FIT3171 2018 S2

UG Databases

FIT Database Teaching Team

Contents

1 Introduction to SQL Developer

The exercises in this first tutorial will introduce you to the SQL Developer software that you will use to connect to the Oracle database. The main aim of these exercises is to get you familiar with the SQL Developer interface/menu/options. At the end of this exercise, you should be able to:

1. make a connection and login to your account on the Monash Oracle database
2. change your Oracle password, and
3. disconnect from Monash Oracle database in SQL Developer.

There are a number of different ways to connect to an Oracle database. In this unit, you will use Oracle software called SQL Developer.

The SQL Developer software can be downloaded from your unit's Moodle site. Information on how to install and configure SQL Developer is available on the Moodle site under Software and Documentation section. Please also note that accessing the Oracle database from a machine located outside Monash University's network will **require you to connect to the Monash VPN (Virtual Private Network) service**. Information on how to install the Monash VPN client and then connect to the VPN service can be found at https://www.monash.edu/esolutions/network/vpn

Before looking at how we connect to the Oracle database and run SQL using SQL Developer, please take a few moments and configure SQL Developer so that the software will satisfy the needs of this unit. To do this follow the steps shown below.

Please note that Oracle sometimes relocates the preference items shown below as they update to a newer version. The images shown below are for SQL Developer version 17.3.1 (the currently installed version in the on-campus labs). If you are using a later version and are unable to find a particular setting, use the "Search" option in the top left of the preferences screen to find the desired item you wish to configure.

SQL Developer Preference Settings

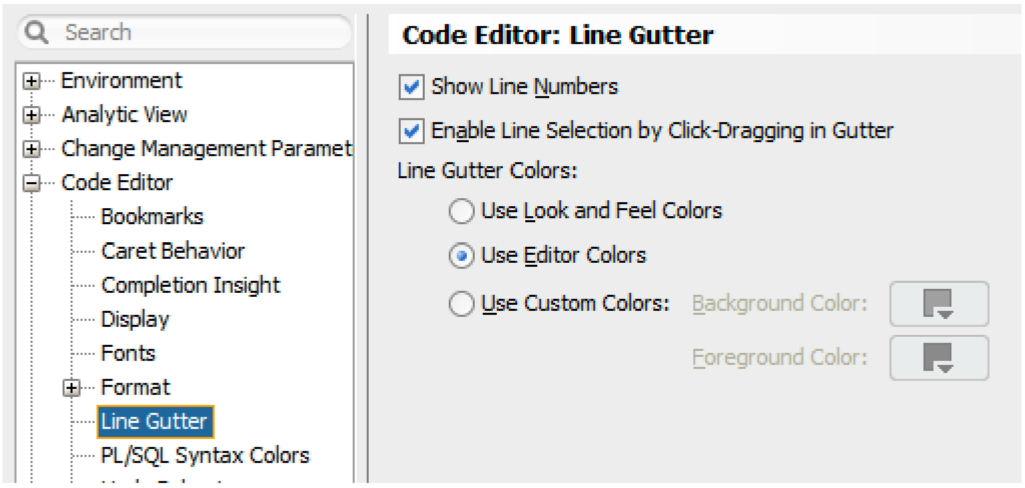
To help with your studies through the remainder of the semester you should configure SQL Developer to:

- display line numbers, and
- auto format SQL Code

This document also contains a number of other important configurations that you should implement before working extensively with SQL Developer. Under Windows, the SQL Developer preferences are accessed from the Tools menu, under MacOS from the Oracle SQL Developer menu on the left.

Display Line Numbers

To do this select: Preferences - Code Editor - Line Gutter - Show Line Numbers (Check Show Line Numbers):



Code Editor: Line Gutter

[✓] Show Line Numbers

[✓] Enable Line Selection by Click-Dragging in Gutter

Line Gutter Colors:

○ Use Look and Feel Colors

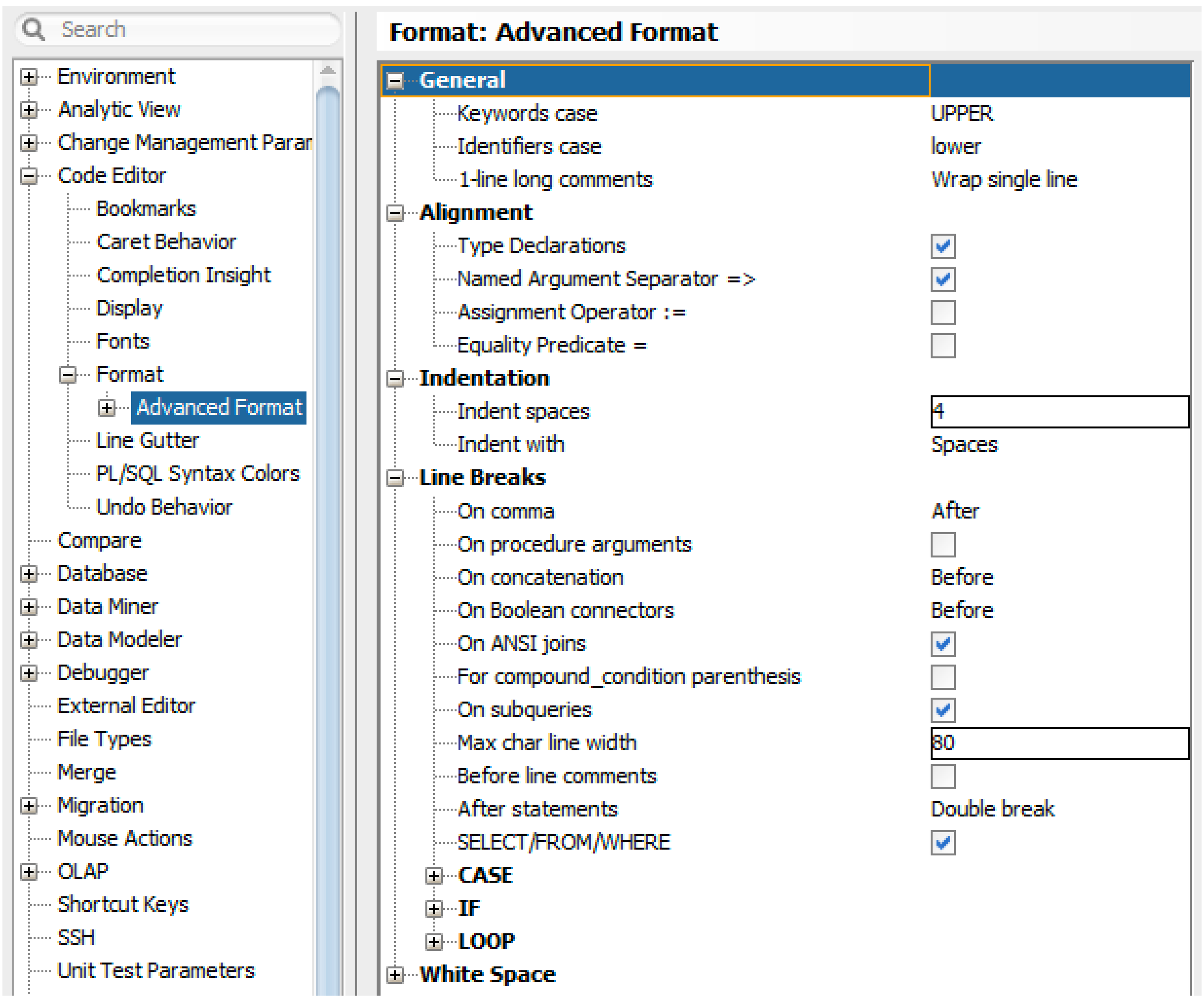
[● Use Editor Colors]

○ Use Custom Colors: Background Color: Foreground Color:

SQL Code AutoFormat

We now wish to configure SQL Developer so that it will reformat any SQL we enter into a well set out "pretty" format. The aim will be to simply type SQL commands without worrying about layout and then use the SQL Developer assigned format key (see below) to automatically format the code.

Preferences - Code Editor - Format - Advanced Format. As a starting point, we recommend the settings as shown below (Select OK when completed):



Search

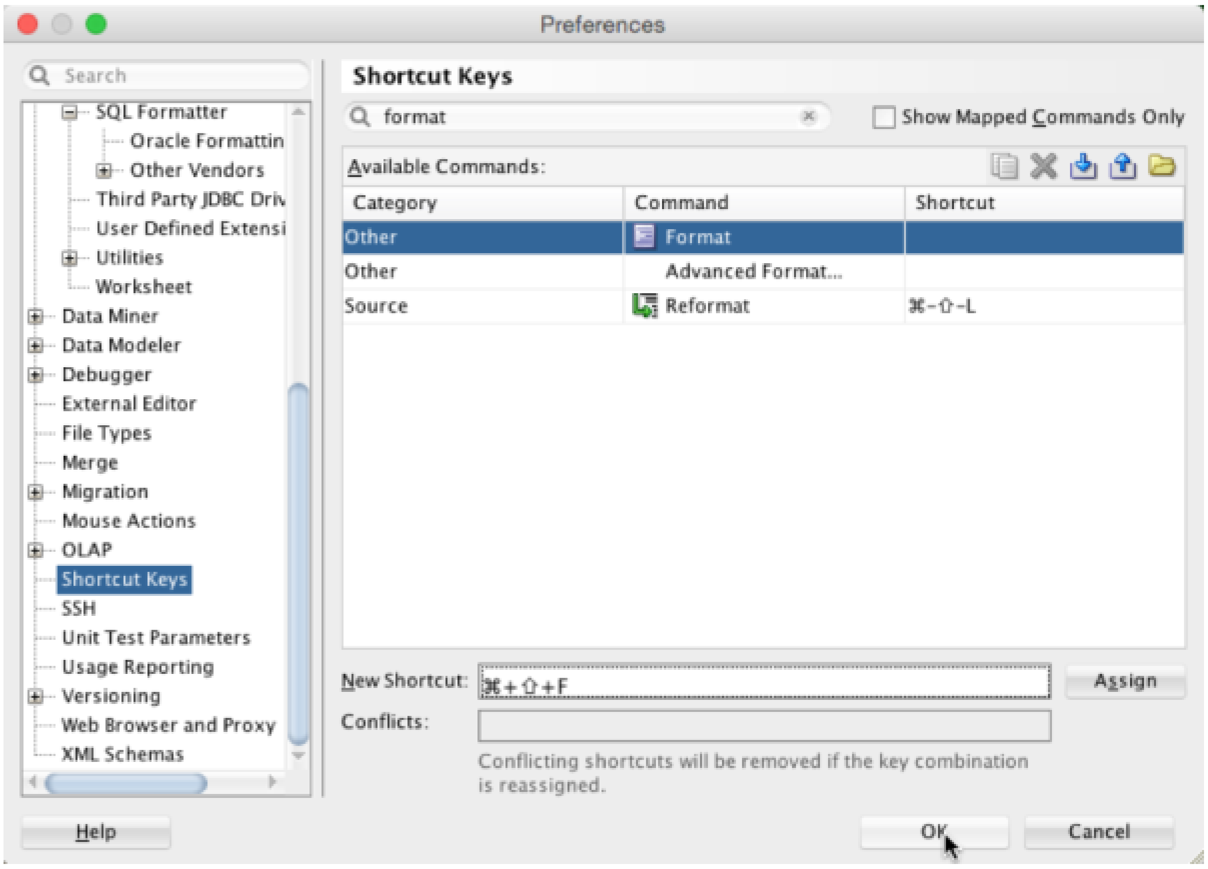
- Environment
- Analytic View
- Change Management Parar
- Code Editor
  - Bookmarks
  - Caret Behavior
  - Completion Insight
  - Display
  - Fonts
  - Format
    - Advanced Format
  - Line Gutter
  - PL/SQL Syntax Colors
  - Undo Behavior
- Compare
- Database
- Data Miner
- Data Modeler
- Debugger
- External Editor
- File Types
- Merge
- Migration
- Mouse Actions
- OLAP
- Shortcut Keys
- SSH
- Unit Test Parameters

Format: Advanced Format

General	
Keywords case	UPPER
Identifiers case	lower
1-line long comments	Wrap single line
Alignment	
Type Declarations	✓
Named Argument Separator =>	✓
Assignment Operator :=	☐
Equality Predicate =	☐
Indentation	
Indent spaces	4
Indent with	Spaces
Line Breaks	
On comma	After
On procedure arguments	☐
On concatenation	Before
On Boolean connectors	Before
On ANSI joins	✓
For compound_condition parenthesis	☐
On subqueries	✓
Max char line width	80
Before line comments	☐
After statements	Double break
SELECT/FROM/WHERE	✓
CASE	
IF	
LOOP	
White Space	

Depending on your personal preferences you might like to return to these settings later on in the unit and reconfigure them to your wishes. You are free to use any settings which you are happy with, the settings shown are a starting point for your study.

To automate formatting we now need to assign a format key combination - select Preferences - Short Cut Keys and set a shortcut key for "Format". Your installation may already have a key set for format (eg. on Windows it is Ctrl+F7) - if you are happy with that key you do not need to change anything. Below shows setting a shortcut key (command+shift+F) for a mac:



Shortcut Keys

format

Show Mapped Commands Only

Available Commands:

Category	Command	Shortcut
Other	Format	
Other	Advanced Format...	
Source	Reformat	⌘-⇧-L

New Shortcut: ⌘+⇧+F

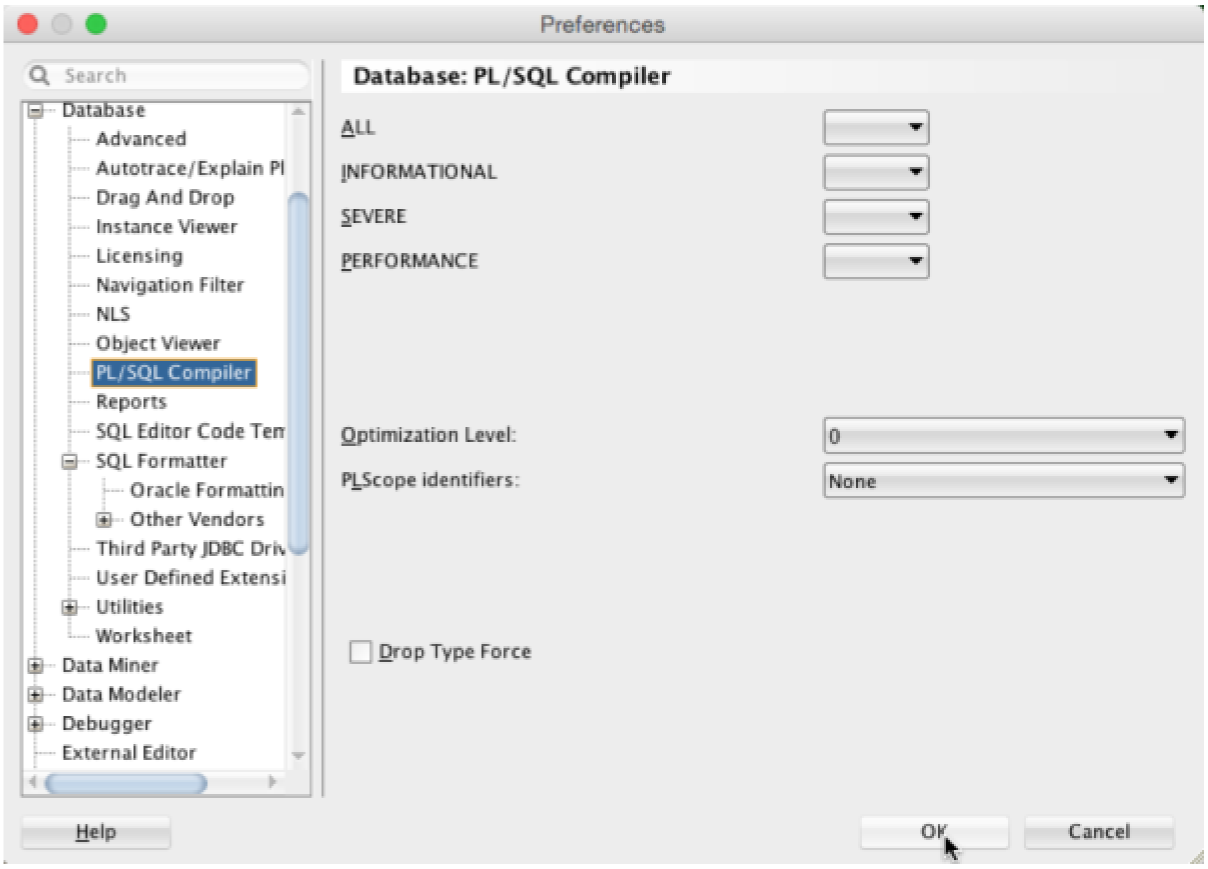
Conflicts:

Conflicting shortcuts will be removed if the key combination is reassigned.

Then you can enter SQL on a single line, or multiple lines, and select your "Format" shortcut key to "pretty" format the query.

