into this folder. As you save give the model a name eg. cust_orders:

When you initially save a model **ensure** it is always saved into a folder specifically created for the model. This folder must be ***empty*** prior to you making your first save.

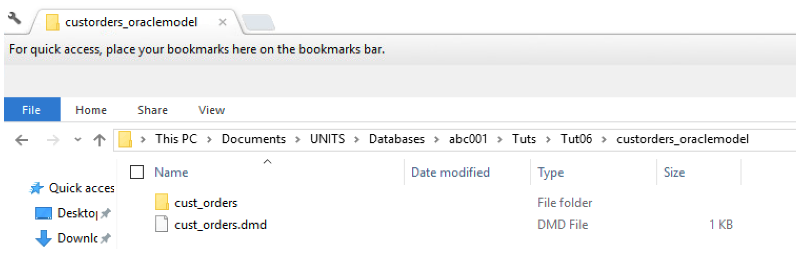
Click “Yes” on the version control system pop up window, add a proper commit comment then click OK



If an error message appears when you save a model, please refer to section 6.1.3 to fix the issue. When saved you will see that the folder contains both a file (cust_orders.dmd) and a folder (cust_orders) named after your model name:



or


custorders_oraclemodel
For quick access, place your bookmarks here on the bookmarks bar.
File
Home
Share
View
This PC › Documents › UNITS › Databases › abc001 › Tuts › Tut06 › custorders_oraclemodel
Quick access
Desktop
Downloads
Name
Date modified
Type
Size
cust_orders
File folder
cust_orders.dmd
DMD File
1 KB

BOTH of these files are required components of your model. Please note you **MUST NOT** manually edit/rename/move/delete etc **any** of the files in this folder. This folder must be managed entirely through the use of SQL Developer Data Modeler.

e. Commit/Push Model onto GitLab Server

The model must be **saved before you use GIT** to push your model to the FITGitLab server. It is a good idea to **regularly save and push your model** so that you have a clear development history. Here are the steps:

1. Save the model (Save Design)
2. Click Team → Git → Add All (it might output "There are no files to add.", you can ignore this message, simply click OK)
3. Click Team → Git → Commit, write a meaningful commit message and click ok
4. Click Team → Git → Push, key in your GitLab password, click Finish
5. Use the Web UI (login to the web interface of the server) to check that your files and folders are correctly being pushed

f. Drawing Rest of the Model

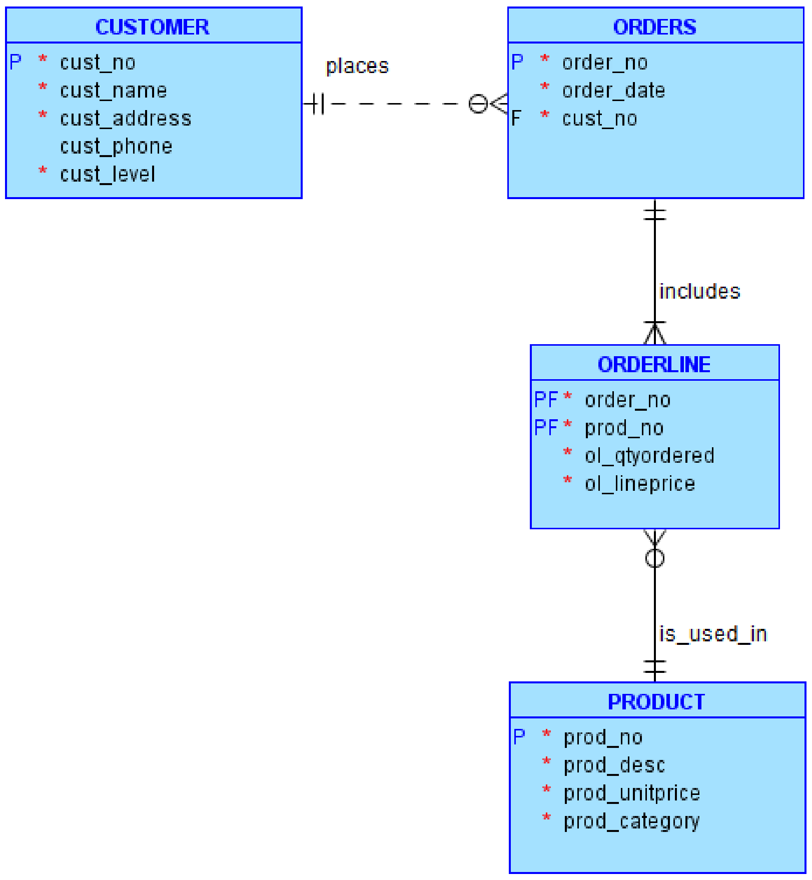
Proceed and add the PRODUCT entity. The PRODUCT attributes are (product number, product description and product unit price), <u>you should select a suitable data type and size for each of these attributes, remembering we try to use a numeric value for a PK</u>.

Data Modeler can draw M:N relationships on its logical models, however, you cannot add attributes to such relationships - ***do not use the M:N relationships in our models, they are NOT acceptable on a logical model***.

Add a new entity ORDERLINE and then connect this with ORDERS and PRODUCT via 1:N identifying relationships. Do not add any attributes until you have added the relationships. Now complete the ORDERLINE attributes (quantity ordered, total line price) - again, you should select a suitable data type and size for each of these attributes.

The conceptual model, which you have been mapping here, now needs to be re-examined in the light of further information that your client might provide. This can arise as feedback on your conceptual model (conceptual models are an excellent client-designer communication tool) and/or based on further details which you have collected. For our unit, we are enhancing the conceptual model (assignment 1A) with normalisation of a number of client sample forms and potentially some further requirements (assignment 1B).

Your full logical model will have the form:

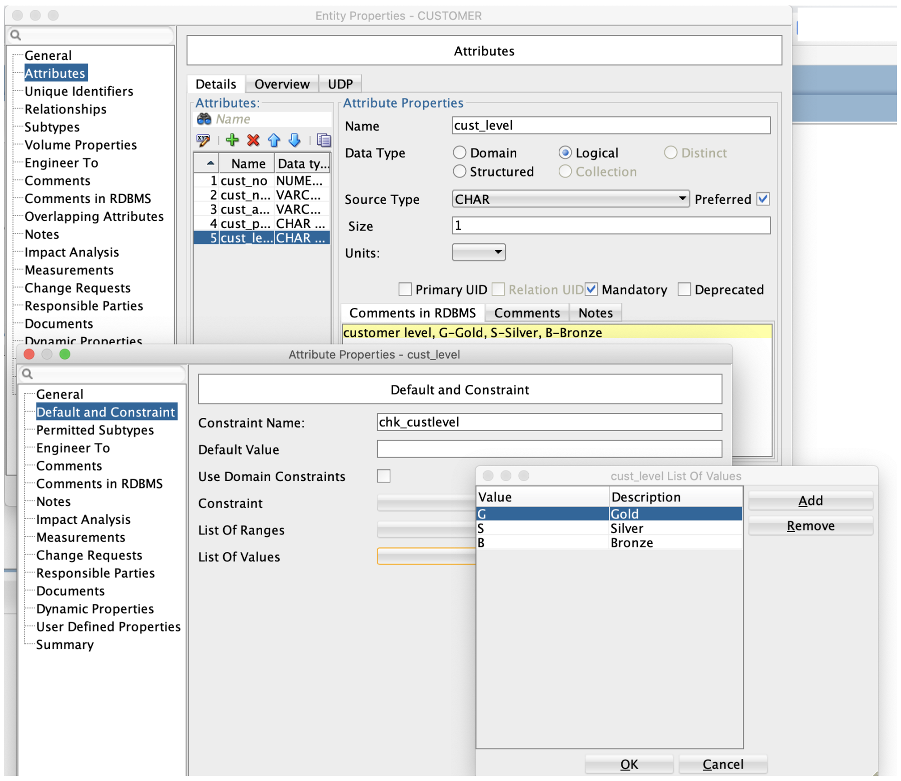


g. Adding Constraint

The client requirements indicate that the customer level must be recorded as Gold, Silver or Bronze. The level of the customer is determined by the "*total value of the orders they have placed in the previous financial year*" - this is not something that will be managed by your model, it will need some coding logic that is beyond the scope of your modelling - your model provides the data structure/source from which such a determination will be able to be made (here data from the orders and oderline relation will be used). However, within the model, we need to ensure that the level is constrained to the allowed values.