PL/SQL Scope Identifiers

Please ensure you set the PL/Scope identifiers, under Database, to "None":



Database: PL/SQL Compiler

ALL
INFORMATIONAL
SEVERE
PERFORMANCE

Optimization Level: 0
PLScope identifiers: None

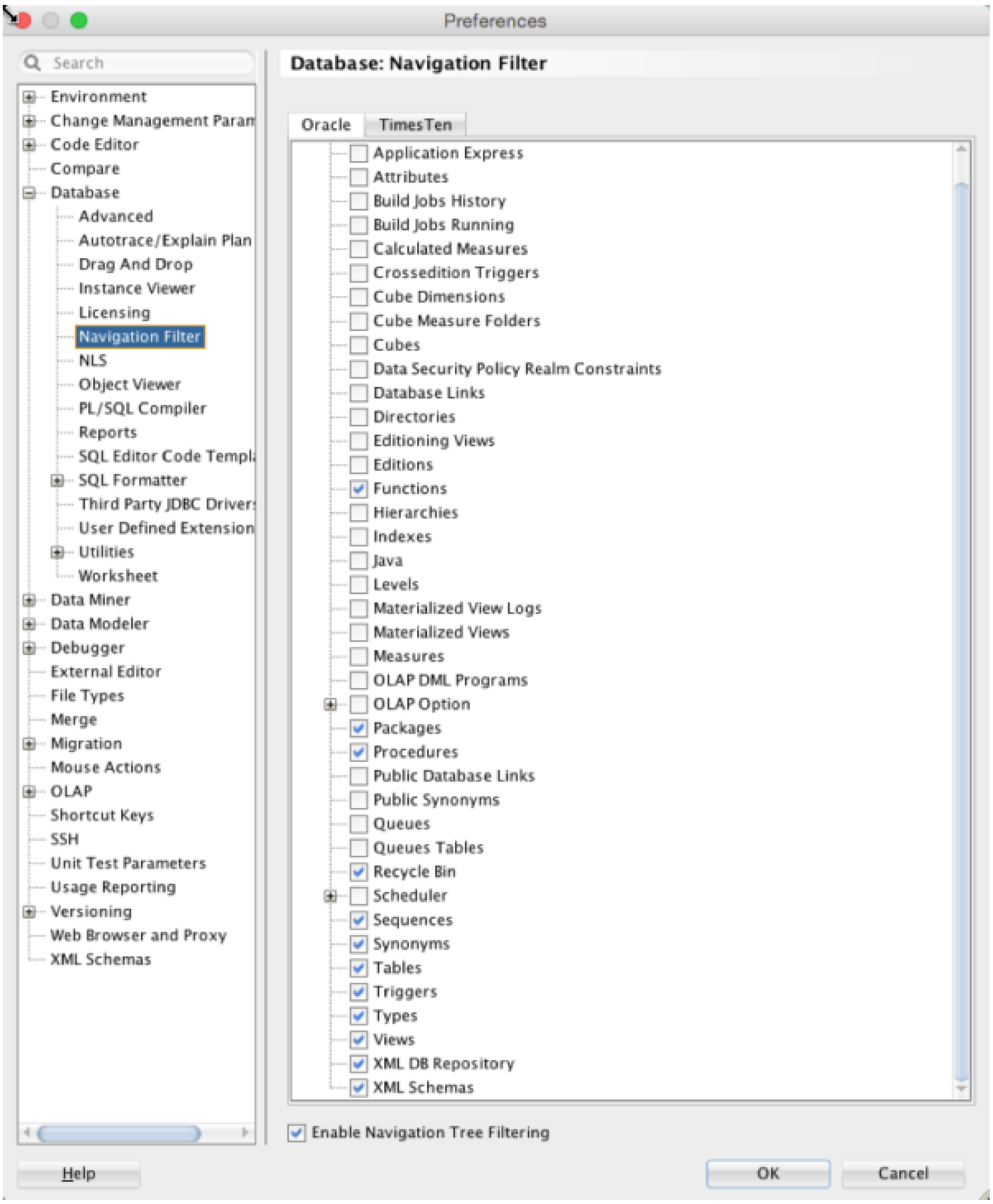
Drop Type Force

Navigation Filter (Optional)

SQL Developer displays a large, and often confusing array of items under its connection tree, these can be limited by applying a navigation filter (Database - Navigation Filter).

Select "Enable Navigation Tree Filtering"

Select to show nothing on the "Times Ten" tab. The suggested items for the Oracle tab are:



Database: Navigation Filter

- [ ] Application Express
- [ ] Attributes
- [ ] Build Jobs History
- [ ] Build Jobs Running
- [ ] Calculated Measures
- [ ] Crossedition Triggers
- [ ] Cube Dimensions
- [ ] Cube Measure Folders
- [ ] Cubes
- [ ] Data Security Policy Realm Constraints
- [ ] Database Links
- [ ] Directories
- [ ] Editioning Views
- [ ] Editions
- [✓] Functions
- [ ] Hierarchies
- [ ] Indexes
- [ ] Java
- [ ] Levels
- [ ] Materialized View Logs
- [ ] Materialized Views
- [ ] Measures
- [ ] OLAP DML Programs
- [ ] OLAP Option
- [✓] Packages
- [✓] Procedures
- [ ] Public Database Links
- [ ] Public Synonyms
- [ ] Queues
- [ ] Queues Tables
- [✓] Recycle Bin
- [ ] Scheduler
- [✓] Sequences
- [✓] Synonyms
- [✓] Tables
- [✓] Triggers
- [✓] Types
- [✓] Views
- [✓] XML DB Repository
- [✓] XML Schemas

[✓] Enable Navigation Tree Filtering

Copying your configuration to a new PC/Laptop

The configurations you have completed can be transferred to a different computer by manually copying the file product-preferences.xml. This process is also useful in the Monash on-campus labs if your PC has been configured to reset the configuration with each login - simply save the preferences file to your mapped desktop folder (see the section below 'Working in the On Campus Labs" on how to create this mapping).

Simply copy the file product-preferences.xml from the configured version to the new version where you want the same settings:

On windows the files is located in the folder:

C:\Users\yourusername\AppData\Roaming\SQL Developer\systemx.x.x.x.x\o.sqldeveloper\product-preferences.xml

for the current SQL Developer version and a user lsmi1 this would be:

C:\Users\lsmi1\AppData\Roaming\SQL Developer\system17.3.1.279.0537\o.sqldeveloper\product-preferences.xml

If the AppData folder does not appear under C:\Users\yourusername you will need to [turn show hidden files and folders on](https://support.microsoft.com/en-au/help/14201/windows-show-hidden-files) (https://support.microsoft.com/en-au/help/14201/windows-show-hidden-files).

On the mac the files is located in the folder:

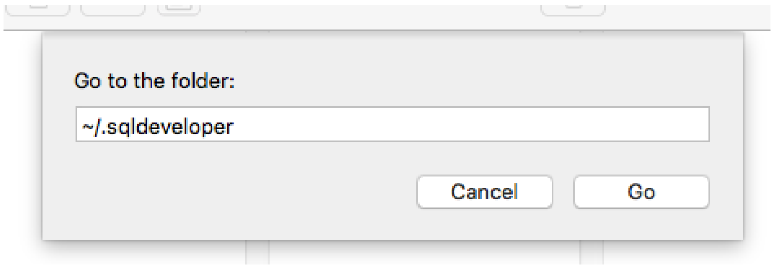
/Users/yourusername/.sqldeveloper/systemx.x.x.x.x/o.sqldeveloper.x.x.x.x.x/product-preferences.xml

for the current SQL Developer version and a user lsmi1 this would be:

/Users/lsmi1/.sqldeveloper/system17.3.1.279.0537/o.sqldeveloper/product-preferences.xml

You are strongly recommended to make a copy of this file, once you are happy with your settings, and keep it in a safe place.

To open the /Users/yourusername/.sqldeveloper folder on your mac, in Finder select from the Finder menu Go - Go to Folder and enter the path ~/.sqldeveloper :



SQL Developer Language

If your laptop or computer is setup to use a non-English language (eg. Chinese), when SQL Developer installs it will make use of this system language as its default language.

For your Monash study, you ***must* modify** this so that the language being used (and displayed) by SQL Developer is English. To achieve this, edit the sqldeveloper.conf file (ensure SQL Developer is not running when editing this file) - it's normal location is:

MS Windows:

C:\Program Files\sqldeveloper\sqldeveloper\bin\sqldeveloper.conf

OSX:

/Applications/SQLDeveloper.app/Contents/Resources/sqldeveloper/sqldeveloper/bin/sqldeveloper.conf

and add the line:

AddVMOption -Duser.language=en

Note that you ***must carefully type (case is important) this line into the sqldeveloper.conf file yourself***, do not copy and paste it from this document. This entry must be on a line by itself and left aligned (add a new line at the end).

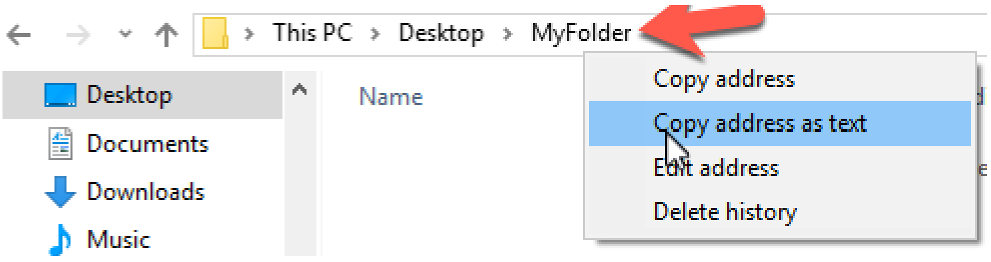
Also, note this must be done with a text editor. For MS Windows please do not use Windows notepad instead use something like Notepad++ (http://notepad-plus-plus.org/)

Working in the On-Campus Labs

The technique which the University uses in the on-campus Windows 10 labs to allow your files and desktop to be portable between different machines causes SQL Developer considerable problems in saving files.

To save files (.sql, .dmd, etc) from SQL Developer on the Monash Windows 10 on-campus labs we must use a mapped drive. If you have not as yet created such a drive, please use the following procedure.

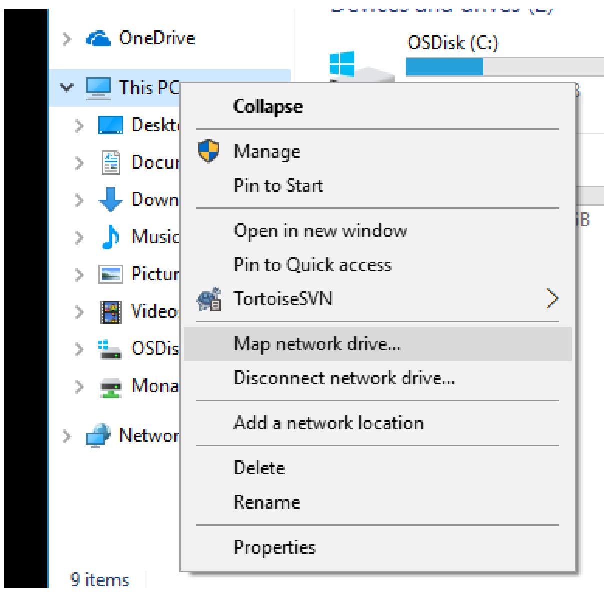
First, create a folder on your Desktop (say MyFolder, or you can use any folder you have previously created), navigate to this folder and then right click in the path bar of the Windows file manager on the folder, and select "Copy address as text":



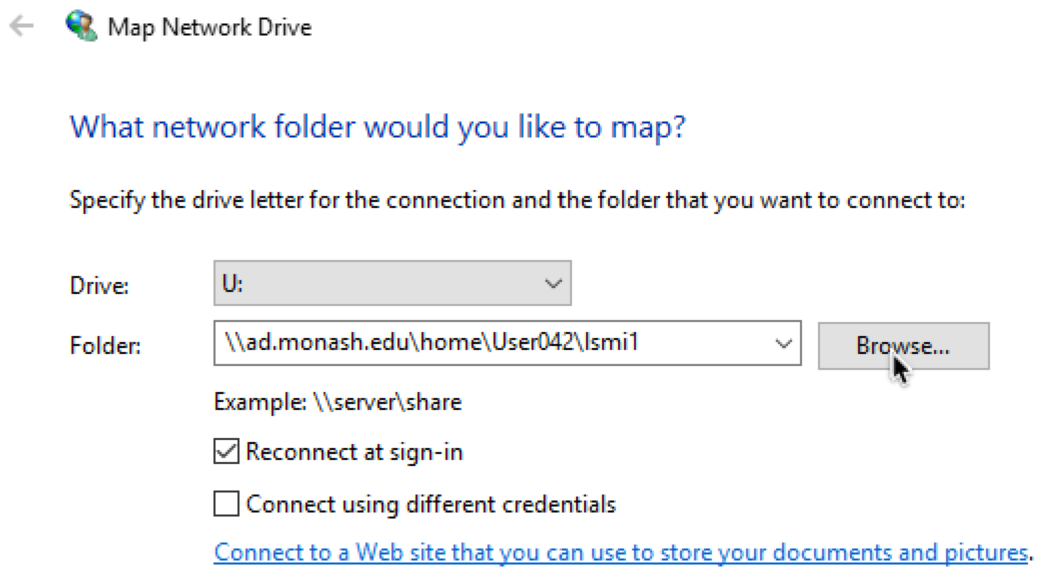
This will result in an address of the form:

```
\\ad.monash.edu\home\User042\lsmi1\Desktop\MyFolder
```

You should then use the first part of this path (exclude the ending \Desktop\MyFolder) to map a drive in Windows (use any drive letter you wish) however, we suggest U. First right click This PC and select "Map Network Drive":



Then enter the path that you copied above (without the Desktop eg. \\ad.monash.edu\home\User042\lsmi1) into the Folder entry:


Map Network Drive

What network folder would you like to map?

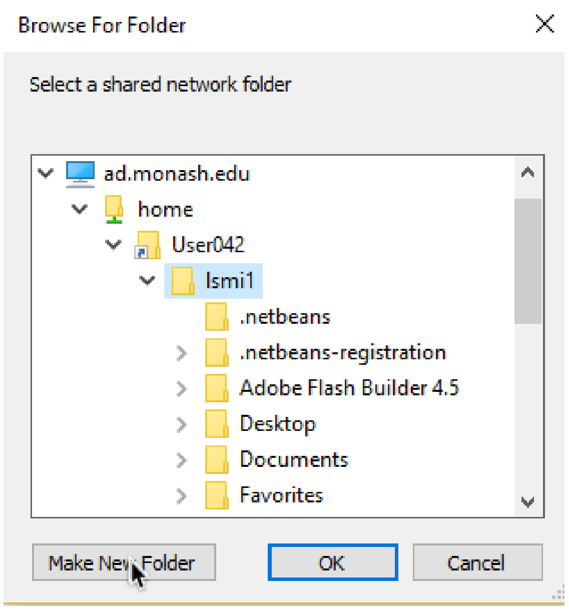
Specify the drive letter for the connection and the folder that you want to connect to:

Drive: U:

Folder: \\ad.monash.edu\home\User042\lsmi1

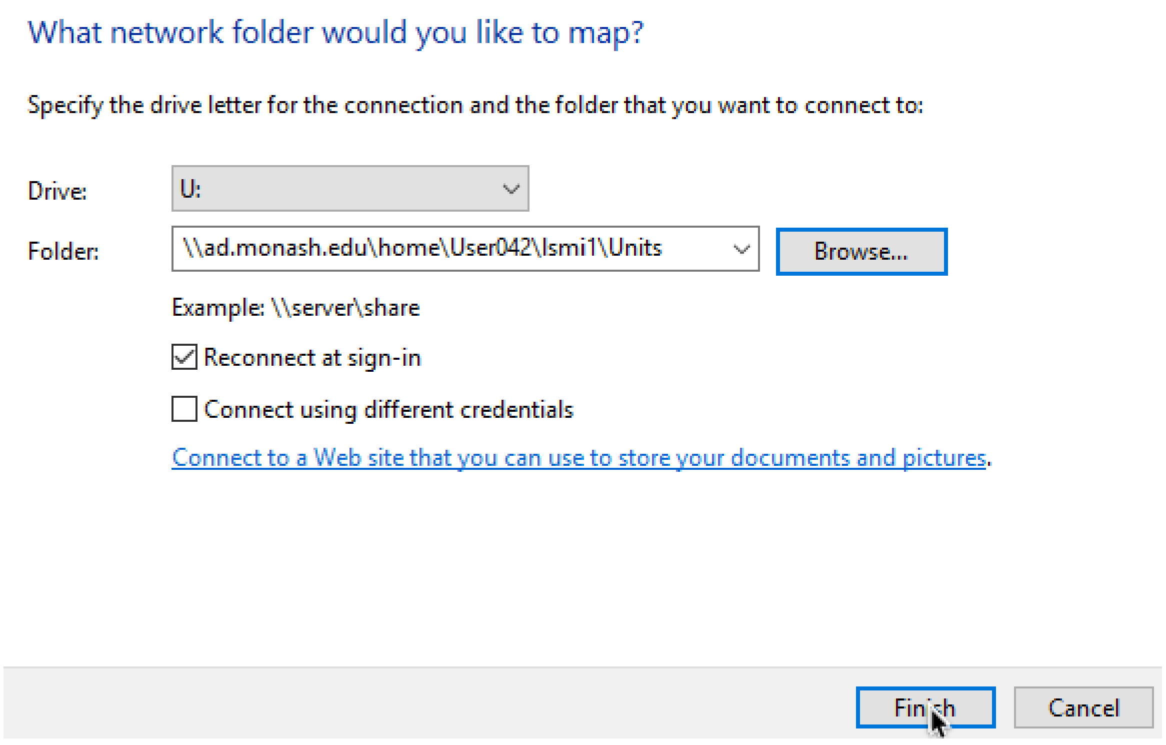
Browse...
Example: \\server\share
☑ Reconnect at sign-in
☐ Connect using different credentials
Connect to a Web site that you can use to store your documents and pictures.

Be sure to check "Reconnect at sign-in", then select Browse



and add a new Folder via "Make New Folder" under your authcate username. We suggest you call the folder "Units" - you can then create subfolders below this (after we have mapped the drive) for the various units you will study throughout your course. This setup only needs to be done once.

Finally, after creating the folder and selecting OK:


What network folder would you like to map?

Specify the drive letter for the connection and the folder that you want to connect to:

Drive: U:

Folder: \\ad.monash.edu\home\User042\lsmi1\Units

Example: \\server\share

[✓] Reconnect at sign-in

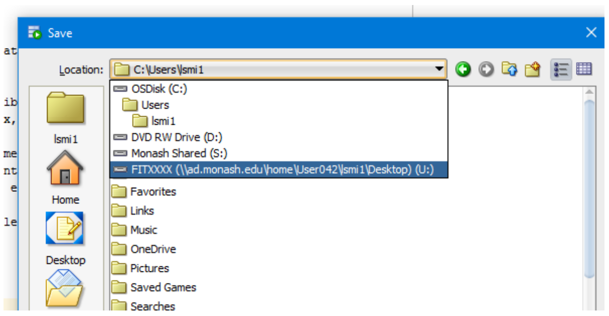
[ ] Connect using different credentials

Browse...
Connect to a Web site that you can use to store your documents and pictures.
Finish
Cancel

select Finish (double check "Reconnect at sign-in" is selected first). You now will have a drive U: below which you can create unit based subfolders. This approach has the advantage that the files you save on drive U: will be saved immediately to the drive. If you save anything to the Desktop these files are only transferred when you log out and can substantial slow down your login and logout times.

Please note that students currently have a space limit of 5Gb on this server.

You will then be able to load and save files in the on-campus labs from your mapped drive eg. U: and access this drive from *any* University Windows Lab PC that you login to. To reach the drive you may need to pull down the SQL Developer folders list:

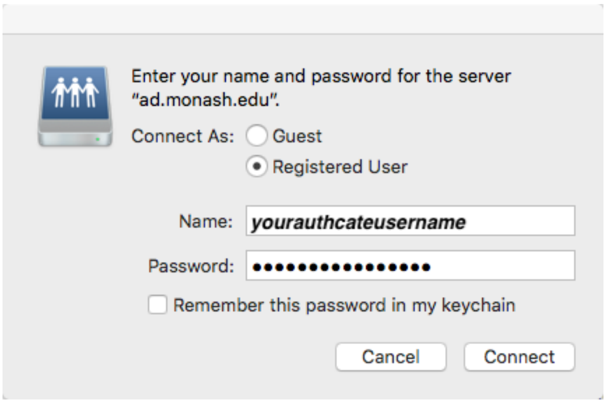

Save

Location: C:\Users\lsmi1

- OSDisk (C:)
  - Users
    - lsmi1
- DVD RW Drive (D:)
- Monash Shared (S:)
- FITXXXX (\\ad.monash.edu\home\User042\lsmi1\Desktop) (U:)

To access drive U: from the University MS Windows labs add the batch file, available under Moodle, to your University desktop. Simply double click on it when you first log in and your mapping will be recreated. The same batch file can be run from MS Windows at home, or on your laptop, *provided* the VPN is active - you will be prompted for your username (enter as MONASH\username) and authcate *password*.

You can also access this share on a Mac via Samba, *provided* the VPN is active. In Finder select Go - Connect to Server and enter a "Server Address" in the form smb://ad.monash.edu/home/User*1234/username/*Units *(replacing 1234 and username to point to your share)*. The Mac OS implementation of samba shares has a few "issues", one of which is having the share reliably listed in the "Shared" sidebar of Finder (you may see ad.monash.edu but not be able to navigate to your Units folder). The approach you should take to address this is, under Finder Preferences, General ensure that "Connected servers" is checked to show on your desktop before attempting to connect. When prompted for your username simply use your authcate username do not preface it with MONASH\.


Enter your name and password for the server "ad.monash.edu".

Connect As: Guest [✓ Registered User]

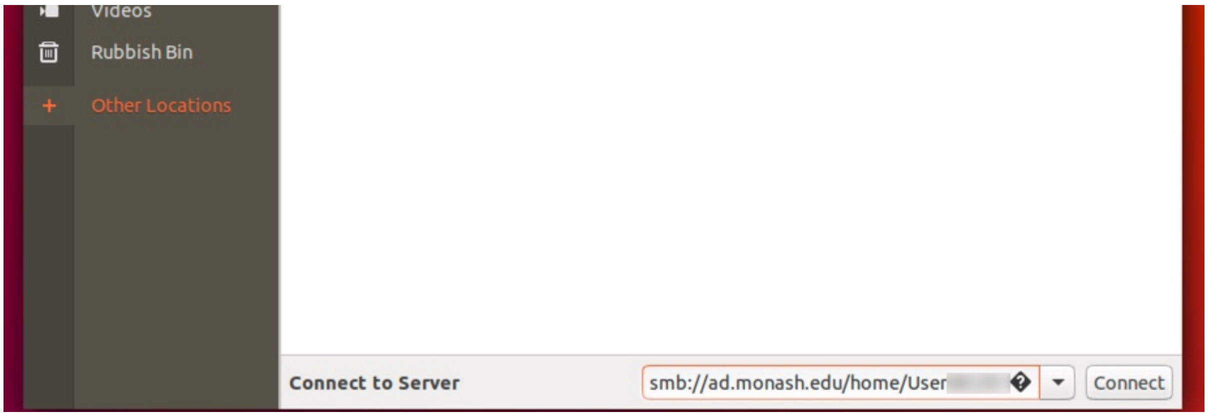
Name: *yourauthcateusername*

Password: ●●●●●●●●●●●●●●●●

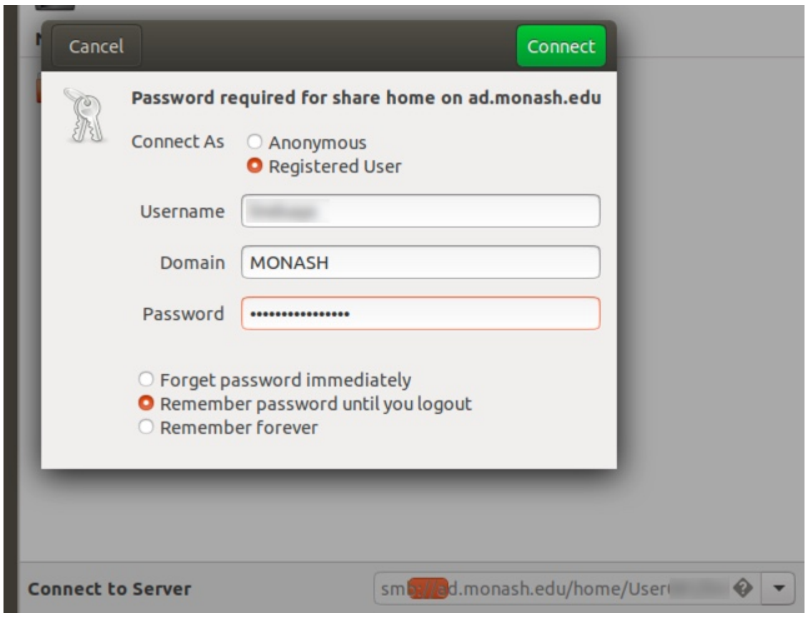
☐ Remember this password in my keychain

When you connect to your share you will see SYSVOL, NETLOGON (which you should ignore) and your UNITS drive displayed on your desktop. Double-clicking on Units will open the drive in Finder.

The share can also be accessed via Samba under Linux distros such as Ubuntu. If you are off campus you will need to first install the CISCO VPN software available from vpn.monash.edu (note that if the software does not open correctly with a GUI you will need to resolve a missing dependency for the pango library - see here https://zngguvnf.org/2017-12-04-ubuntu-17-10-and-cisco-anyconnect.html (https://zngguvnf.org/2017-12-04--ubuntu-17-10-and-cisco-anyconnect.html)). To access the share - open Files, Select Other Locations and enter your smb path into the "Connect to Server" field, when prompted enter your login details:



Connect to Server smb://ad.monash.edu/home/User



Password required for share home on ad.monash.edu

Connect As ( ) Anonymous [✓ Registered User]

Username

Domain MONASH

Password ••••••••••••••••

( ) Forget password immediately
[✓ Remember password until you logout]
( ) Remember forever

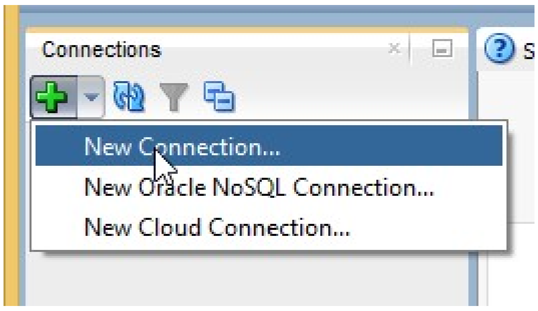
Connect to Server sm[redacted]d.monash.edu/home/User

1.1
Connecting to Oracle database using SQL Developer

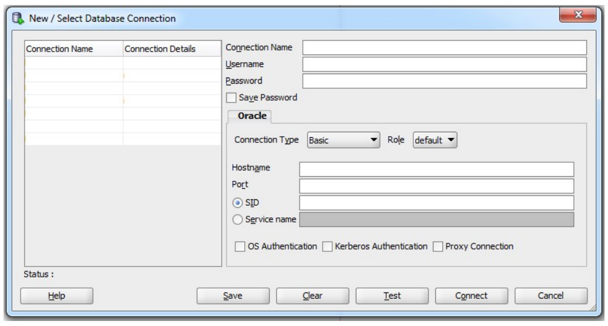
In the next few sections, you will learn how to use your newly configured SQL Developer software to access an Oracle database.

1. Adding a new connection

After running SQL Developer, right click on the Connections icon in the left panel, as shown below and select the New Connection option.

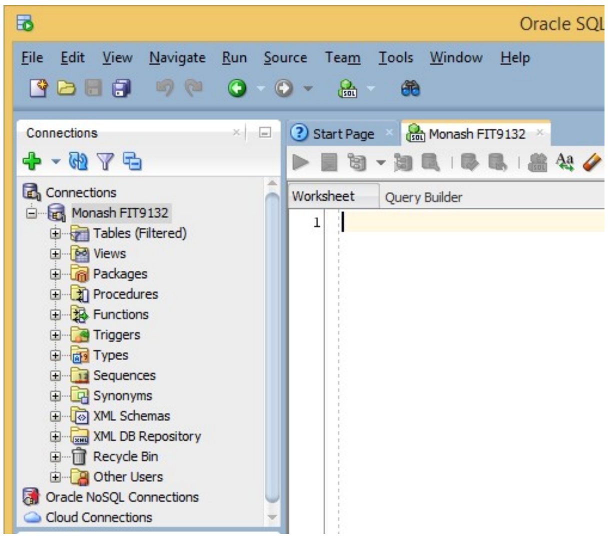


Now you will see the New / Select Database Connection window, as shown below.



The connection details that you need to connect to the Oracle database through SQL Developer will be provided by your tutor/lecturer. As well as setting up this connection, it is very important that you configure SQL Developer correctly (as explained at the start of Module 1).

After connection you will see:



SQL Statements can be entered in the right hand panel, labelled "Worksheet".

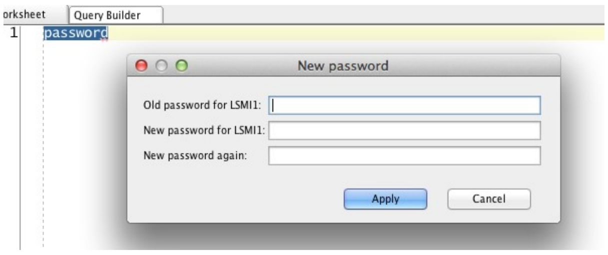
2. Changing your password

After logging in for the first time you should change your password from the one supplied by your tutor. **DO NOT** set your Oracle password the same as your standard authcate password.

Oracle has several **important** limits on the password you set:

- The password **is case sensitive**
- May be 1 - 30 characters in length
- Must begin with an alphabetic character
- Can contain only alphanumeric characters and the underscore (_) or dollar sign ($)

You should set your password in SQL DEVELOPER using the PASSWORD command (your password is hidden). Type the word password in your worksheet and click on the Execute Statement button (the green arrow) in the toolbar to run the command:



```
password
```

Old password for LSMI1:
New password for LSMI1:
New password again:

After changing your password for the first time please log out and then re-log back in to check that your password change has been successful.

If at any stage you find that you are unable to login to Oracle due to password problems ***please email your tutor to have your password reset***.

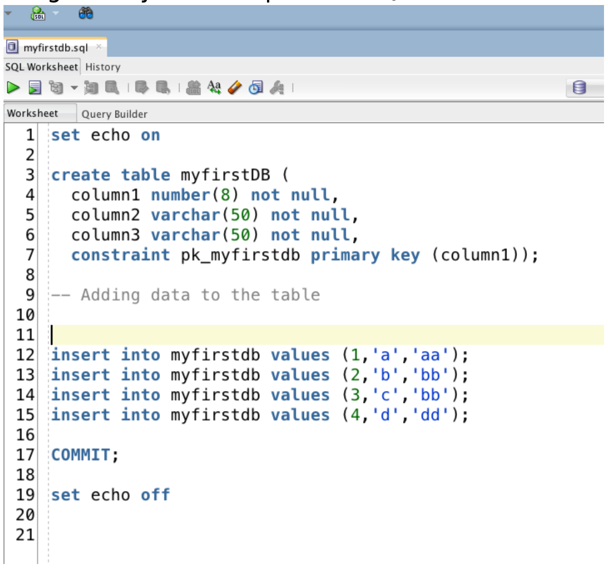
2
Using SQL Developer GUI to manage data

There are two main approaches to interact with Oracle database using SQL Developer, through the Graphical User Interface (GUI) and SQL Worksheet. In this module, you will learn to use the SQL Worksheet to create the database and to use the GUI to add, update and delete data from the database.

1. Opening an SQL file in the SQL Worksheet.

We will use SQL worksheet to create the database. Follow these steps:

1. Download the myfirstDB.sql from Moodle.
2. Open SQL Developer on your machine.
3. Open the connection to an Oracle server.
4. Drag the myfirstDB.sql to the SQL Worksheet area.

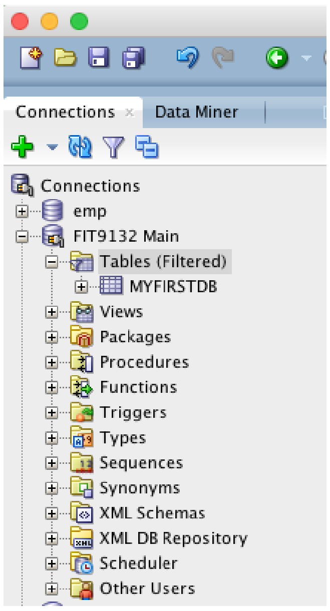


```
set echo on

create table myfirstDB (
  column1 number(8) not null,
  column2 varchar(50) not null,
  column3 varchar(50) not null,
  constraint pk_myfirstdb primary key (column1));

-- Adding data to the table


insert into myfirstdb values (1,'a','aa');
insert into myfirstdb values (2,'b','bb');
insert into myfirstdb values (3,'c','bb');
insert into myfirstdb values (4,'d','dd');

COMMIT;

set echo off

```

5. Run the SQL script by pressing the "run script" icon.

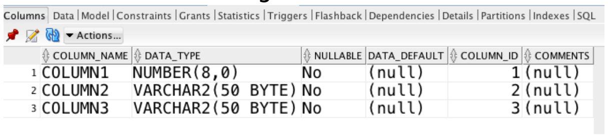
2. Viewing table structure and data inside a table.

1. To view the table using the graphical user interface, expand the Table option in the Connection tab and find "myfirstdb" from the list.



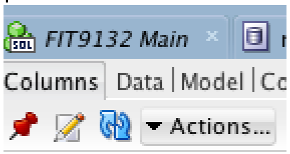
Double click on the "myfirstdb" entry.

2. You will see the listing of the database structure of the "myfirstdb" database.

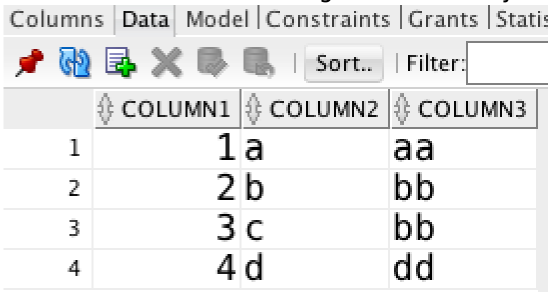

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL
Actions...

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	COLUMN1	NUMBER(8,0)	No	(null)	1	(null)
2	COLUMN2	VARCHAR2(50 BYTE)	No	(null)	2	(null)
3	COLUMN3	VARCHAR2(50 BYTE)	No	(null)	3	(null)

3. To view and change the data inside the myfirstdb, click on the "Data" tab on the right-hand panel:



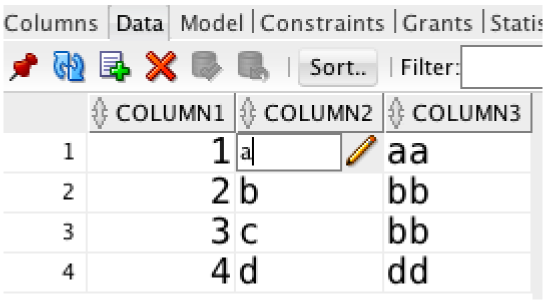
4. You will see the data listing inside the "myfirstdb":



	COLUMN1	COLUMN2	COLUMN3
1	1	a	aa
2	2	b	bb
3	3	c	bb
4	4	d	dd

3. Updating the data inside a table.

1. Double click on the cell where the data needs to be changed. For example, let's change the content of row-1 of column2 from "a" to "z".



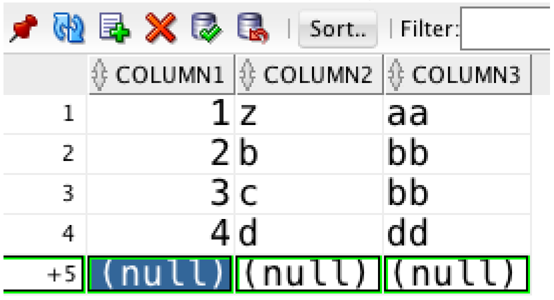
	COLUMN1	COLUMN2	COLUMN3
1	1	a	aa
2	2	b	bb
3	3	c	bb
4	4	d	dd

2. You will see an asterisk for the first row. To accept the changes made on the row(s) with an asterisk, you will need to issue a COMMIT by pressing the "tick" icon.