This should be enforced by adding a CHECK CONSTRAINT. In this case, the check constraint will ensure that the acceptable values for cust_level attribute are only Gold, Silver or Bronze. This constraint becomes part of the database structure, if the user wishes to add a new option, say 'Diamond' as the highest level then the database structure must be altered.

To add the check constraint, select the CUSTOMER entity, then select attributes and double click on cust_level. In the left-hand list of the pop-up window select "Default and Constraint". Give the constraint a name for example "chk_custlevel" (be careful to select an informative name eg. chk_columnname), click Constraint "List Of Values", add Value and Description for each level:



	Name	Data ty...
1	cust_no	NUME...
2	cust_n...	VARC...
3	cust_a...	VARC...
4	cust_p...	CHAR ...
5	cust_le...	CHAR ...

Name: cust_level
Data Type: Domain [✓ Logical] Distinct Structured Collection
Source Type: CHAR Preferred ✓
Size: 1
Units:
Primary UID Relation UID [✓ Mandatory] Deprecated
Comments in RDBMS: customer level, G-Gold, S-Silver, B-Bronze

Constraint Name: chk_custlevel
Default Value:
Use Domain Constraints
Constraint
List Of Ranges
List Of Values

cust_level List Of Values

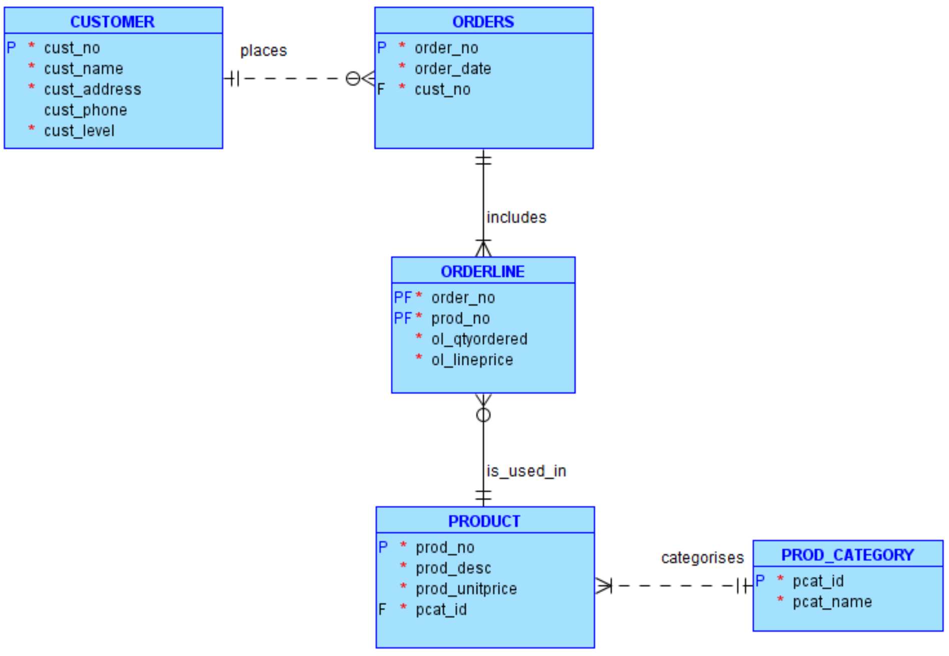
Value	Description
G	Gold
S	Silver
B	Bronze

Where a specific range is needed, select "List Of Ranges" and directly enter the required values into the dialogue boxes which appear.

h. Adding Lookup Table

The client also indicates that flexibility is required in the design to be able to add new and remove current, product categories as circumstances change. This should be handled by adding a LOOKUP table. Lookup tables are a good approach where there is likely to be a need to extend the possible values for an attribute.

To add a product category lookup table in our model, we create a new relation called PROD_CATEGORY and then add a relationship between PROD_CATEGORY and PRODUCT relation as shown below.

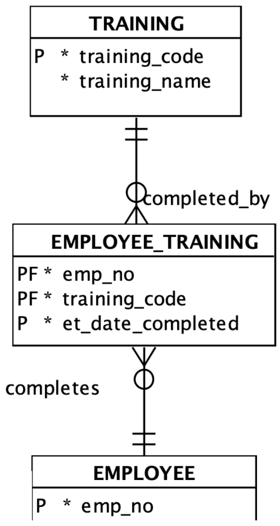


i. Adding Surrogate Key (where appropriate)

In developing a logical model, where *appropriate and documented*, you are free to introduce surrogate primary keys.

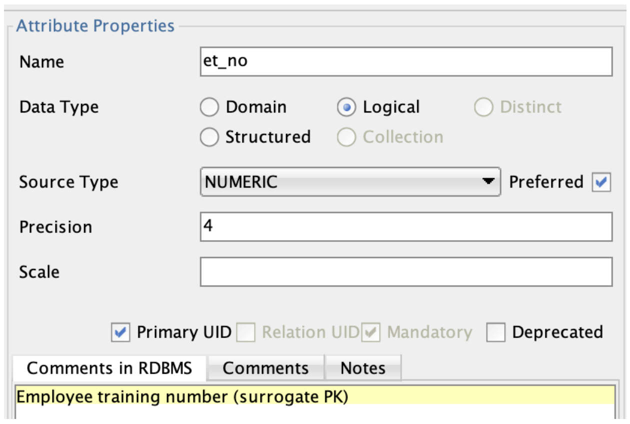
If a surrogate PK is introduced into your design you must ensure that the original key's uniqueness is still maintained by enforcing a unique identifier that includes the attributes which compose the original key.

As an example, we will look at the Monash Software case study. In the Monash Software model we have an EMPLOYEE_TRAINING relation (entity):



Here the PK of EMPLOYEE_TRAINING is composed of three attributes, all of different datatypes - in such a situation we need to add a surrogate PK.

The first step is to add a new attribute (here et_no) to be the new PK - **do not select/use the SQL Developer option "Create Surrogate Key"**. As designers, we need to be in control of surrogates added not leave this to the software.



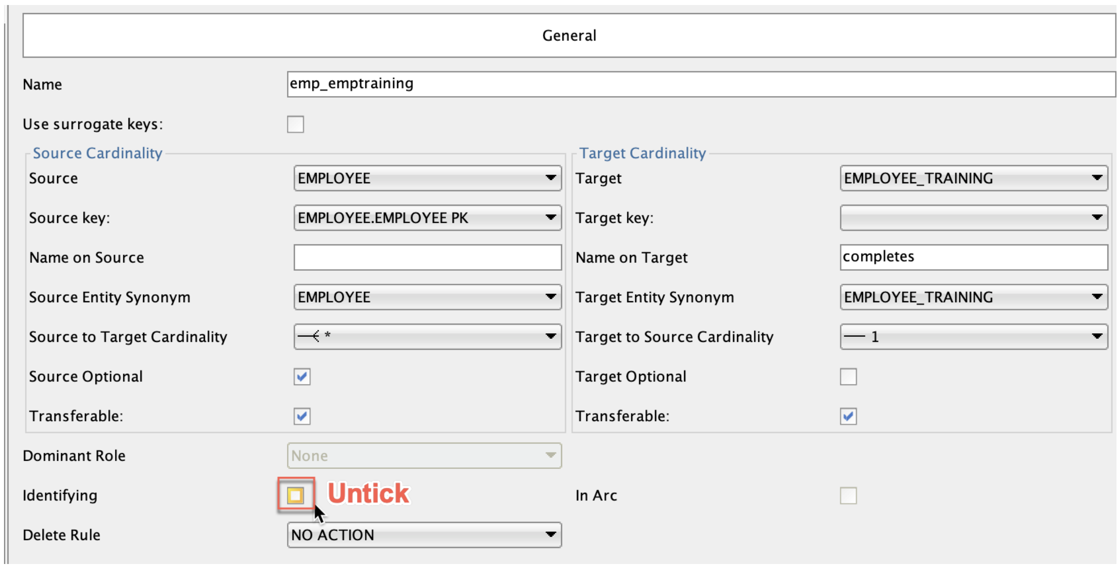
Name	et_no
Data Type	◯ Domain [✓ Logical] ◯ Distinct ◯ Structured ◯ Collection
Source Type	NUMERIC; Preferred ✓
Precision	4
Scale	

✓ Primary UID ☐ Relation UID ✓ Mandatory ☐ Deprecated

Employee training number (surrogate PK)

As part of this addition, we need to uncheck et_date_completed as "Primary" since it will not be part of the new primary key.