3. When the database accepts the COMMIT instruction you have made, the asterisk should disappear from row 1.

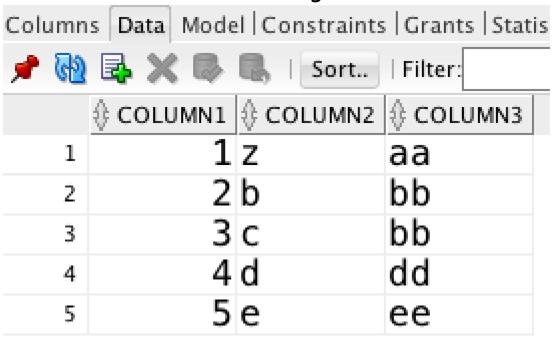
4. Adding a new row to the table.

1. Click on the insert new row icon.
2. You will see a new row is being added to the table with all values listed as (null).



	COLUMN1	COLUMN2	COLUMN3
1	1	z	aa
2	2	b	bb
3	3	c	bb
4	4	d	dd
+5	(null)	(null)	(null)

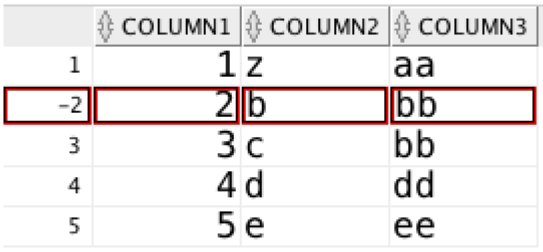
3. Replace the (null) value with 5, e and ee.
4. Click on the COMMIT icon to save the changes.
5. You will see the new being added to the table.



	COLUMN1	COLUMN2	COLUMN3
1	1	z	aa
2	2	b	bb
3	3	c	bb
4	4	d	dd
5	5	e	ee

5. Deleting a row from the table.

1. Click on the row to be deleted, for example, choose row 2.
2. Once the row is highlighted, choose the delete icon from the menu.
3. You will see the row to be marked with a negative sign at the front of the row number.



	COLUMN1	COLUMN2	COLUMN3
1	1	z	aa
-2	2	b	bb
3	3	c	bb
4	4	d	dd
5	5	e	ee

4. Click on the COMMIT icon to save the deletion.
5. The second row now should contain the data 3, c, bb.

2.1 Data Anomalies

A poor database design will lead to problems during the operation of the database. The following exercises will help you to identify anomalies(problems) associated with a poorly designed database.

First, you will need to create the database and to populate the data. To perform this task, you need to make sure you have completed the module Introduction to SQL Developer.

Creating a database

1. Download the file student_poor.sql and student_good.sql from Moodle.
2. Open the SQL Developer software.
3. Connect to the Oracle server.
4. Open the files you have downloaded from Moodle in SQL Developer.
5. Run the student_poor.sql.
6. Run the student_good.sql.

Manipulating data inside the database

There are two databases you have created after you completing steps 5 and 6.

1. The poorly designed database contains a single table called student_poor.
2. The well-designed database contains three tables called
    - **student_good**
    - **unit_good**
    - **enrolment_good**

Tasks

- Locate the listing of these tables in SQL developer to check whether you have completed step 5 and 6 of Creating database correctly.
- Performs the database operations instructed below and observe the impact of the action on the data. Several questions have been provided to help your observation.
- For each of the action, perform the action in both the poorly designed and well designed databases. Compare the observations when the action took place in those databases.

DATABASE OPERATIONS

UPDATE

1. Change the name of FIT9131 for student number 1111 into Foundation of Java in the student_poor table.
2. Observe the data in the student_poor table.
    - Can you make the change?
    - How many different names does FIT9131 have? What will be a potential problem with this situation?
    - If you want to ensure the name of FIT9131 to be consistent, how many rows did you suppose

to change in the database? Is it possible for you to easily check that all relevant rows have been updated?

3. Now, perform the same update in the well design database.
    - What table do you need to change to reflect the required change?
    - How many row(s) do you need to change?
    - How does this design ensure consistency of the unit name?
4. What problem(s) does the well design database overcome?

DELETE

1. Wendy Wang decided to withdraw from FIT9133. You are required to delete this enrolment from student_poor table.
2. Observe the data in the student_poor table.
    - Can you find out from the database the name of a unit with the code FIT9133 after the delete operation is completed?
    - Can you find out from the database the student number of Wendy Wang after the delete operation is completed?
3. Now, perform the same deletion in the well design database.
    - In what table the data need to be deleted?
    - Can you find the student number of Wendy after the deletion?
    - Can you find the unit name for FIT9133 after the deletion?

INSERT

1. A new unit FIT5000 Data Analytics is introduced in the course. Add the new unit code and unit name to the student_poor table.
2. Observe the data in the student_poor table.
    - Can you perform the insert operation? What error message did you get?
    - What additional information do you need to include to add a new unit information to the student_poor table? Is it a good idea? Why/Why not?
3. Now, perform the same insertion in the well design database.
    - In what table the data need to be added?
    - Can you add the new unit information in the database without having any student enrolled into the unit?

Summary

Designing what data and how it is stored in an organisation is important. A poorly design data store may lead to problems such as those you have seen in the exercises. The poor_student table used in this module contains three different things/objects (physical object - student and abstract concepts - unit, enrolment). These three objects/concepts are separated into three different tables in the well design database. This is how data should be typically stored in the database. In this unit, you will learn how to ensure that you will build database without the associated problems shown in this exercise. This will be done by looking at Relational model, its theory and design methodology.

3
The Relational Data Model FIT3171

In this week's work, we will look at the fundamental concepts on which the Relational Database model is built.

1. Discuss the following terms

- Relation
- Attribute
- Domain
- Tuple
- Degree and Cardinality of a Relation

2. Consider the CUSTOMER and ORDER relations below:

CUSTOMER (CUSTOMER-ID, NAME, ADDRESS)
ORDER (ORDER-ID, DATE, CUSTOMER-ID)

Assume a single customer may have any number of orders.

1. Identify the primary key and foreign key attributes in these relations.
2. Can you think of a reason why we would not just store all the customer and order information in one relation so that we would not have to perform the join operation?

3. Choosing Primary key

1. In any relation, tuples must be unique. However, in many cases, the set of all the attributes in a relation is not considered a candidate key. Why not?On the other hand, suppose we do have a relation where the set of all attributes is a candidate key. In this case, show that this set must, therefore, be the only candidate key and hence the primary key.
2. Consider a relation that depicts a tutorial room booking in a university. Each faculty assigns a person to handle the booking for all tutorial classes for that faculty. The person's email address is given to the university's booking system as a contact person.
   BOOKING(b_date, b_starttime, b_endtime, unit_code, contact_person, room_no, tutor_id)
   - Identify candidate key(s) and primary key for the relation if the following business rules are applicable:
     - More than one tutorial classes of the same unit may run at the same time (parallel sessions are possible).
     - A tutor may teach several classes of the same unit.
     - All tutorial classes are 2 hours long.
   - Identify candidate key(s) and primary key for the relation if the following business rules are applicable:
     - Tutorial classes can be either 1 hour or 2 hours long.
     - A tutor can only teach one tutorial class in a given unit.
     - There are no parallel sessions of tutorial classes.

4. Relational Algebra (Adapted from Exercise 3.6 of Connolly, Begg and Strachan)

Suppose we have the following 4 relations:

HOTEL(HOTEL-NO, NAME, ADDRESS)
ROOM(ROOM-NO, HOTEL-NO, TYPE, PRICE)
BOOKING(HOTEL-NO,GUEST-NO, DATE-FROM, DATE-TO, ROOM-NO)
GUEST(GUEST-NO, NAME, ADDRESS)

Generate the relational algebra for the following queries:

1. List the names and addresses of all hotels
2. List all single rooms with a price below $50
3. List the names and addresses of all guests
4. List the price and type of all rooms at the Grosvenor Hotel
5. List all names and addresses of guests currently staying at the Grosvenor Hotel (assume that if the guest has a tuple in the BOOKING relation, then they are currently staying in the hotel)

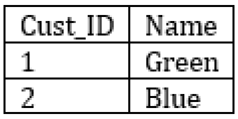
5. There are 7 relational algebra operators, namely:

SELECTION, PROJECTION, JOIN, UNION, INTERSECTION, DIFFERENCE and CARTESIAN PRODUCT. In fact, 5 of these operators may be considered primitive operators in the sense that the others may be expressed in terms of the primitive operators. The primitive operators are:

SELECTION, PROJECTION, UNION, DIFFERENCE and CARTESIAN PRODUCT

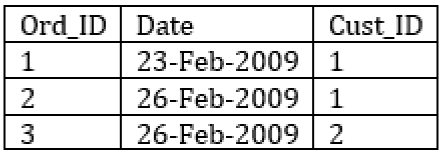
Using the sample tables below, show how the JOIN operation can be expressed in terms of the fundamental operators by showing the process to do a natural join of customer and order.

- **CUSTOMER table:**



Cust_ID	Name
1	Green
2	Blue

- **ORDER table:**



Ord_ID	Date	Cust_ID
1	23-Feb-2009	1
2	26-Feb-2009	1
3	26-Feb-2009	2

6. The database below is for a department store and describes stock, staff, clients, and sales.

ITEM

ITEM	DESCRIP	PRICE
K3	Knife Set	$17.95
K5	Ladle	$6.95
K11	Scraper	$0.95
L12	Rack	$22.95
L3	Table	$399.50
L6	Stool	$17.95

SALE

ITEM	STAFF	NUMSOLD	CLIENT	DATE
K3	Simon	6	Clark	20170311
K11	Simon	1	Cilla	20170121
K11	Simon	1	Cilla	20170123
L12	Sorcha	5	Charles	20161130
K3	Sean	1	Clive	20170221
L12	Sean	1	Cilla	20170228
L12	Simon	2	Clive	20170228
K3	Sean	2	Charles	20161129

STAFF

NAME	POSITION
Sandra	Manager
Simon	Clerk
Steve	Packer
Sean	Clerk
Sorcha	Director
Sian	Clark

STOCK

ITEM	NUMSTOCK
K3	105
K11	66
L3	0
L12	4
L6	13

CLIENT

NAME	**ADDRESS**
Clive	India Rd
Clark	Kent St
Charles	Windsor Av
Cilla	Black St

Using Relational Algebra answer the following queries. You must represent your answer in symbolic notation and where a query has several solutions, your answer must represent the most efficient solution.

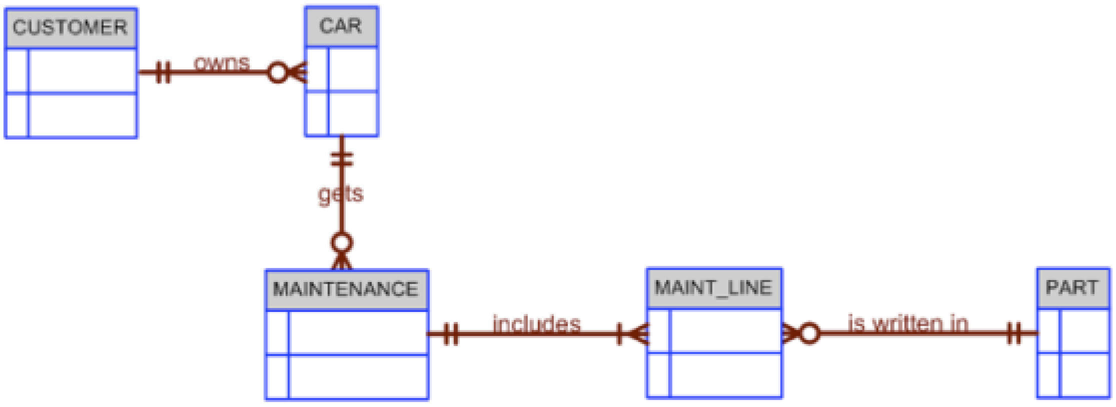
1. What items are out of stock?
2. What items haven't been sold?
3. Which clerks don't have any sales?
4. What categories (positions) of staff have made sales?

4 Conceptual Modelling

Conceptual Design - Theory Questions

Reference: Coronel, C and Morris, Database Systems: Design, Implementation & Management, Chapter 4, Selected Review Questions and Problems.

1. What two conditions must be met before an entity can be classified as a weak entity? Give an example of a weak entity.
2. What is a identifying relationship, and how is it depicted in a Crow's Foot ERD?
3. Given the business rule "an employee may have many degrees," discuss its effect on attributes, entities, and relationships. (Hint: Remember what a multivalued attribute is and how it might be implemented.)
4. What is a composite entity, and when is it used?
5. Suppose you are working within the framework of the conceptual model in Figure Q4.5.Figure Q4.5 The Conceptual Model for Question 5



   o Write the business rules that are reflected in it.

   o Identify all the cardinalities.

6. What is a recursive relationship? Give an example.
7. How would you (graphically) identify each of the following ER model components?

   o an entity

   o the cardinality (0,N)

   o a weak relationship, and

   o a strong relationship

8. Discuss the difference between a composite key and a composite attribute. How would each be indicated in an ERD?
9. What two courses of action are available to a designer when he or she encounters a multivalued

attribute?

10. What is a derived attribute? Give an example.
11. How is a relationship between entities indicated in an ERD? Give an example, using the Crow's Foot notation.
12. Discuss two ways in which the 1:M relationship between COURSE and CLASS can be implemented. (Hint: Think about relationship strength.)
13. How is a composite entity represented in an ERD, and what is its function? Illustrate using the Crow's Foot model.
14. Briefly, but precisely, explain the difference between single-valued attributes and simple attributes. Give an example of each.
15. What are multivalued attributes, and how can they be handled within the database design?

4.1 Using Tools to draw ERD

Tools to draw ER diagrams

There are several tools available to draw ER diagrams. Some of them are available to be used within a web browser. Some examples of these are:

● Lucidchart (https://www.lucidchart.com/) - this product is a browser-based diagramming tool; it is able to draw a wide range of different diagrams, including ER Diagrams, or

● any other drawing package you wish. One excellent alternative is Gliffy (https://www.gliffy.com/)

At this stage of our study, we do not wish to use a CASE tool - it is important that we first establish a clear understanding of Entity Relationship modelling.

Setting up Lucidchart

The Lucidchart Education account details and sign up are here (https://www.lucidchart.com/pages/usecase/education).

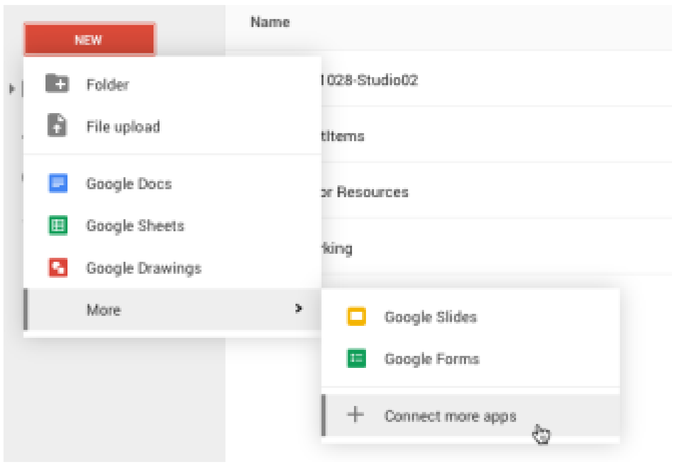
Students *must create their own account* by signing up for a free account with your Monash educational email address at the URL listed above. This will result in an email being sent to your student account confirming your account details and providing a link to set your password. Alternatively, you can set your password when you first login see (see the top left of your first screen).

As a first step, you should look at the provided tutorials, in particular, "Entity Relationship Diagrams (https://lucidchart.zendesk.com/entries/21606135-Entity-relationship-diagrams)" (begin with "Manual ERD Creation", note that the model we are asking you to build should not include the "Type" column)

After you have set up your account, Lucidchart can be logged into directly at the Lucidchart site: https://www.lucidchart.com/ or alternatively accessed from your student Google Drive (from my.monash select the "Drive" link).

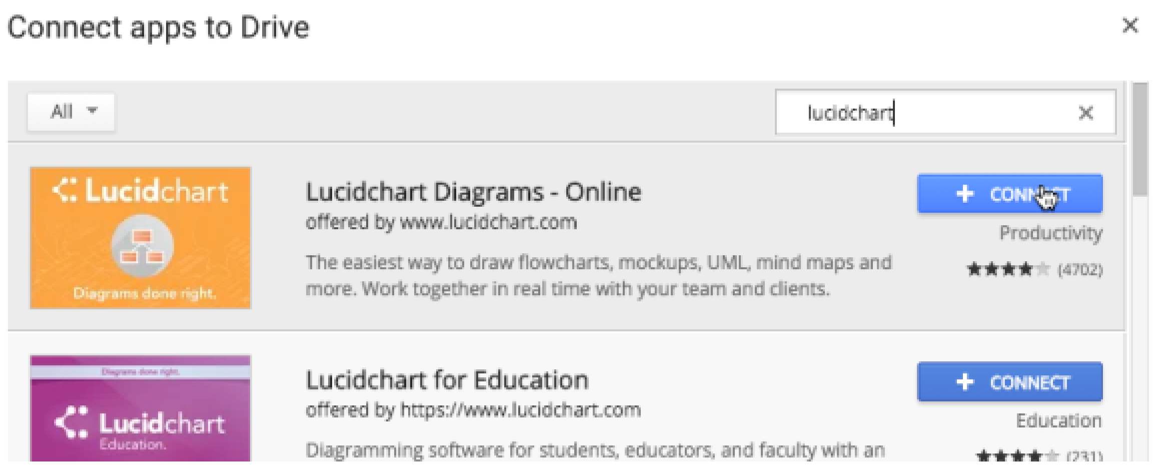
Connect LucidChart to your Google Drive [OPTIONAL]

Connect to Google Drive, click "New" and then more, and "Connect more apps"



This will open the app picker which lists a wide range of applications that can be added to your Google drive.

You can browse through and find Lucidchart or alternatively type "Lucidchart" in the top right search box:


Connect apps to Drive
All
lucidchart
Lucidchart
Diagrams done right.
Lucidchart Diagrams - Online
offered by www.lucidchart.com
The easiest way to draw flowcharts, mockups, UML, mind maps and more. Work together in real time with your team and clients.
CONNECT
Productivity
(4702)
Lucidchart Education.
Lucidchart for Education
offered by https://www.lucidchart.com
Diagramming software for students, educators, and faculty with an
CONNECT
Education

Select the "+ CONNECT" button for Lucidchart Diagrams - Online

After this process has been completed Lucidchart charts created from within your Google Drive will be stored in your Google Drive and on the Lucidchart server. Full details of the integration are available here (https://lucidchart.zendesk.com/hc/en-us/articles/207300016-Lucidchart-in-Google-Drive).

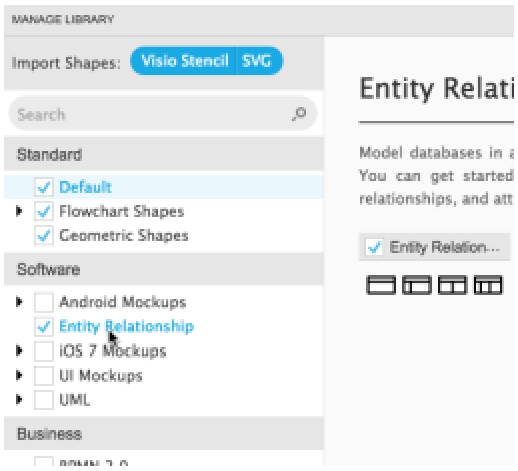
To create a new Lucidchart - simply select New in Google Drive, and then select "Lucidchart" (you may need to expand "More" to see the Lucidchart option).

If this is the first document you have created as a Lucid Chart from your Google Drive you will be asked to approve Lucidchart's access to your Google Drive - please ensure you select "Accept". This will then transfer you into the Lucidchart workspace with a new document open.

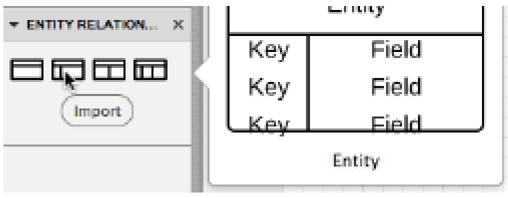
Creating a new LucidChart Diagram

Either use Google Drive, as above, or login directly to LucidChart. If you log in directly you will need t0 select to create a new "Entity Relationship (ERD)", then Blank ERD and then "Start Drawing".

If the ERD shapes are not listed in the left panel, add the ERD shapes by selecting "More Shapes" (bottom left) and then checking "Entity Relationship" and then "Save":



The symbol we will use to represent an entity is the second symbol from the left:

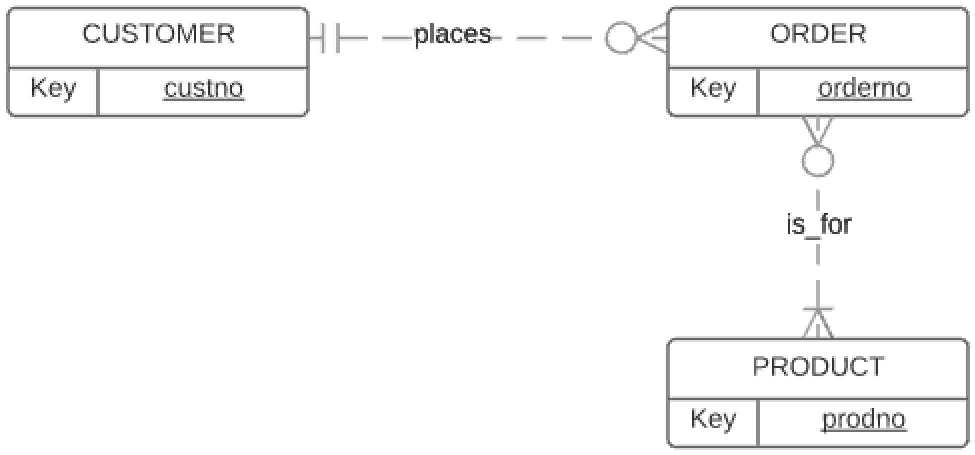


Using Lucidchart

Given a scenario represented by the following entities, where customers place orders for products:

- CUSTOMER - customer number, name, address, phone number
- ORDER - order number, order date and for each product ordered the quantity ordered and the total line price
- PRODUCT - product number, product description and product unit price

An initial ERD using Lucidchart for this scenario would be:



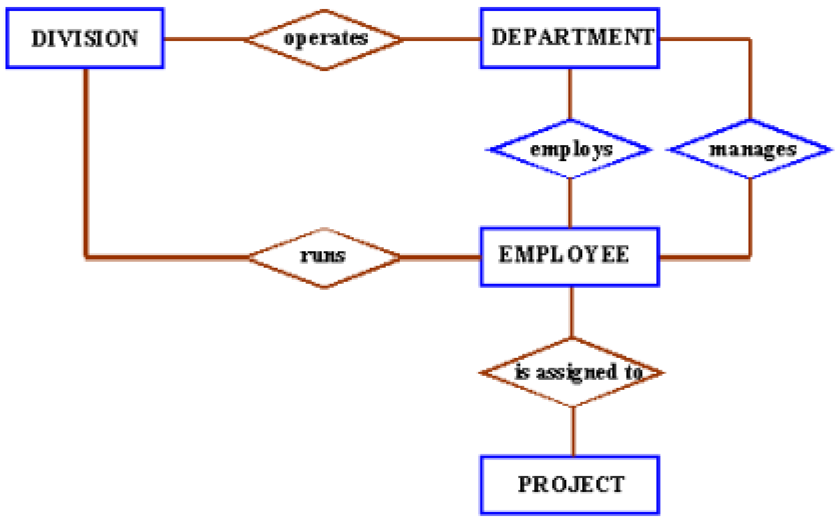
This ERD only shows the keys of each of the entities, sometimes an ERD is drawn such that it will show all of the attributes for the scenario. In such a complete (or full) ERD you must not show or label foreign keys, the use of foreign keys indicates that you are looking at a logical model where a choice has been made to use a relational database, rather than a conceptual model.

Prepare the ERD shown above using your choice of drawing tool and then create a copy of this initial model and add all attributes listed in the scenario above to your copy. In Lucidchart, you can easily create a copy by selecting the bottom tab of the diagram, click the down arrow and then select "Duplicate".

4.2 Building Conceptual Models

Conceptual Modelling - Practical Work

For this weeks lab exercises, we are going to prepare conceptual models (Entity Relationship Diagrams - ERDs) for a number of scenarios. At this point we are not interested in the database implementation of these models, our aim will be to model without any consideration of the database system in which the model may ultimately be implemented.



1. Use the above diagram and the business rules below, create an Entity Relationship Diagram using crows-foot notations.
   - Include:
     - all appropriate connectivities,
     - all cardinalities and
     - at least the minimum number of attributes required to implement the model
   - Business Rules:
     - A department employs many employees, but each employee is employed by one department.
     - Some employees, known as "rovers," are not assigned to any department.
     - A division operates many departments, but each department is operated by one division
     - An employee may be assigned to many projects, and a project may have many employees assigned to it.
     - A project must have at least one employee assigned to it.
     - One of the employees manages each department, and each department is managed by one employee
     - One of the employees runs each division, and each division is run by one employee.
2. Prepare an Entity Relationship Diagram (ERD) showing all attributes and the identifier of each entity for the following description of a Property Rental System:

- 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 building 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 instance of damage to its properties - property, date, type of damage and repair cost are recorded. Repair costs are charged directly to tenants
- Normal property maintenance is also noted - property, date, type of maintenance and cost are recorded. Maintenance costs are charged to the property owner.
- 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.

5
Logical Modelling

Logical Modeling

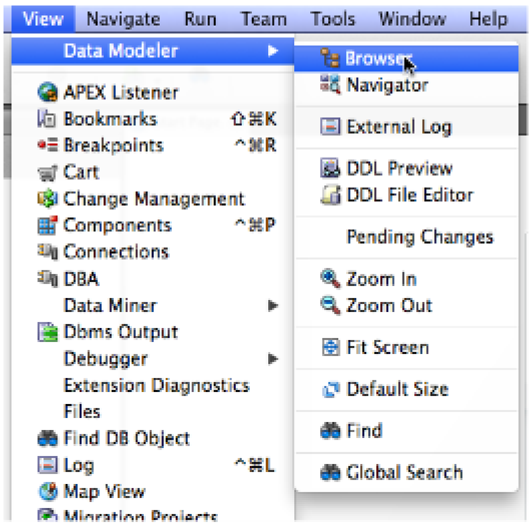
After preparing a conceptual model the next stage is to select the type of database we will use (for us relational) and convert our conceptual model into an appropriate logical model. For logical modelling, we will make use of Oracle SQL Developer Data Modeler.

Task A: Using SQL Developer Data Modeler

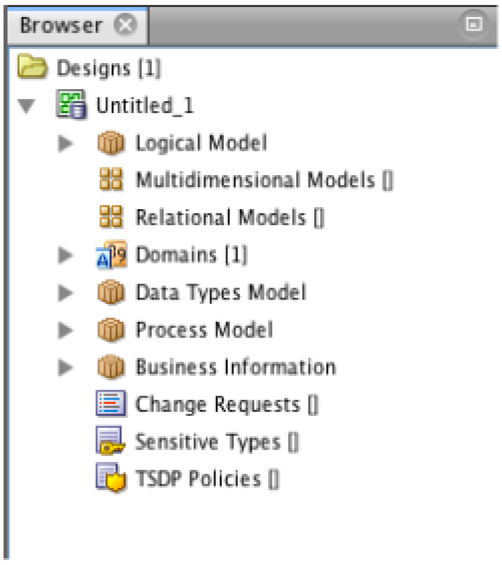
This software is a commercial level tool with an extensive set of features including support for Subversion versioning and source control thus permitting teams of developers to work collectively on a design. Given the extensive range of features, we are only going to be looking at a subset of these for 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.

Accessing Data Modeler

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

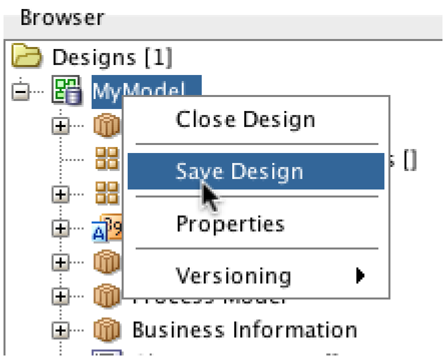


The browser opens with a new (unnamed) model:

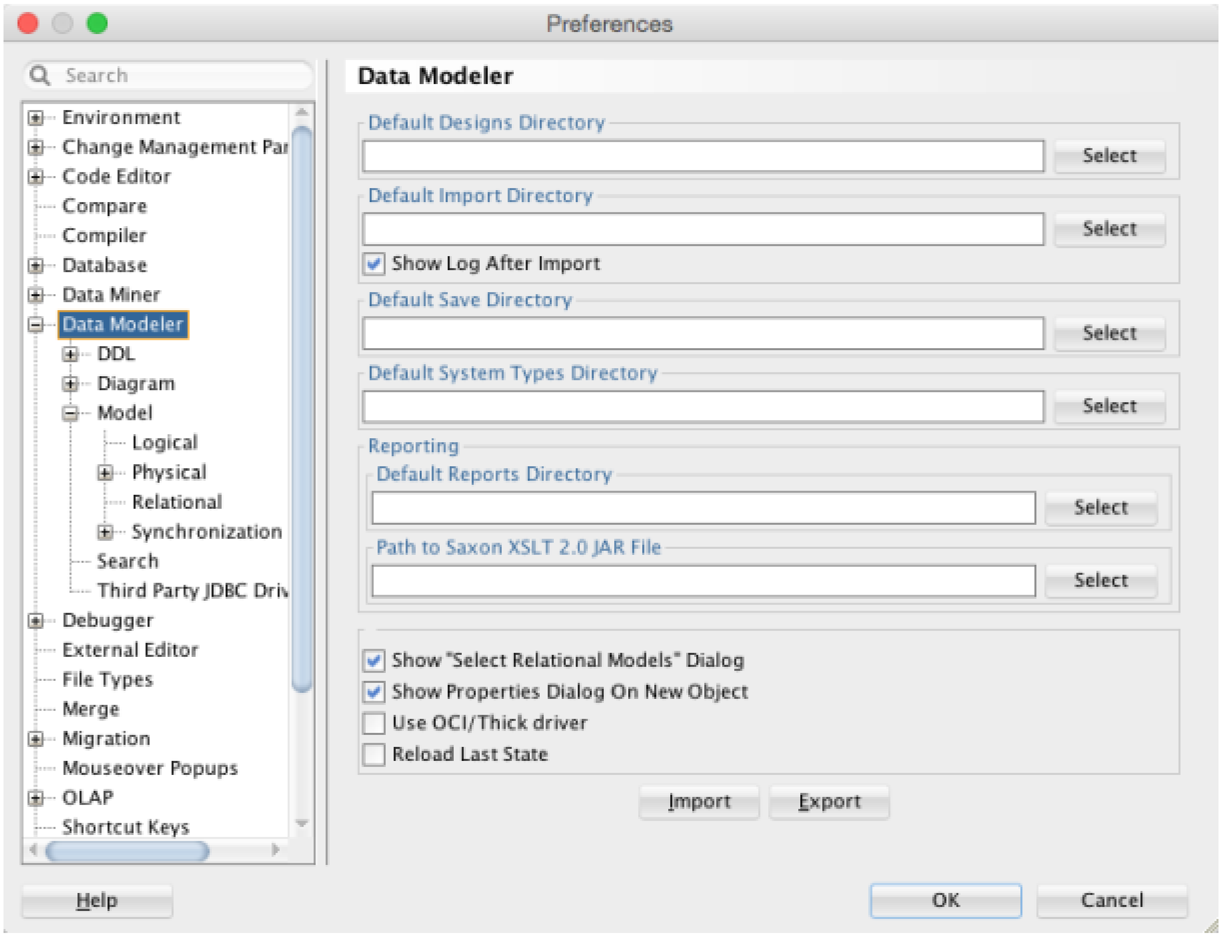


To begin creating a logical model, right-click Logical Model and select Show. This will open a blank model in the main working panel of SQL Developer and add a range of new icons to the main toolbar. Hover over each of these new icons to become familiar with what they represent. You should regularly save your design - right click on the design (here Untitled_1 as the design has not been saved as yet) and select a save location, after the first save the design will be named.

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.



A number of features should be configured using the SQL Developer preferences (Windows: Tools-Preferences, OSX: Oracle SQL Developer-Preferences), select the Data Modeler:



Data Modeler

Default Designs Directory

Default Import Directory

☑ Show Log After Import

Default Save Directory

Default System Types Directory

Reporting

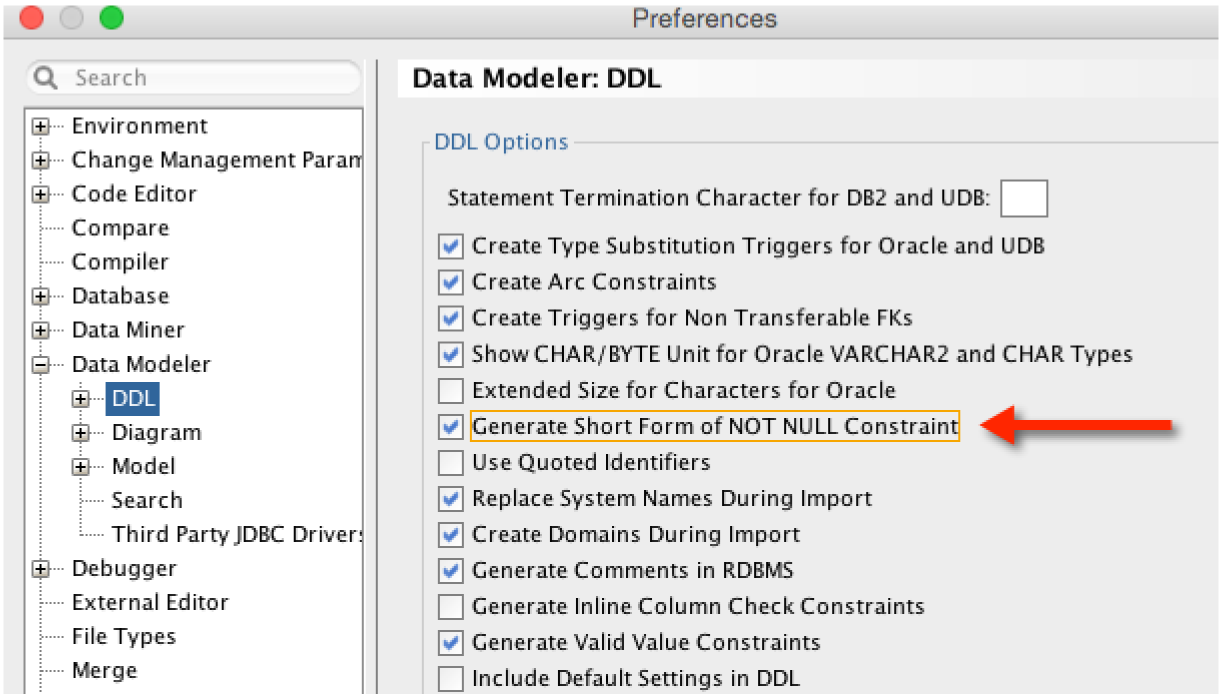
Default Reports Directory

Path to Saxon XSLT 2.0 JAR File

☑ Show "Select Relational Models" Dialog
☑ Show Properties Dialog On New Object
☐ Use OCI/Thick driver
☐ Reload Last State

We wish to modify two 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
☐ Include Default Settings in DDL