Next, we need to modify the incoming relationships - *completed_by* and *completes* into non-identifying relationships. For both, double click the relationship and untick the ‘identifying’ box in the relationships’ properties.

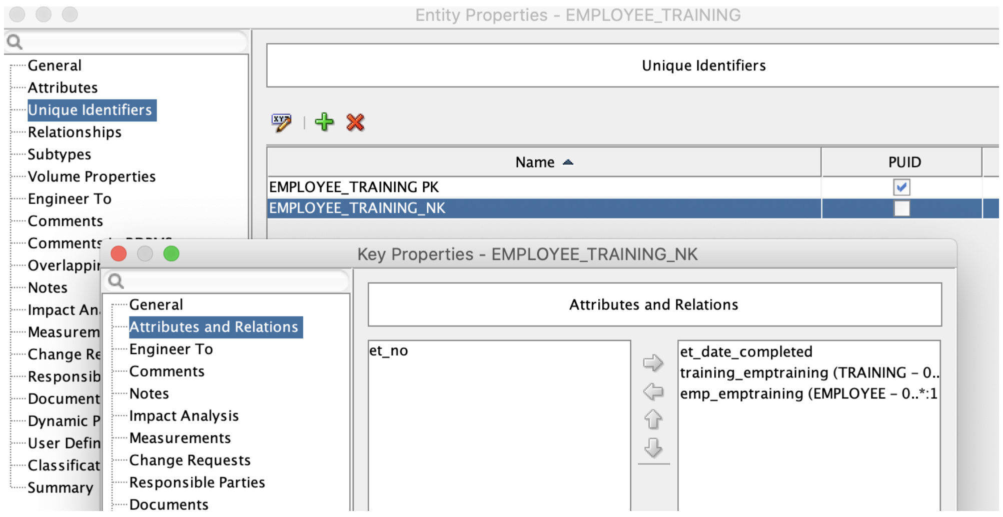


Name	emp_emptraining
Use surrogate keys:	☐

Source	EMPLOYEE	Target	EMPLOYEE_TRAINING
Source key:	EMPLOYEE.EMPLOYEE PK	Target key:	
Name on Source		Name on Target	completes
Source Entity Synonym	EMPLOYEE	Target Entity Synonym	EMPLOYEE_TRAINING
Source to Target Cardinality	—< *	Target to Source Cardinality	— 1
Source Optional	✓	Target Optional	☐
Transferable:	✓	Transferable:	✓
Dominant Role	None		
Identifying	☐	In Arc	☐
Delete Rule	NO ACTION		

We now need to protect our natural key (the previously declared key or natural key).

On the EMPLOYEE_TRAINING relation (entity), you need to select Unique Identifiers and add a new identifier (click the green add icon), this new unique identifier should be named appropriately. Double click the new unique identifier and add attributes in your previously identified Natural Key (here emp_no, training_code and et_date_completed) by bringing these attributes to the right box as shown in the picture below:



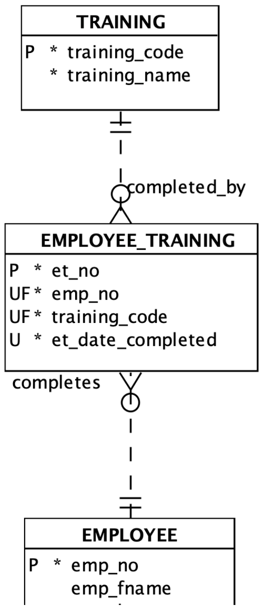
Unique Identifiers

Name ▲	PUID
EMPLOYEE_TRAINING PK	✓
EMPLOYEE_TRAINING_NK	

Attributes and Relations

et_no	et_date_completed
	training_emptraining (TRAINING – 0..
	emp_emptraining (EMPLOYEE – 0..*:1

Here is the EMPLOYEE_TRAINING relation with an added surrogate key.



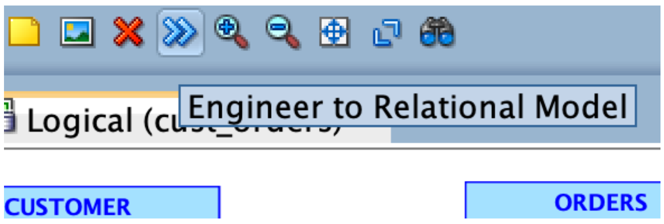
Note that the PK is shown as et_no, our introduced surrogate key, the natural key emp_no, training_code and et_date_completed have a U alongside the attributes to indicate these must be unique in this way protecting the natural key.

Important:

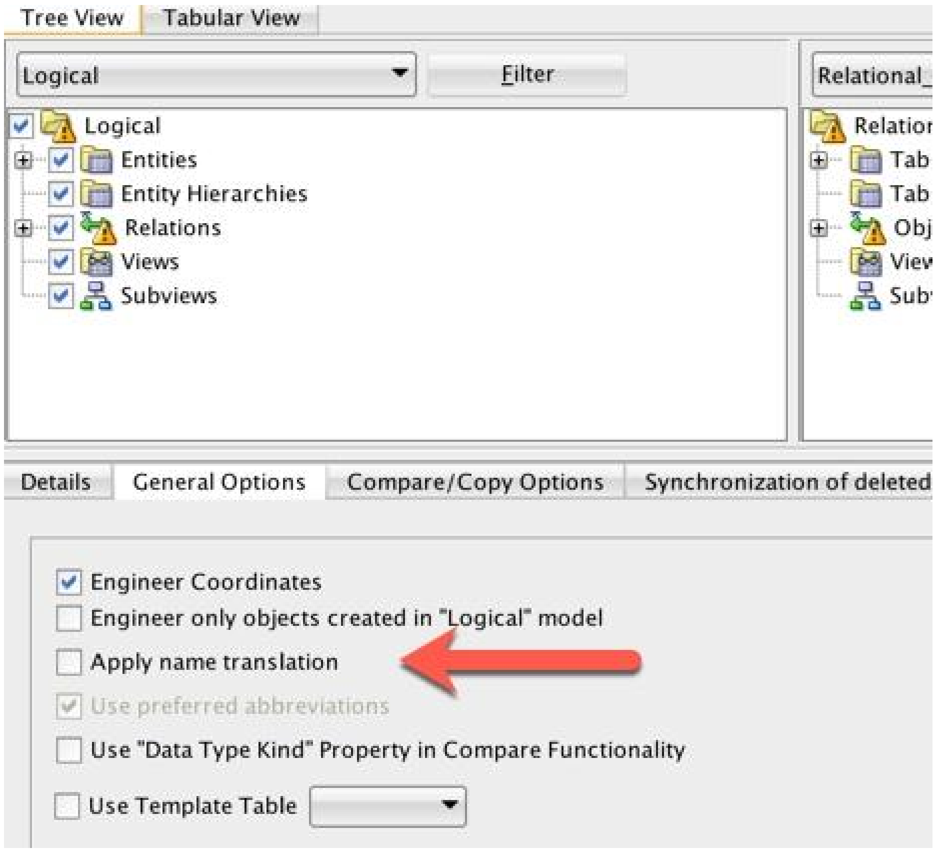
Before proceeding to develop a physical model, prior to implementation in our database of choice, we need to **validate *every relation shown on our final logical model.*** In this step we are ensuring the primary key selections are correct, the relations are in at least 3NF and that no insert/update/delete anomalies remain.

6.2.4 Develop a model - Stage 2 The Relational (Physical) Model

This completed logical model is now "Engineered" to a "Relational Model".



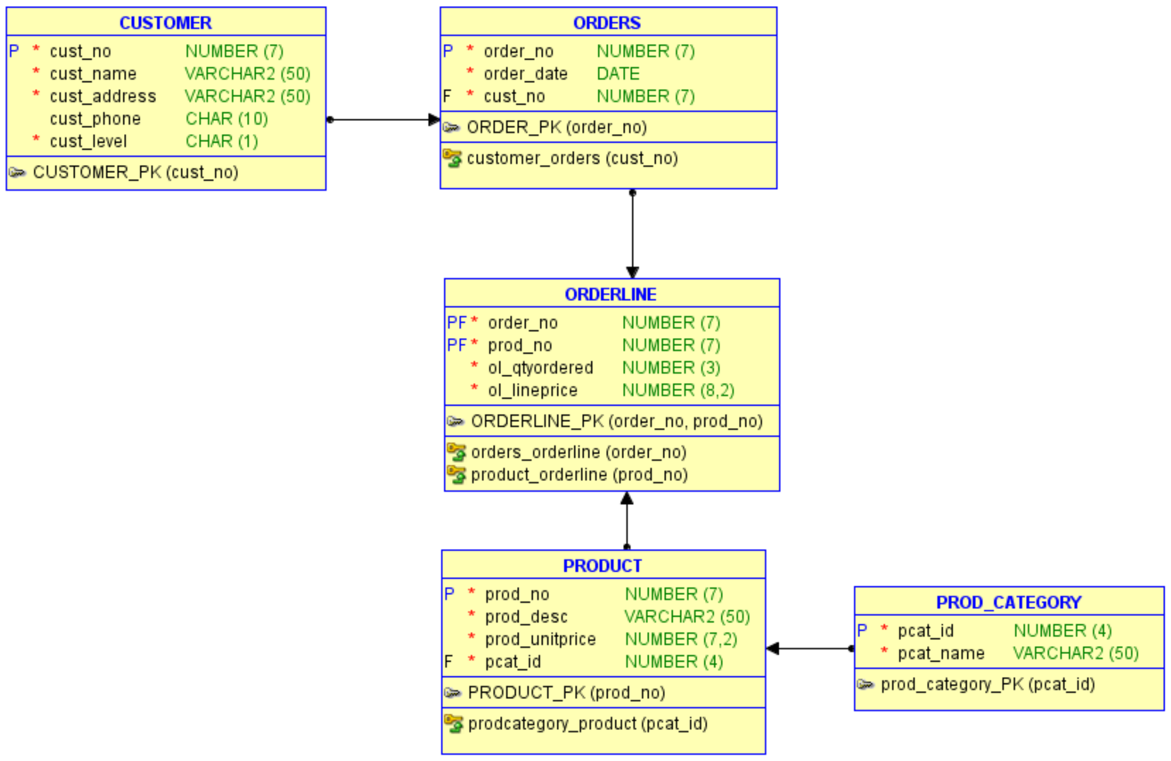
In the pop-up window which appears, on the "General Options" tab ensure "Apply name translation" is ***not*** checked:



☑ Engineer Coordinates
☐ Engineer only objects created in "Logical" model
☐ Apply name translation
☑ Use preferred abbreviations
☐ Use "Data Type Kind" Property in Compare Functionality
☐ Use Template Table

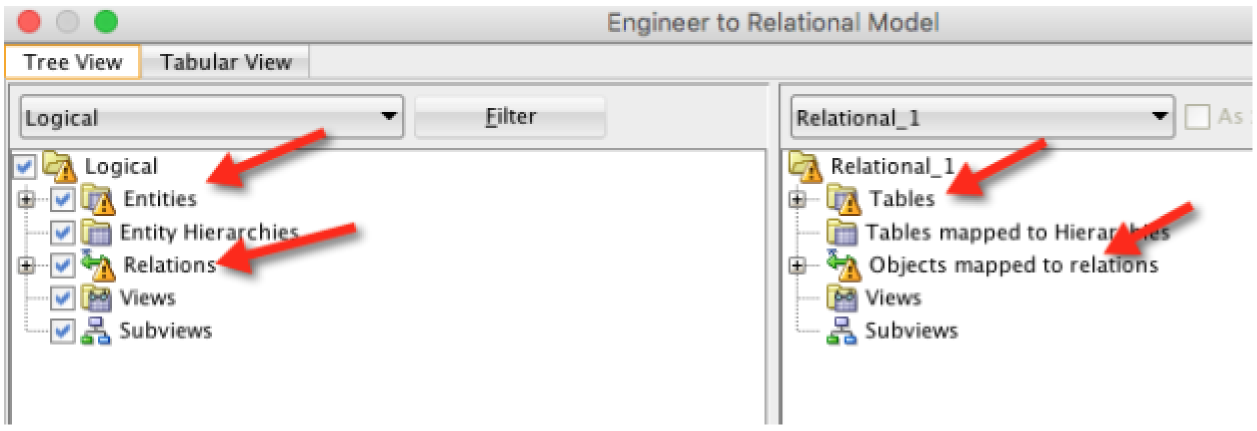
With this option not checked, Data Modeler will use exactly the same table and attribute names for both your logical and relational models (the names you selected), rather than apply its own set of rules and potentially change a name on you. If a naming error occurs you will need to correct it on your logical model and regenerate the relational model.

Select the bottom left button, "Engineer". The logical model will then be engineered and a Relational Model is opened:


CUSTOMER
P * cust_no NUMBER (7)
* cust_name VARCHAR2 (50)
* cust_address VARCHAR2 (50)
cust_phone CHAR (10)
* cust_level CHAR (1)
CUSTOMER_PK (cust_no)
ORDERS
P * order_no NUMBER (7)
* order_date DATE
F * cust_no NUMBER (7)
ORDER_PK (order_no)
customer_orders (cust_no)
ORDERLINE
PF * order_no NUMBER (7)
PF * prod_no NUMBER (7)
* ol_qtyordered NUMBER (3)
* ol_lineprice NUMBER (8,2)
ORDERLINE_PK (order_no, prod_no)
orders_orderline (order_no)
product_orderline (prod_no)
PRODUCT
P * prod_no NUMBER (7)
* prod_desc VARCHAR2 (50)
* prod_unitprice NUMBER (7,2)
F * pcat_id NUMBER (4)
PRODUCT_PK (prod_no)
prodcategory_product (pcat_id)
PROD_CATEGORY
P * pcat_id NUMBER (4)
* pcat_name VARCHAR2 (50)
prod_category_PK (pcat_id)

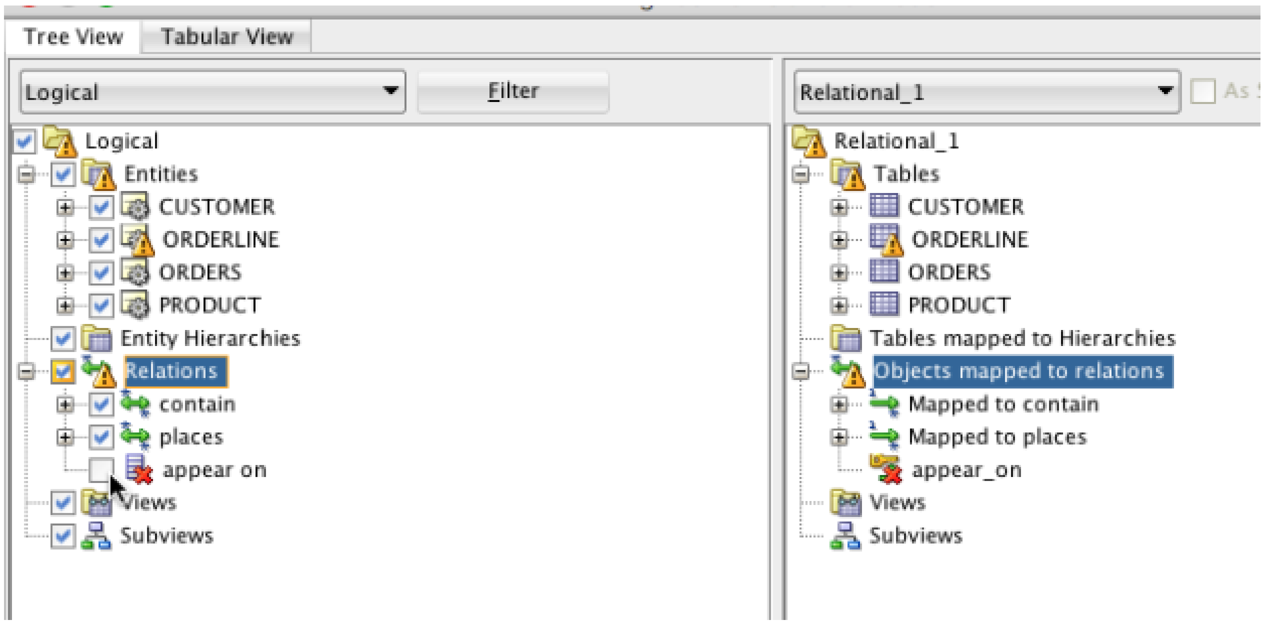
Please NOTE: the current version of Data Modeller has a bug that affects the generated relational model in some situations. Where you rename an attribute, such as from the result of a recursive relationship (for example you rename emp_no1 to mentor_no in the Monash Software model), the first time you generate the relational model Data Modeller will revert the attribute to the original name such as emp_no1. For this reason, after generation carefully check the generated model and if you note this has occurred, regenerate the relational model (without changing any settings) - after this second generation, the renamed attributes will be named as you selected.