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

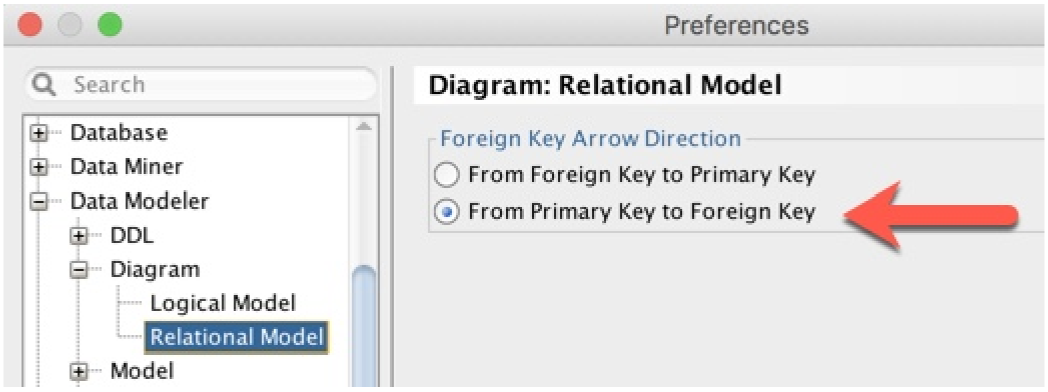
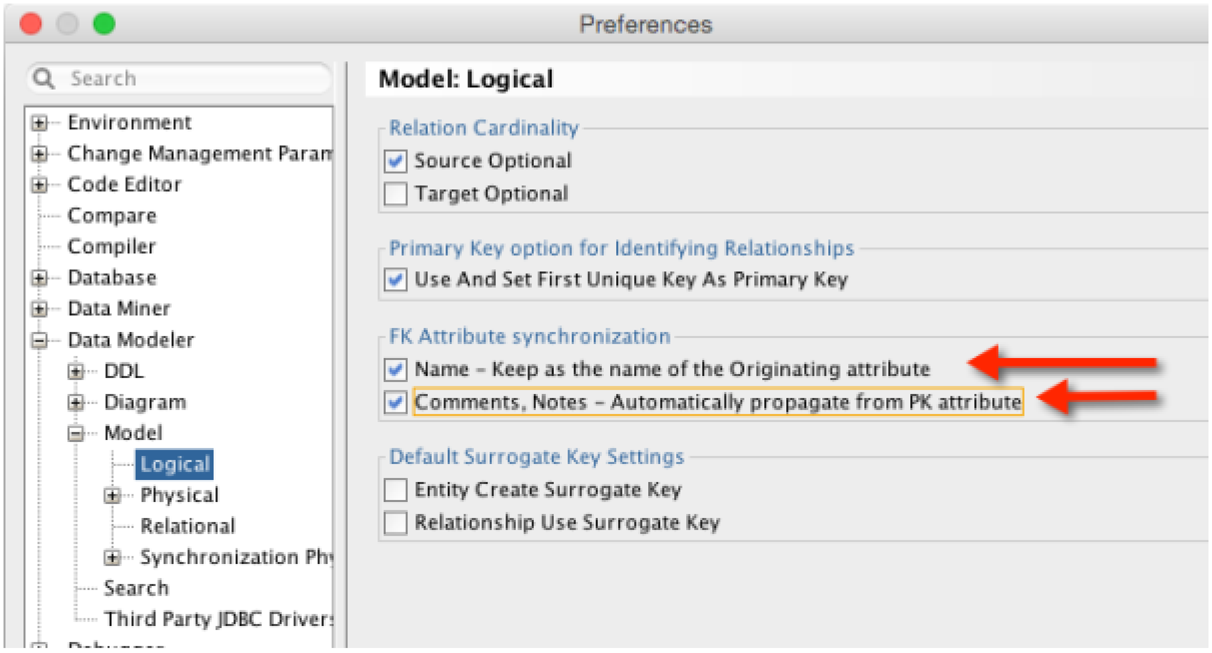


Diagram: Relational Model

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

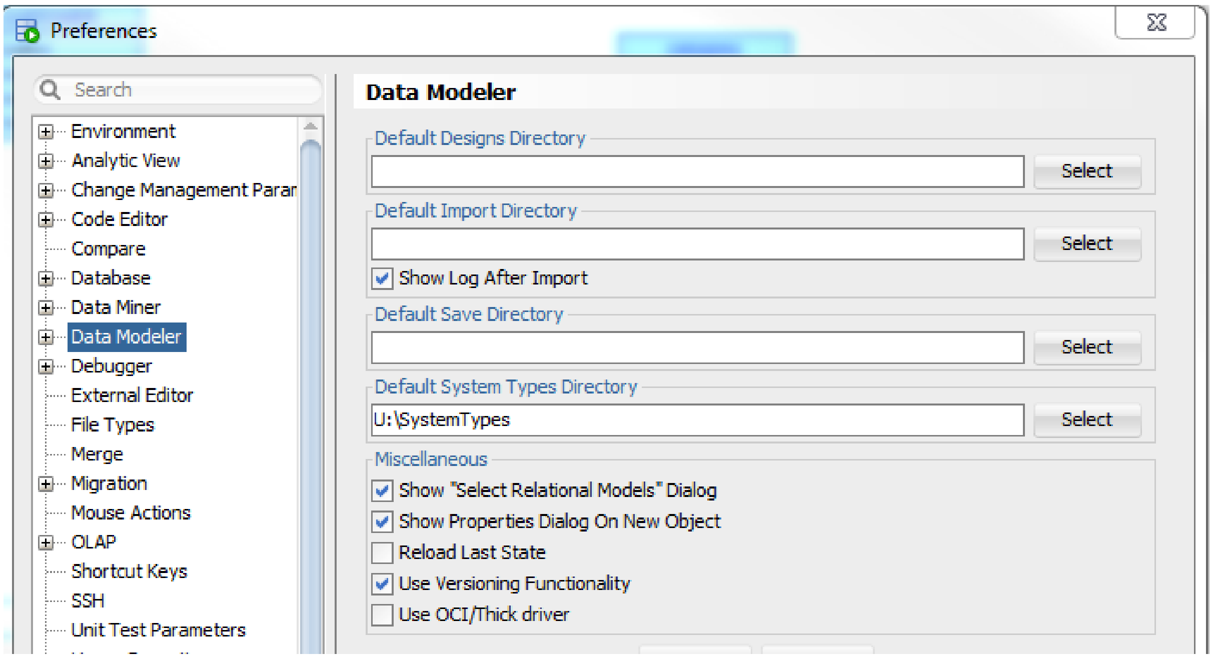
FK Attribute synchronization
- [✓] Name – Keep as the name of the Originating attribute
- [✓] Comments, Notes – Automatically propagate from PK attribute

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 labs we will leave the remaining settings at the default values.

If you are working on a **Monash on-campus PC** please set up a folder to save the Modeler System Files:

- In file explorer, connect to your mapped drive (U:) and create a folder called SystemTypes (note that this folder name MUST NOT contain spaces)
- In the Data Modeler preferences point the "Default System Types Directory" to this newly created folder



Preferences

Search

- Environment
- Analytic View
- Change Management Parar
- Code Editor
- Compare
- Database
- Data Miner
- Data Modeler
- Debugger
- External Editor
- File Types
- Merge
- Migration
- Mouse Actions
- OLAP
- Shortcut Keys
- SSH
- Unit Test Parameters

Data Modeler

Default Designs Directory: [ ] Select

Default Import Directory: [ ] Select

[✓] Show Log After Import

Default Save Directory: [ ] Select

Default System Types Directory: U:\SystemTypes Select

Miscellaneous

[✓] Show "Select Relational Models" Dialog

[✓] Show Properties Dialog On New Object

[ ] Reload Last State

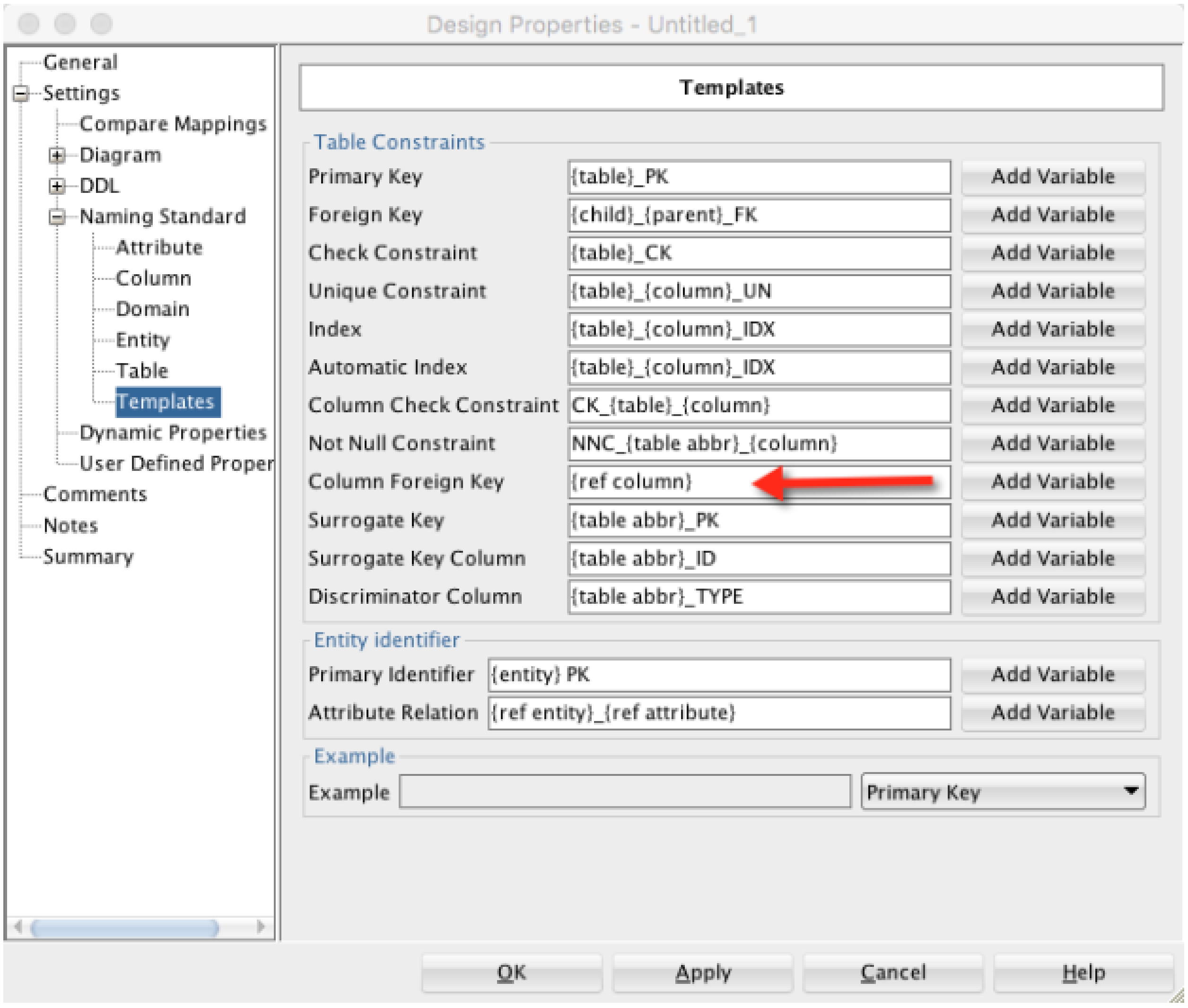
[✓] Use Versioning Functionality

[ ] Use OCI/Thick driver

This is necessary since Data Modeler needs to write the logical data types you define when it saves a project. If the default System Types Directory is a location Data Modeler cannot write to, e.g. c:\program files, an error message will be generated when you save your model.

Develop a model - Stage 1 The Logical Model:

Before starting a new project, 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.

For your first model we will implement a Customer-Orders system, represented by the following entities, where customers place orders for products:

CUSTOMER - customer number, name, address, phone number

ORDER - order number, order date, customer number and for each product ordered the quantity ordered and the total line price

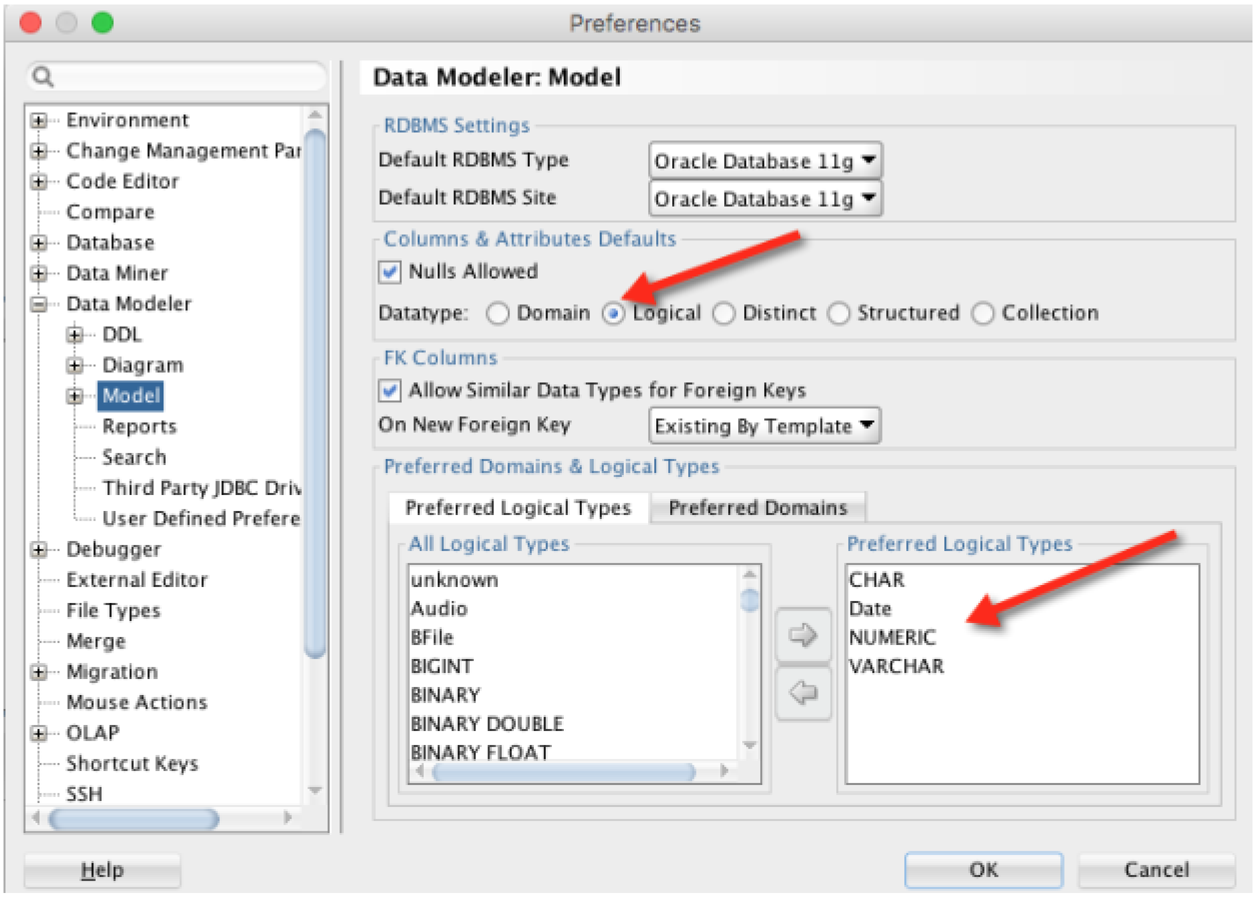
PRODUCT - product number, product description and product unit price

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 which 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 which 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 and types and a preferred range of types. In your SQL Developer preferences set:



Preferences

Environment
Change Management Par
Code Editor
Compare
Database
Data Miner
Data Modeler
DDL
Diagram
Model
Reports
Search
Third Party JDBC Driv
User Defined Prefere
Debugger
External Editor
File Types
Merge
Migration
Mouse Actions
OLAP
Shortcut Keys
SSH

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

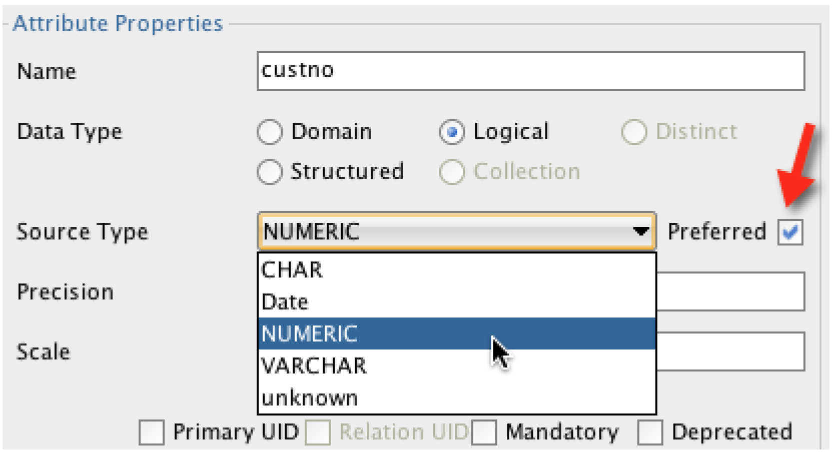
All Logical Types: unknown, Audio, BFile, BIGINT, BINARY, BINARY DOUBLE, BINARY FLOAT

Preferred Logical Types: CHAR, Date, NUMERIC, VARCHAR

Help OK Cancel

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.

Then when adding an attribute:



Attribute Properties

Name: custno

Data Type: Domain [✓ Logical] Distinct Structured Collection

Source Type: NUMERIC Preferred ✓

Precision

Scale

CHAR
Date
NUMERIC
VARCHAR
unknown

Primary UID Relation UID Mandatory Deprecated

check the 'Preferred' tick box and you will only see those types you have selected as preferred (clearly you may modify the set preferred data types to suit your needs).