If you are **re-engineering** a model (ie. trying to generate a previous relational model, after changes to your logical model) it is very important that you note that **SQL Developer does not automatically sync deletions from your logical model** - such changes must be individually selected to be synced to your relational model. When re-engineering a previous model carefully check the "Engineer to Relational Model" for any triangles with an exclamation mark symbol:



Such symbols represent issues you ***must address*** before generation.

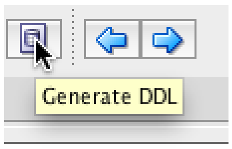
For example here under relations (where we are removing the "appear on" relation as a demonstration of what occurs):



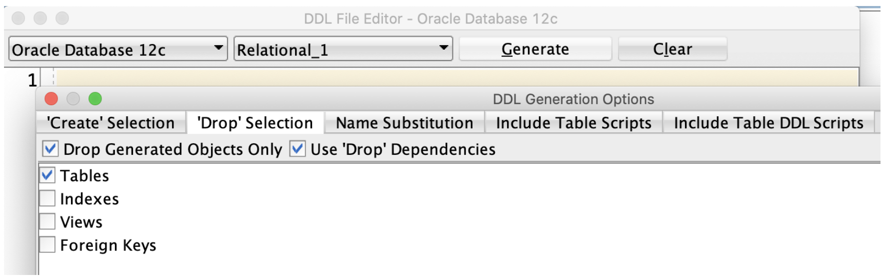
the removal of "appear on" has not been selected to be engineered to the relational model. You need to check the box if you wish it to be engineered (which we normally would).

If your relational model gets very confused you can select the relational model tab, do a ctrl+A or Apple+A and delete all the objects. The model can then be regenerated. Under ***no circumstances should you delete the relational model itself*** (in the left browser navigator), a bug in several versions of the software can result in such an action causing your relational model to "disappear".

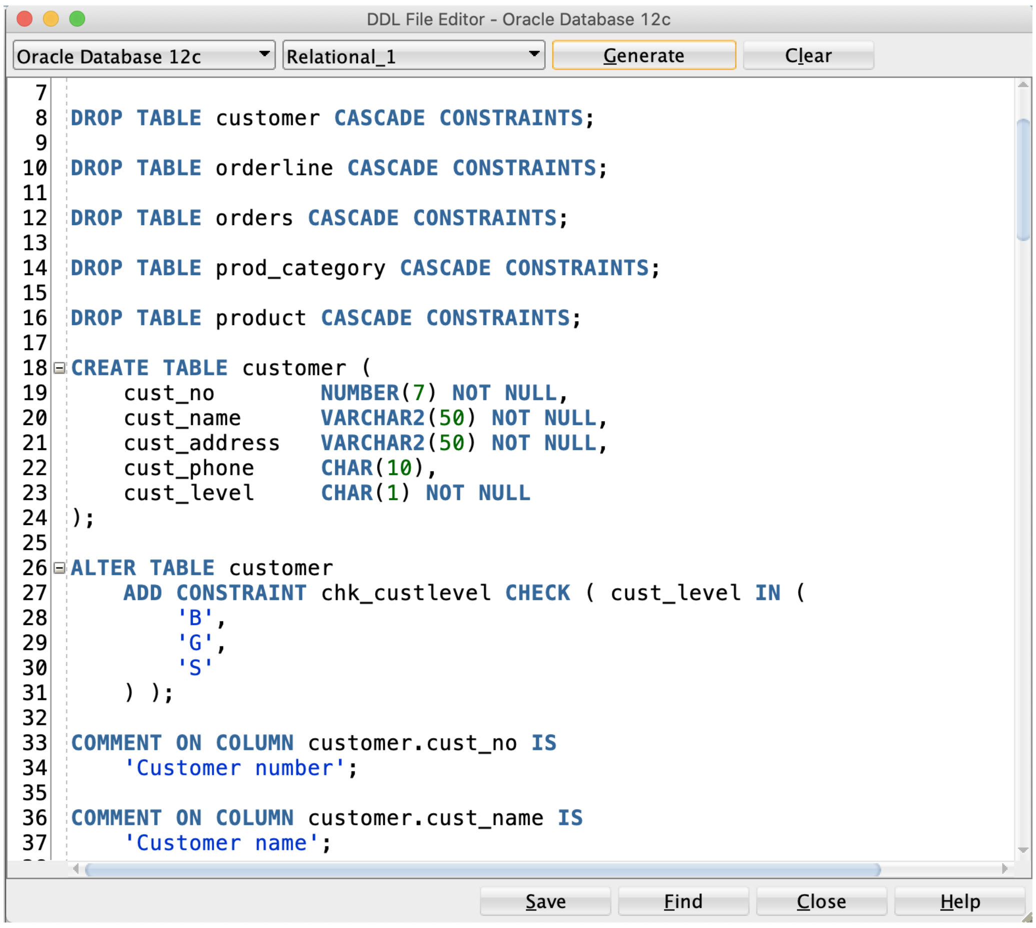
When you have configured the relational model as you wish, select Generate DDL from the top toolbar:



Select "Generate" in the pop-up window, specify the DDL Generation Options you wish (Drop tables should be included) and then select OK to generate the DDL.



☑ Drop Generated Objects Only ☑ Use 'Drop' Dependencies
☑ Tables
☐ Indexes
☐ Views
☐ Foreign Keys



```

DROP TABLE customer CASCADE CONSTRAINTS;

DROP TABLE orderline CASCADE CONSTRAINTS;

DROP TABLE orders CASCADE CONSTRAINTS;

DROP TABLE prod_category CASCADE CONSTRAINTS;

DROP TABLE product CASCADE CONSTRAINTS;

CREATE TABLE customer (
    cust_no         NUMBER(7) NOT NULL,
    cust_name       VARCHAR2(50) NOT NULL,
    cust_address    VARCHAR2(50) NOT NULL,
    cust_phone      CHAR(10),
    cust_level      CHAR(1) NOT NULL
);

ALTER TABLE customer
    ADD CONSTRAINT chk_custlevel CHECK ( cust_level IN (
        'B',
        'G',
        'S'
    ) );

COMMENT ON COLUMN customer.cust_no IS
    'Customer number';

COMMENT ON COLUMN customer.cust_name IS
    'Customer name';
```

The generated file can be Saved as an Oracle schema script by selecting "Save". You MUST use an extension of .sql, not the default .ddl - rename the file extension after saving if you use this approach (SQL Developers SQL client cannot load .ddl files). The simplest way to save this is to *not* use the "Save" button, use Ctrl+A to select all the script, swap to an SQL Window and use Ctrl+V, and then save the file from the SQL Window. Save the file as custorders_schema.sql.

You must add your student id and student name at the top of the generated SQL file. Additionally, to capture the full run of your schema file you must insert a SPOOL and ECHO on command at the top of your file, and a SPOOL OFF and ECHO off at the end of your script.