On your logical model add an entity named CUSTOMER (name the entity under the General features in the Entity Properties dialog box) and then select the attributes feature. Add the following attributes:

- custno - Logical type: Numeric Precision 7, Scale 0, Primary UID
- custname - Logical type: VARCHAR(50), Mandatory
- custaddress - Logical type: VARCHAR(50), Mandatory
- custphone - Logical type: CHAR(10)

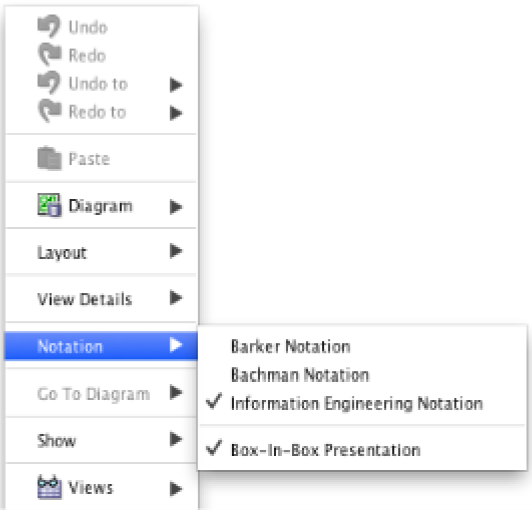
For each attribute add a meaningful description of the attribute under the attribute option - "Comments in the RDBMS". Then add an ORDERS entity with attributes:

- orderno - Logical type: Numeric Precision 7, Scale 0, Primary UID
- orderdate - 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.

For all the logical models we create you should set (right click on the logical model) the

- Notation to "Information Engineering Notation",

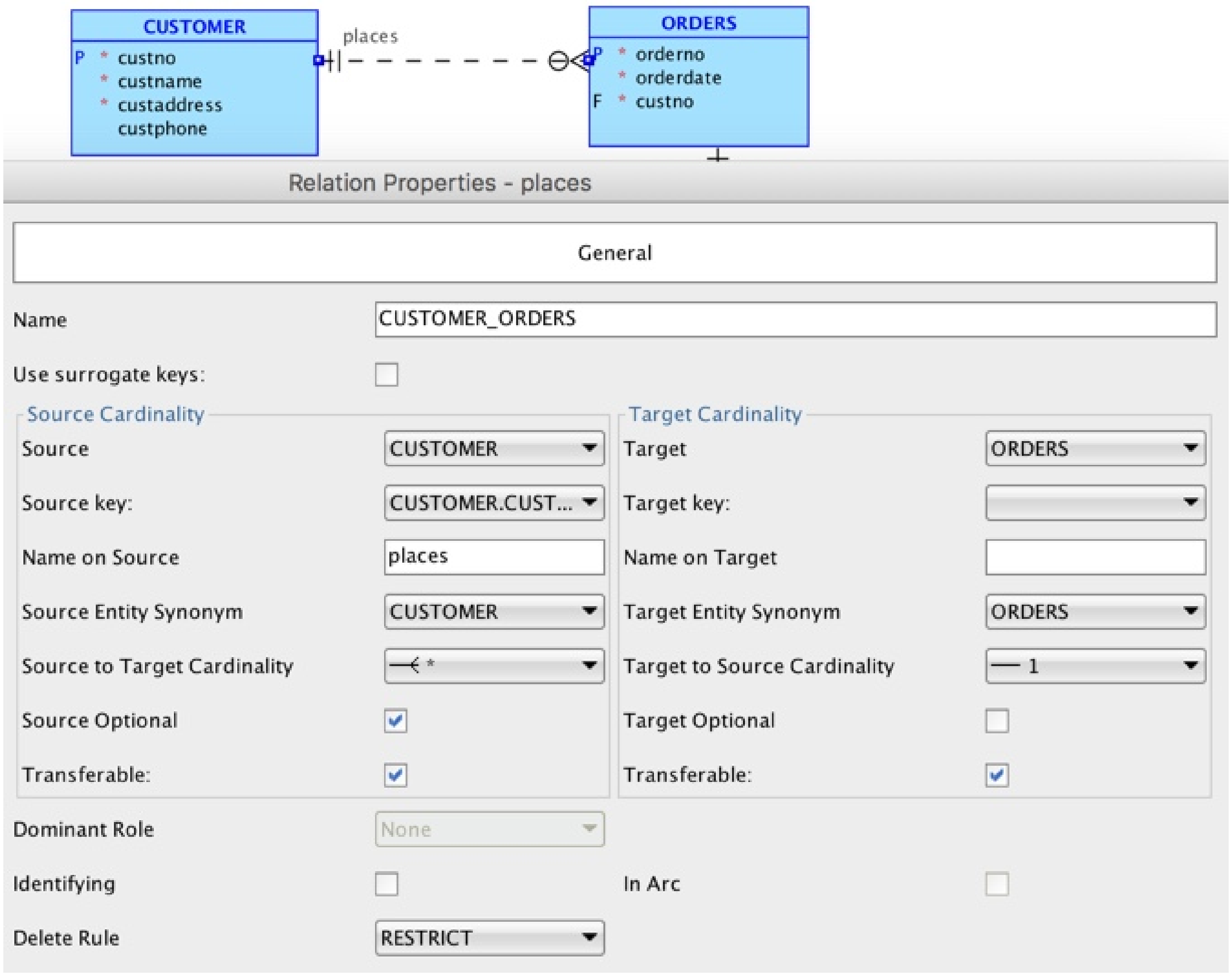


- View Details to "Attributes", and
- Show to "Labels" and "Legend"

When modelling students are required to include the "Legend" on all models (the panel may be moved to fit your model layout). **Models submitted without a legend will not be graded**.

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 and target, and set up the participation (Customer - optional, Order - mandatory ie. not optional):



Relation Properties - places

General

Name	CUSTOMER_ORDERS
Use surrogate keys:	☐

Source Cardinality

Source	CUSTOMER
Source key:	CUSTOMER.CUST...
Name on Source	places
Source Entity Synonym	CUSTOMER
Source to Target Cardinality	—< *
Source Optional	✓
Transferable:	✓

Target Cardinality

Target	ORDERS
Target key:	
Name on Target	
Target Entity Synonym	ORDERS
Target to Source Cardinality	— 1
Target Optional	☐
Transferable:	✓

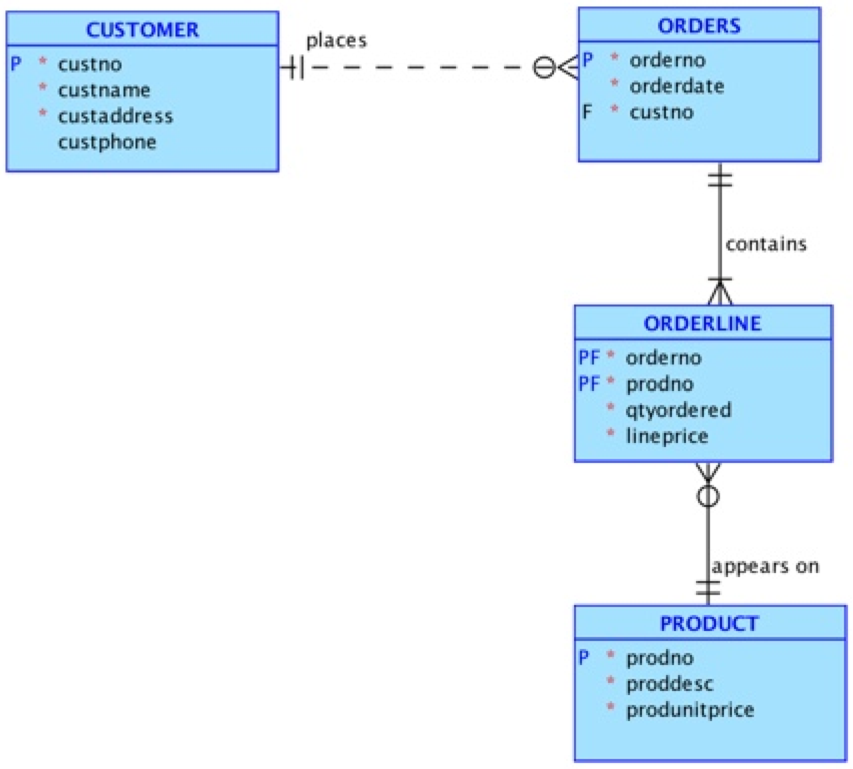
Dominant Role	None		
Identifying	☐	In Arc	☐
Delete Rule	RESTRICT		

Proceed and add the PRODUCT entity.

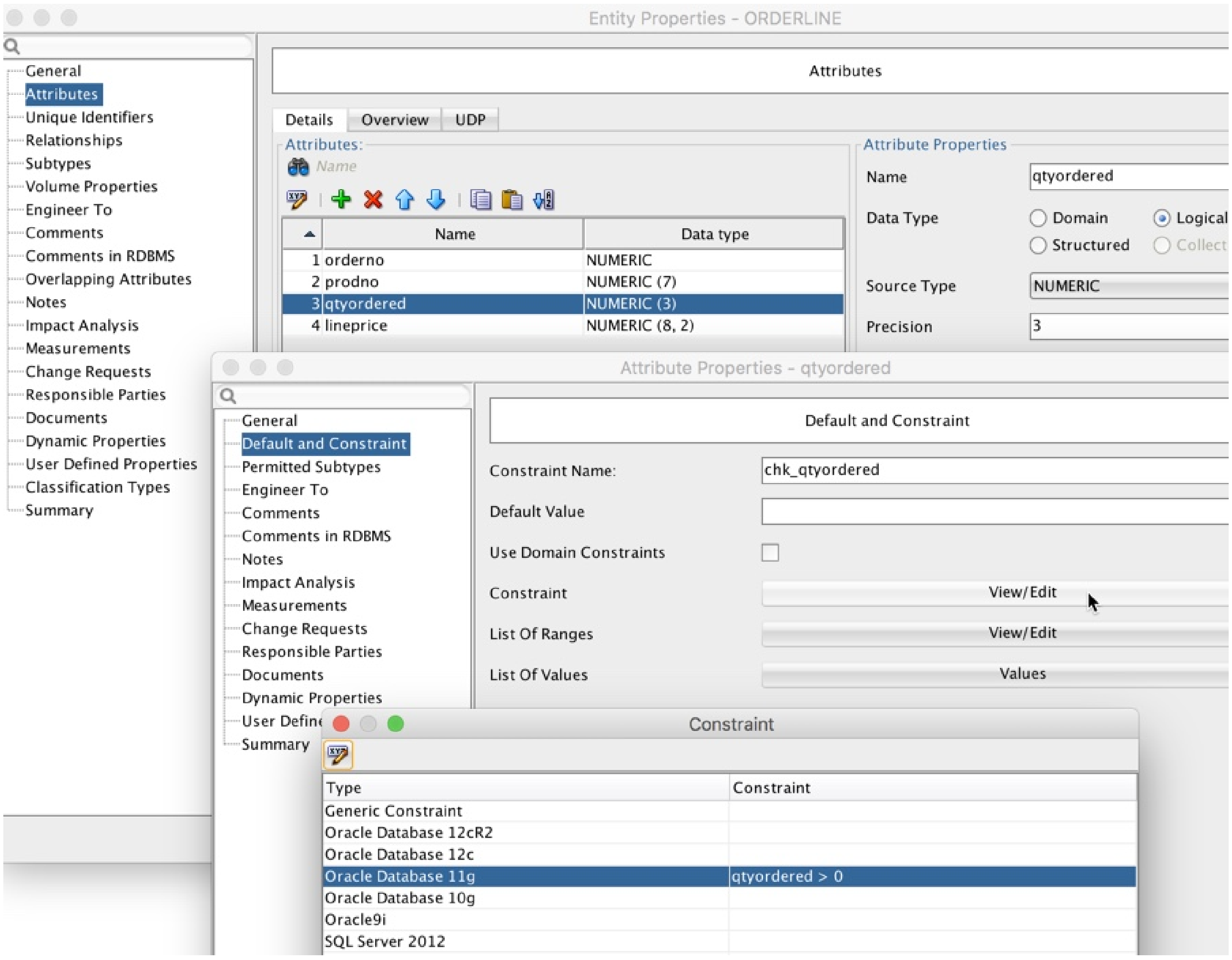
Modeler can draw M:N relations(hips) on its logical models, however, you cannot add attributes to such relationships - ***do not use the M:N relation(ship) in our models***.

Add a new entity ORDERLINE and then connect this with ORDERS and PRODUCT via 1:N Identifying relations(hips).

Your final logical model will have the form:



Constraints can be added to the logical model, as an example lets add a constraint to say that qtyordered in ORDERLINE must be greater than zero. Select the ORDERLINE entity, then select attributes and double click on qtyordered. In the left hand list of the pop-up window select "Default and Constraint". Give the constraint a name for example "chk_qtyordered" (be careful to select an informative name eg. chk_columnname), click Constraint "View/Edit", select Oracle Database 11g, click on the edit icon in the top left corner and then enter the constraint into the constraint editor:



	Name	Data type
1	orderno	NUMERIC
2	prodno	NUMERIC (7)
3	qtyordered	NUMERIC (3)
4	lineprice	NUMERIC (8, 2)

Name	qtyordered
Data Type	( ) Domain [✓ Logical] ( ) Structured ( ) Collect
Source Type	NUMERIC
Precision	3

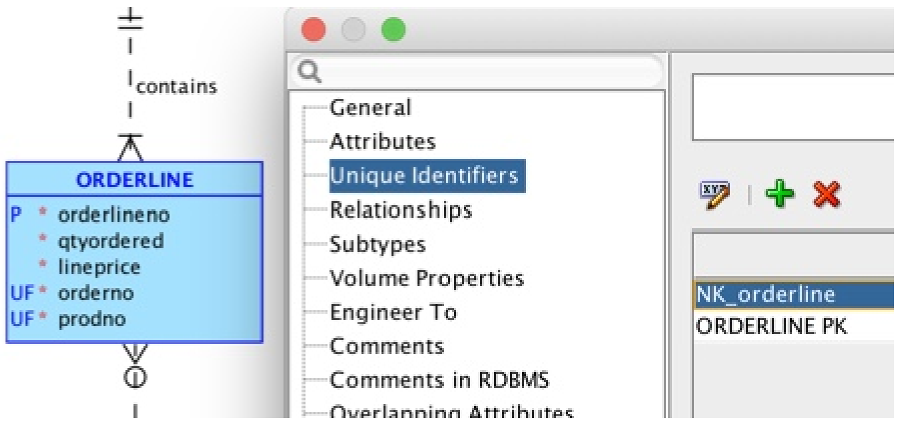
Constraint Name:	chk_qtyordered
Default Value	
Use Domain Constraints	☐
Constraint	View/Edit
List Of Ranges	View/Edit
List Of Values	Values

Type	Constraint
Generic Constraint	
Oracle Database 12cR2	
Oracle Database 12c	
Oracle Database 11g	qtyordered > 0
Oracle Database 10g	
Oracle9i	
SQL Server 2012	

Note that here, the constraint added is the inner part of the standard SQL CHECK clause.

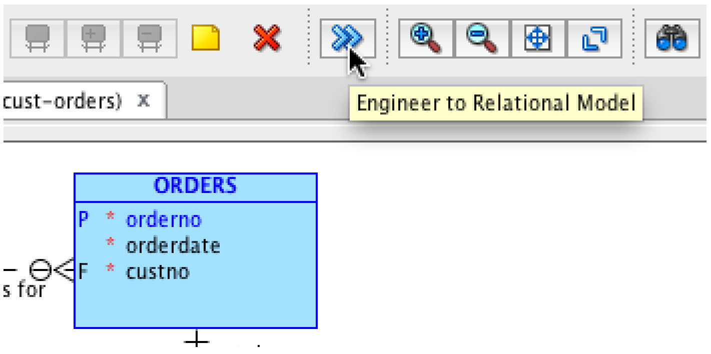
A better approach where a specific range, or set, of values is needed it to select "List Of Ranges" or "List Of Values" and directly enter the required values into the dialog boxes which appear.