For example - added to the top of your script:

```
--student id: 12345678
--student name: Firstname Lastname

-- Capture run of script in file called custorders_schema_output.txt
set echo on
SPOOL custorders_schema_output.txt
```

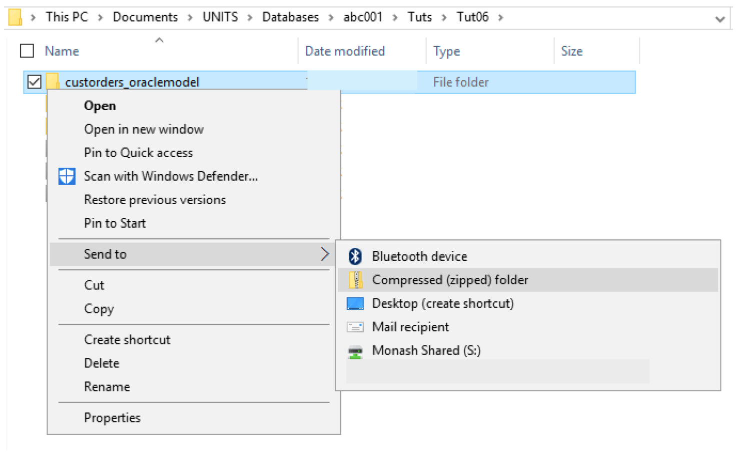
….. the SQL Developer Generated Script goes here ……

```
----------------------------------------------------
SPOOL off
set echo off
```

Test your generated file against Oracle and confirm it operates correctly. This script must be saved in an appropriate folder of your local repo and pushed to the FITGitLab server.

It is also possible to configure Data Modeler to directly synchronise the design into the Data Dictionary of a database connection (*we will not use this approach*).

Important Note:
To **submit your final assignment model to Moodle**, you must zip at the top folder lever (here the CustomerOrder folder) - right-click on the folder and use your zip tool. The zip archive must contain the top-level folder and **both** the **.dmd file** and the **folder matching the project name**



BEFORE submitting your assignment you MUST copy your zip archive to a new EMPTY folder, extract it and then ensure that you can navigate to this new folder and open your model successfully.

6.3 Property-Rental Model

6.3.1 Drawing Logical Model

Map the conceptual model (ERD) and normalisation result (3NF) of "Property Rental System" case study into a logical model in Oracle SQL Developer Data Modeler. In doing so make use of **at least one check clause** and **at least one surrogate key**.

Case Study (taken from week 3 tutorial):

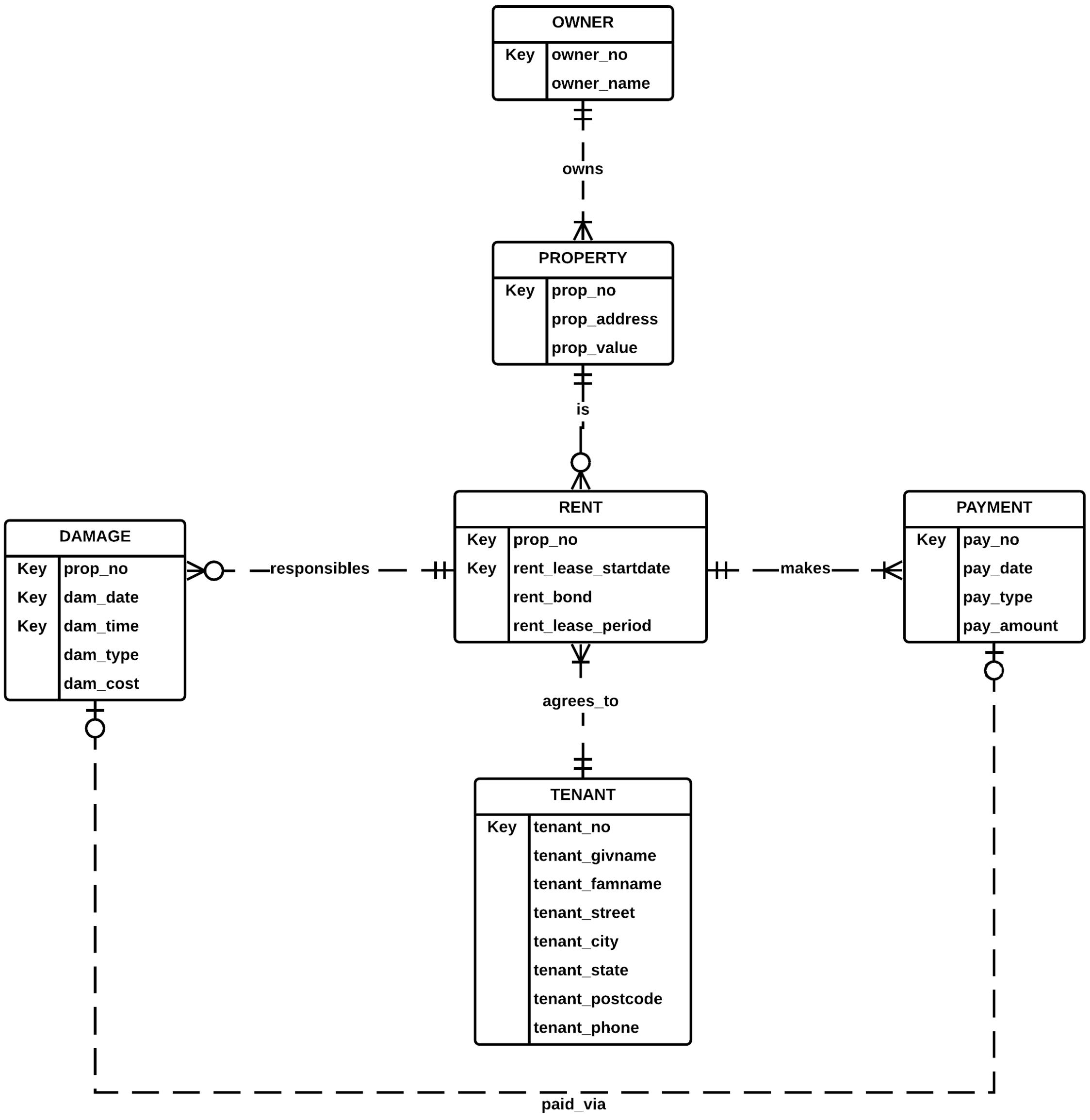
- Properties are rented by tenants. Each tenant is assigned a unique number by the Agency. Data held about tenants include family name, given name, property rented, contact address - street, city, state, postcode & telephone number. A tenant may rent more than one property and many tenants may rent parts of the same property (eg. a large shopping complex).
- Properties are owned by owners. Each property is assigned a unique property number. The agency only recognises a single owner for any of the properties it handles. The owner, address, and value are recorded for each property. Also, the lease period and bond are recorded for each property or sub-property rented. An owner may own several properties. For each owner an owner number is assigned, the owner name is also recorded.
- Properties are subject to damage and the agency records all instances of damage to its properties - property, date, type of damage and repair cost are recorded. Repair costs are charged directly to tenants
- Tenants pay accounts to the Agency - these consist of weekly rental payments, bond payments (for new properties) and damage bills. The date of payment, tenant, property, type of account (Rental, Bond, Damage) and amount are recorded. Each payment is assigned a payment number.

Extra Description:

The following requirements are provided by the agency as a follow up of the property-rental case study discussed in Week 3 Tutorial:

- The agency indicates that all addresses must be treated as simple attributes.
- The agency records normal property maintenance. Maintenance costs are charged to the property owner. A sample of the property maintenance report has been provided in the week 5 tutorial.
- The agency has indicated that it needs flexibility in the design to be able to add new, and remove current, paid by methods as circumstances change.

Conceptual Model (taken from tutorial 3):



Normalisation Result (taken from tutorial 5):

OWNER(<u>owner_no</u>, owner_givname, owner_famname, owner_address)
PROPERTY(<u>prop_no</u>, prop_address, owner_no)
MAINTENANCE(<u>prop_no</u>, <u>maint_datetime</u>, maint_desc, maint_cost)
TENANT(<u>tenant_no</u>, tenant_givname, tenant_famname)
PAYMENT(prop_no, rent_lease_startdate, <u>pay_no</u>, pay_date, pay_type, pay_amount, pay_paidby)
PROPERTY_TENANT(<u>prop_no</u>, <u>rent_lease_startdate</u>, rent_weekly_rate, rent_bond, tenant_no)
<u>Note that PROPERTY_TENANT is the same relation as RENT, thus you should use RENT as the relation name in the logical model</u>

6.3.2 Generate Relational (Physical) Model

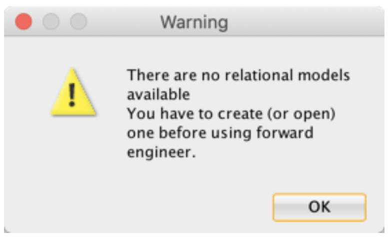
Engineer your Logical Model to a Relational Model, generate DDL and then create the tables etc in Oracle from the generated DDL.

6.4 SQL Developer Data Modeller Issues/Problems

This document lists some of the issues you may experience when using SQL Developer

6.4.1 Relational Model Disappears

You may see an error:



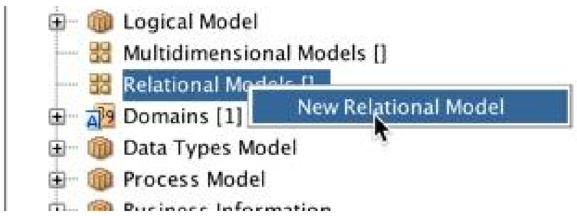
Warning

There are no relational models available
You have to create (or open) one before using forward engineer.

or observe that your relational model (if you had created one has disappeared)

This situation may arise under two circumstances:

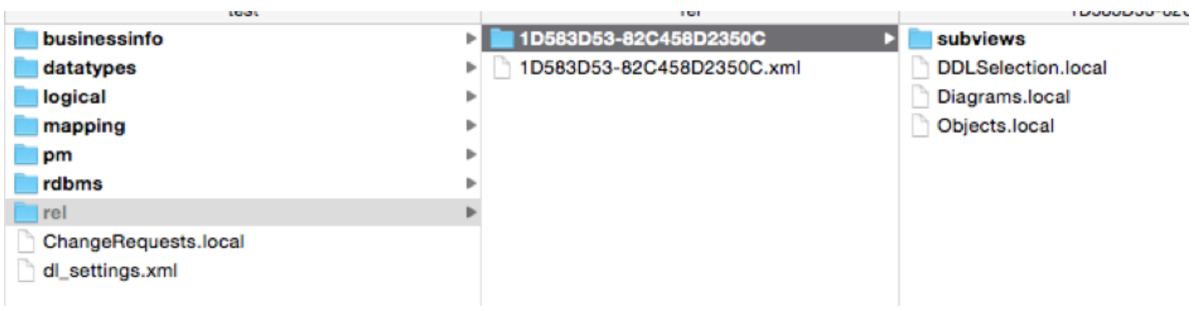
(a) where you have not engineered to a logical model as yet - ie. you are about to do it for the first time. In such a situation an error has occurred at some stage of the project creation/save and the relational model has been removed. To correct this, right-click - Relational model in the Data Modeller Browser and select "New Relational Model":



(b) where you have engineered a logical model but it has disappeared. When a model is saved SQL Developer Data Modeller sometimes fails to save your relational model fully. The Relational diagram is still present in your model but does not show within your project when you open it.

Before proceeding please ensure your model is closed and you have exited from SQL Developer.

To correct this problem, open your model *folder* and navigate to the rel folder (this is where your relational model is saved):



The model shown above is complete, it does not have the missing relational diagram problem. The rel folder should contain a folder and an xml document of the same name. If there is a folder inside rel (it will have a different name, they are all unique) but no xml file of the same name then this is a save error.

Your relational model has not gone, the problem is that the xml document was not saved correctly, follow the steps below to correct this problem:

Make sure your model is closed

1. Copy the supplied XML file in Moodle (week 6) into the rel folder
2. Rename the file from "CHANGE_TO_FOLDER_NAME.xml" to have the same name as the folder in your rel folder, with the .xml extension added eg. as above 1D583D53-B2C458D2350C.xml (yours will be different)
3. Open the file in a *text editor* and change the id="CHANGE_TO_FOLDER_NAME" on line 2 to be the same as your folder name

When you now open your model the relational model should be back.

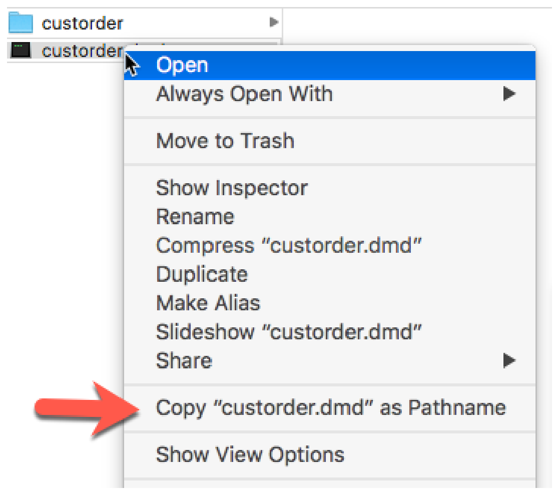
6.4.2 Having problems navigating into a folder

Sometimes the java-based file manager of SQL Developer Data Modeler has problems navigating through and selecting your folders.

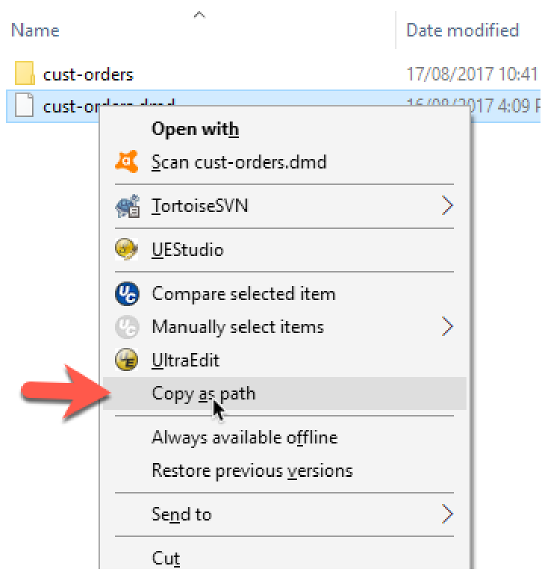
If this occurs, or as a standard alternative, you can type/obtain in the full pathname that you wish to open or navigate to. For example, if you have a Customer-Orders project you wish to open use the following steps:

In your file manager navigate to where your .dmd file for the project is and obtain its full pathname.

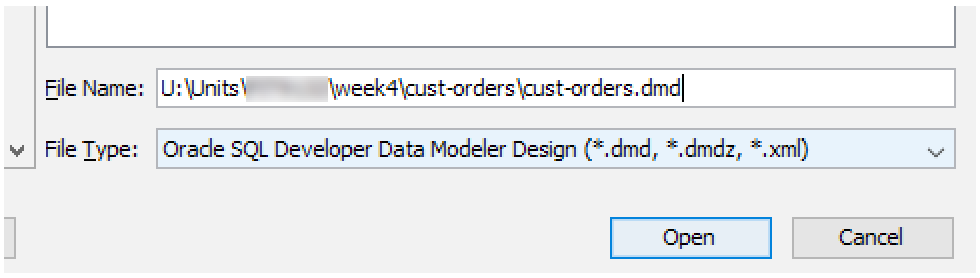
Under Mac OS, right-click the dmd file and right click and while holding this, the "option" key, this will result in:



under MS Windows, hold the shift and while holding it, right-click on the file or folder, "Copy as path" will appear:



This path can then be pasted directly into the SQL Developer dialogue box - to cause it to go to a particular folder or open a particular project. For example, in MS Windows:

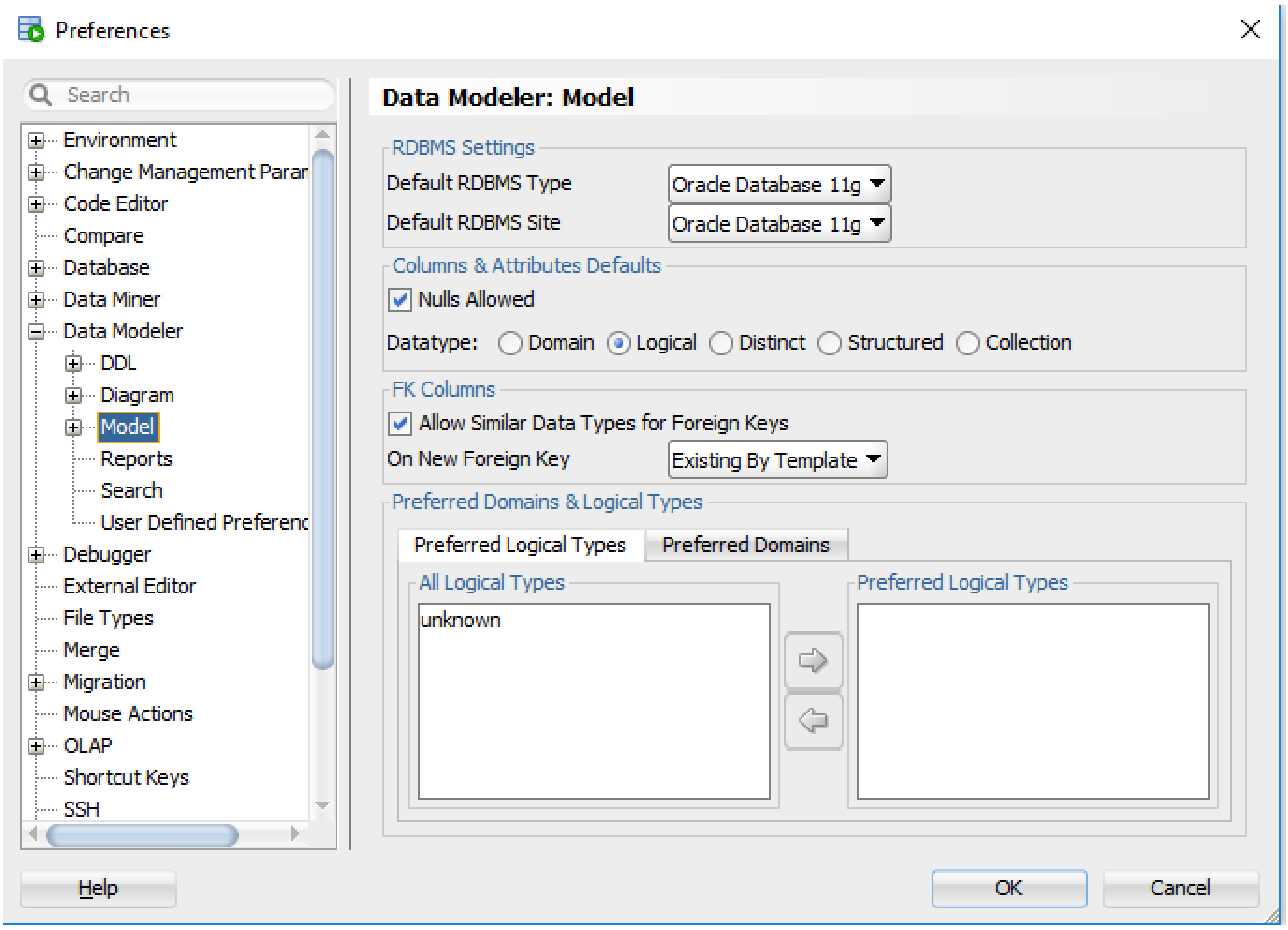

File Name: U:\Units\[redacted]\week4\cust-orders\cust-orders.dmd

File Type: Oracle SQL Developer Data Modeler Design (*.dmd, *.dmdz, *.xml)

Note that in MS Windows you *must* first remove the opening and closing quotes (") from the pathname.

6.4.3 Logical Data Types Disappear

"My data types have disappeared" - when you check in preferences you have no Logical Types:



Data Modeler: Model

RDBMS Settings
Default RDBMS Type: Oracle Database 11g
Default RDBMS Site: Oracle Database 11g

Columns & Attributes Defaults
✓ Nulls Allowed
Datatype: Domain [✓ Logical] Distinct Structured Collection

FK Columns
✓ Allow Similar Data Types for Foreign Keys
On New Foreign Key: Existing By Template

Preferred Domains & Logical Types
Preferred Logical Types | Preferred Domains
All Logical Types: unknown
Preferred Logical Types:

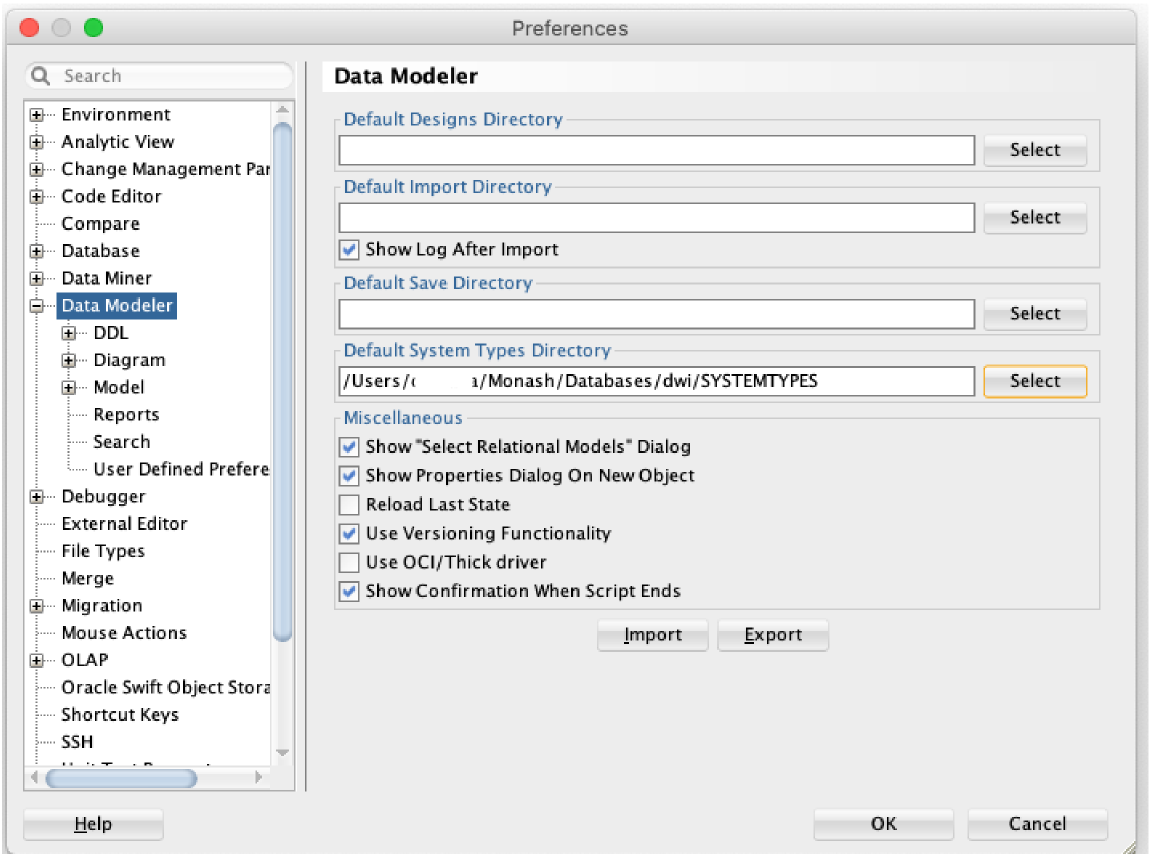
This has occurred because your default Systems Type Directory is now set incorrectly, or you have modified the folder in some manner.

This can occur if you have accidentally deleted the files in your Default System Types Directory if you have placed other files or subfolders in it, if you have a space in the pathname, or if SQL Developer has not correctly saved them at some stage.

To fix this:

- Go back to this setting (see 6.1.3 Configuring System Types Directory) and remove the current value in this entry, i.e. make this entry blank (have no path)
- Save your settings and exit SQL Developer
- Reopen SQL Developer and the types will be back.

You will need to reset up the Default System Types Directory as you previously did (ensure there is no space in the pathname).



Data Modeler

Default Designs Directory

Default Import Directory

[✓] Show Log After Import

Default Save Directory

Default System Types Directory: /Users/(ı/Monash/Databases/dwi/SYSTEMTYPES

Miscellaneous

[✓] Show "Select Relational Models" Dialog
[✓] Show Properties Dialog On New Object
[ ] Reload Last State
[✓] Use Versioning Functionality
[ ] Use OCI/Thick driver
[✓] Show Confirmation When Script Ends

Important

After you have completed your current lab activities, at the end of each session of work, remember to add files, commit and push any changes you have made to the FITGitLab server. ***We would expect a revision history as you build your model step by step. For Assignment 1 B this version history will be assessed.*** **Remember when working on models you MUST ENSURE the model is saved BEFORE attempting any GIT activities.**

You need to get into the habit of establishing this as a standard FIT2094-FIT3171 workflow - Pull at the start of your working session, work on the activities you wish to/are able to complete during this session, stage/commit changes and then Push the changes back to the FITGitLab server