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 which includes the attributes which compose the original key. As an example, say you decided to identify order lines by a unique order line number (*this would not be a good/common choice*, it is being used here for demonstration purposes only). On the ODERLINE entity, you need to select Unique Identifiers and add a new identifier (click the green add icon) with the attributes in your previously identified Natural Key (here orderno and prodno), this new unique indentifier should be named appropriately :

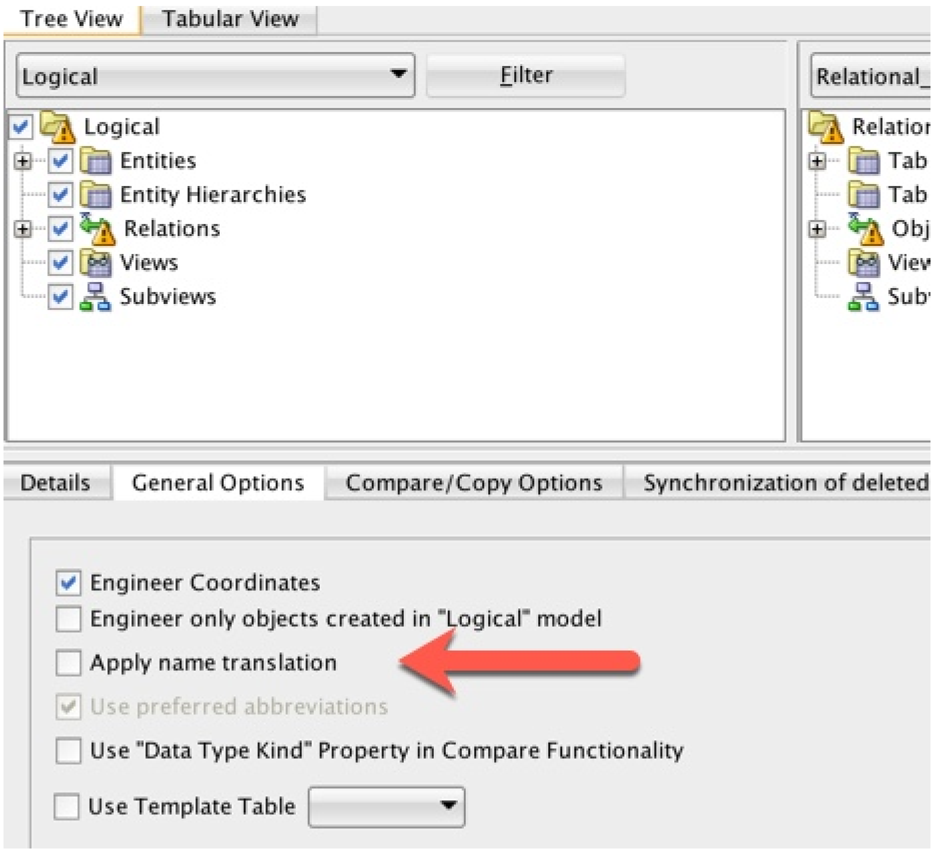


Develop a model - Stage 2 The Relational (Physical) Model:

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



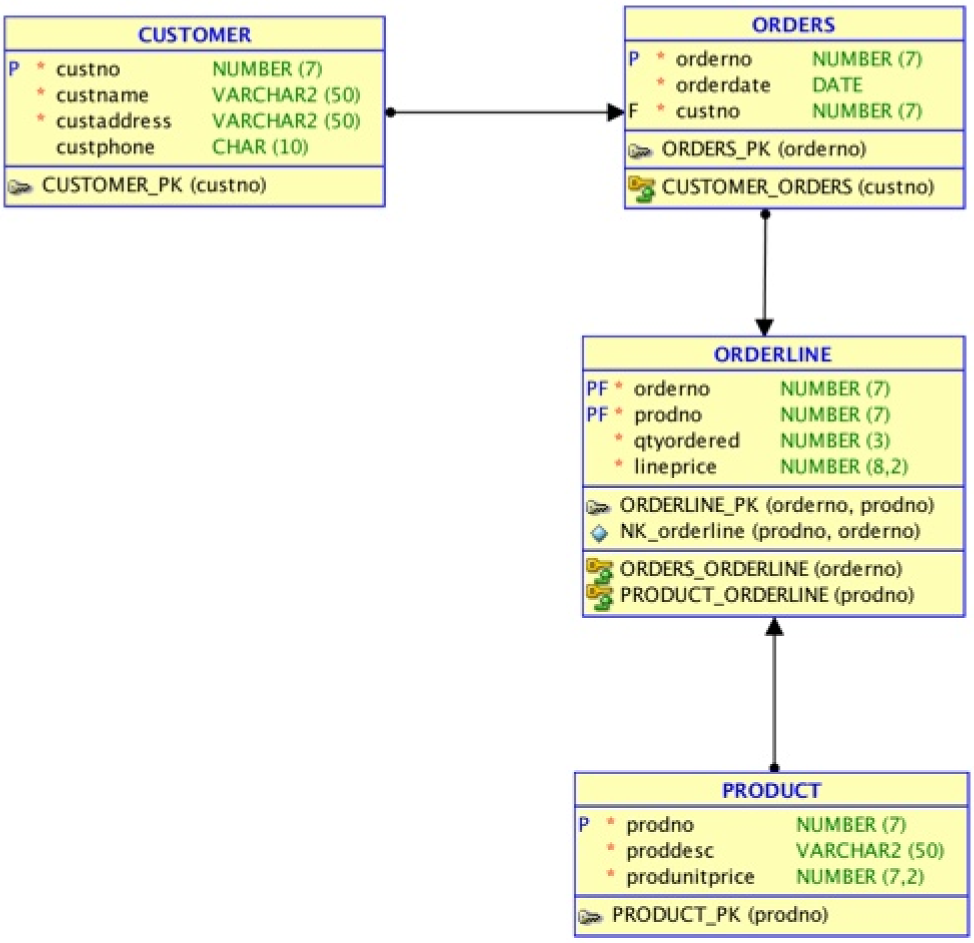
In the pop-up window which appears, on the "General Options" tab ensure "Apply name translation" is ***not*** checked:



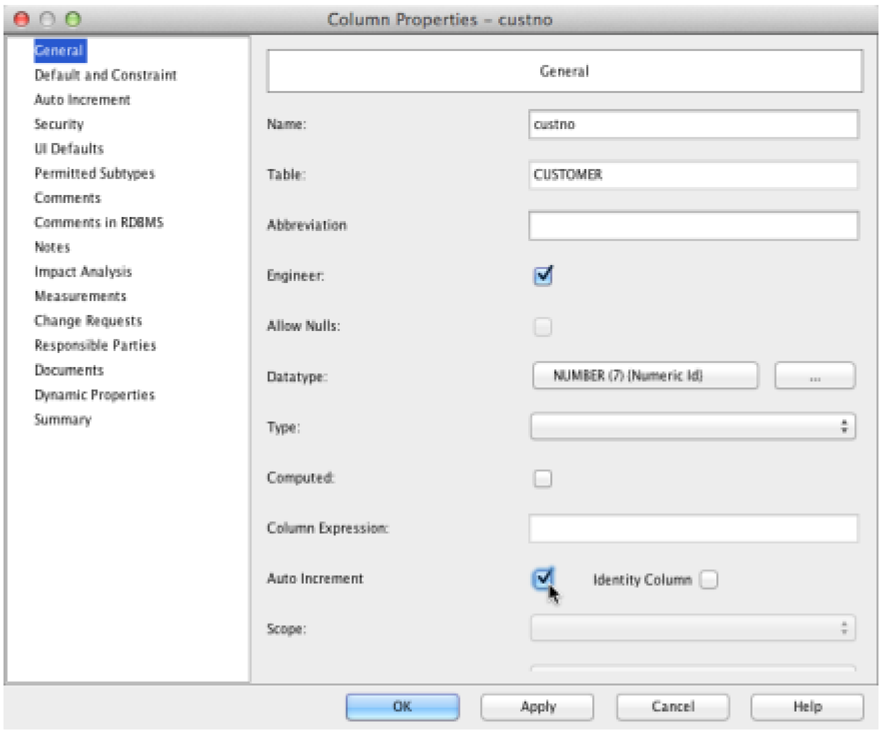
✓ Engineer Coordinates
☐ Engineer only objects created in "Logical" model
☐ Apply name translation
✓ Use preferred abbreviations
☐ Use "Data Type Kind" Property in Compare Functionality
☐ Use Template Table

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 it's 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:



As an example of what can be configured in the Relational model, select the CUSTOMER table, then the columns, then double click on the custno column, in the dialog which opens select "Auto Increment"


Column Properties - custno

General

Name: custno

Table: CUSTOMER

Abbreviation

Engineer: [✓]

Allow Nulls: [ ]

Datatype: NUMBER (7) (Numeric Id) ...

Type:

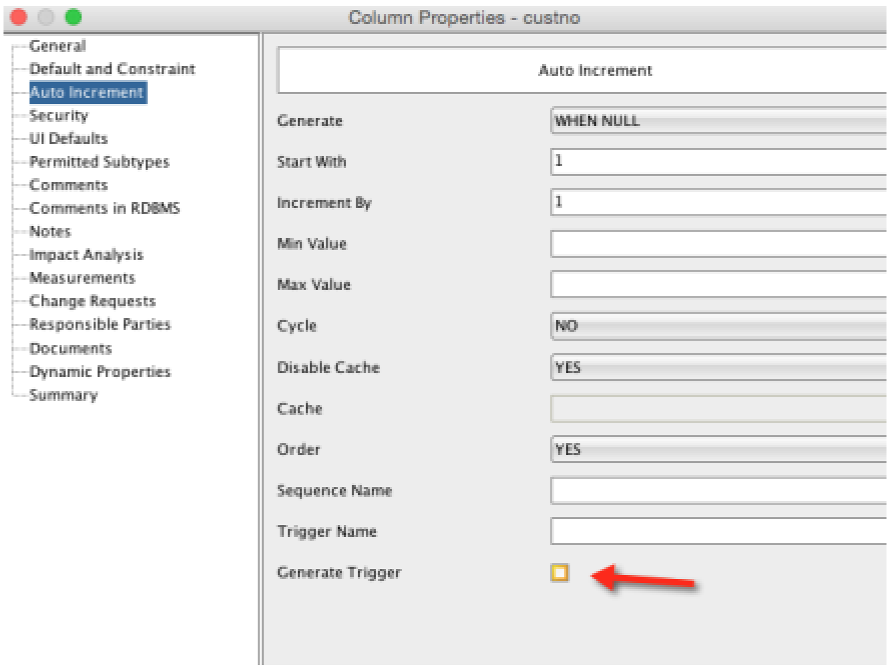
Computed: [ ]

Column Expression:

Auto Increment [✓] Identity Column [ ]

Scope:

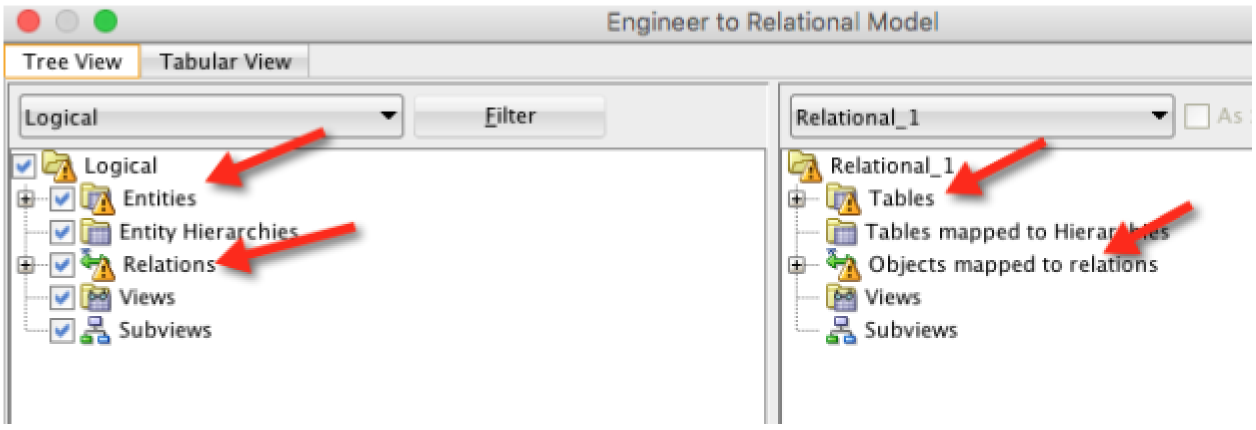
This will result in a sequence being created when the DDL is generated. The default configuration also generates a trigger to support the autoincrement which we do not need. To prevent this trigger being generated, after you have selected "Auto Increment" on General (above), select the Auto Increment option on the left and uncheck "Generate Trigger".



Generate	WHEN NULL
Start With	1
Increment By	1
Min Value	
Max Value	
Cycle	NO
Disable Cache	YES
Cache	
Order	YES
Sequence Name	
Trigger Name	
Generate Trigger	☐

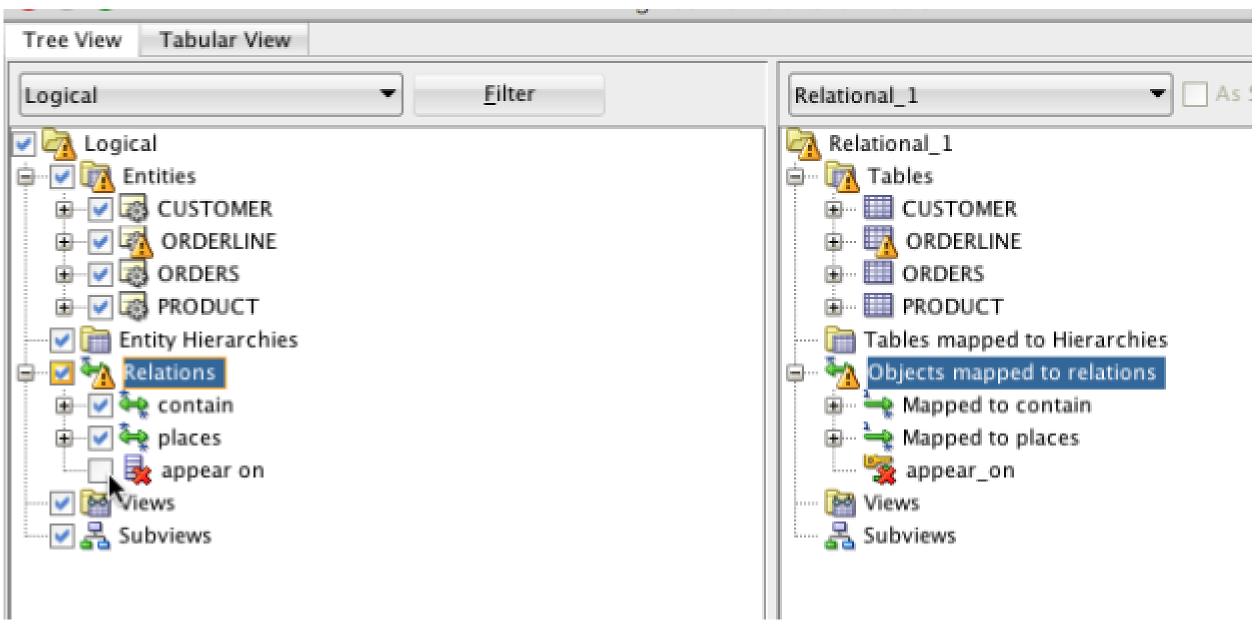
Although shown starting at 1 above, we often start auto-increment from say 100 to allow a developer to add test data under the sequence without making having to start use of the sequence and this "use up" some values.

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.

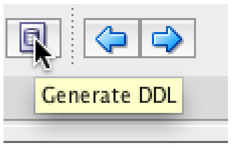
For example here under relations (where we are removing the "appear on" relation as a demonstration of what occurs):



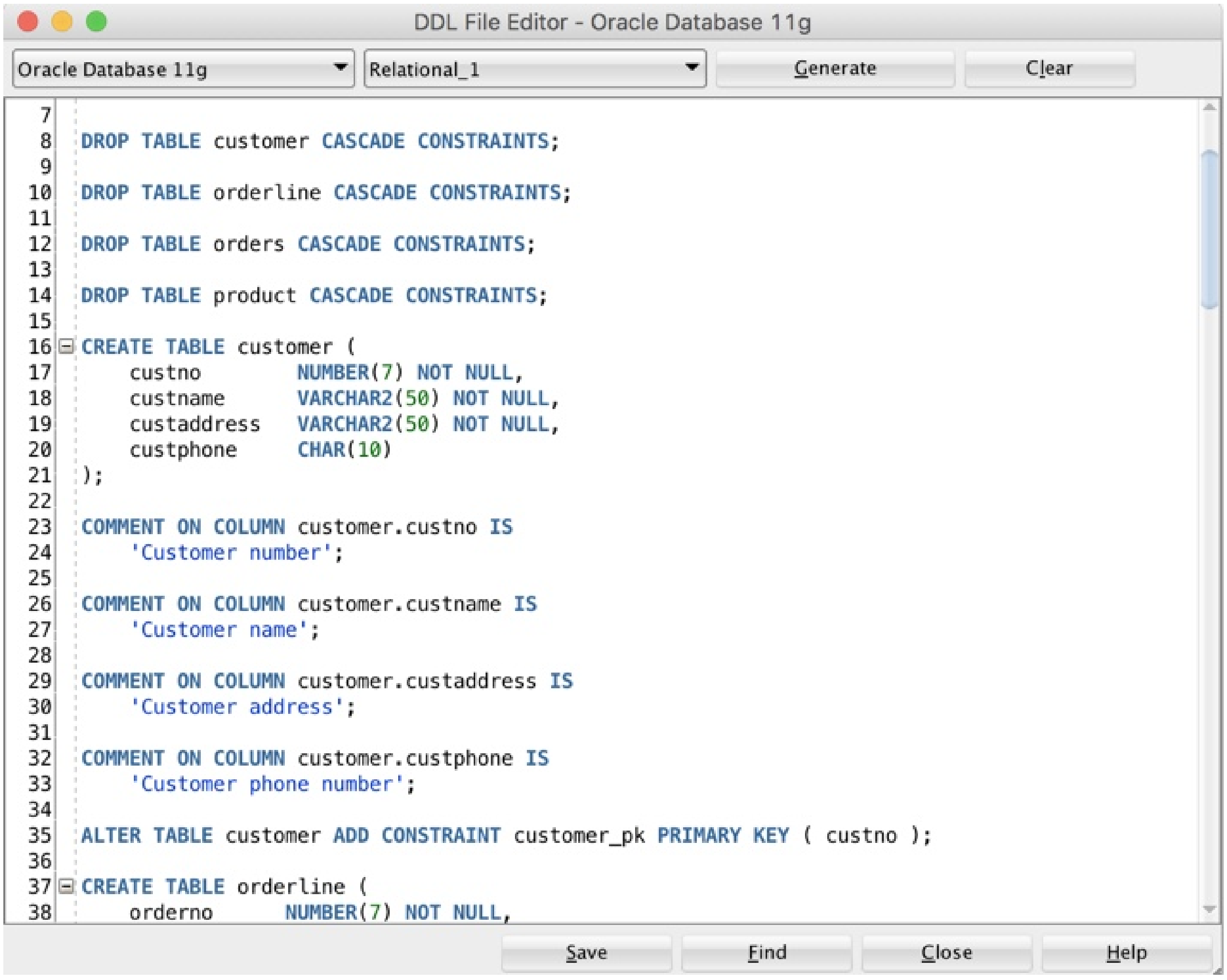
the removal of "appear on" has not been selected to be engineered to the relational model. You need to check the box if you wish it to be engineered (which we normally would).

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 TABLE customer CASCADE CONSTRAINTS;

DROP TABLE orderline CASCADE CONSTRAINTS;

DROP TABLE orders CASCADE CONSTRAINTS;

DROP TABLE product CASCADE CONSTRAINTS;

CREATE TABLE customer (
    custno        NUMBER(7) NOT NULL,
    custname      VARCHAR2(50) NOT NULL,
    custaddress   VARCHAR2(50) NOT NULL,
    custphone     CHAR(10)
);

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

COMMENT ON COLUMN customer.custname IS
    'Customer name';

COMMENT ON COLUMN customer.custaddress IS
    'Customer address';

COMMENT ON COLUMN customer.custphone IS
    'Customer phone number';

ALTER TABLE customer ADD CONSTRAINT customer_pk PRIMARY KEY ( custno );

CREATE TABLE orderline (
    orderno      NUMBER(7) NOT NULL,
```

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. Test your generated file against Oracle and confirm it operates correctly.

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).

5.1 Logical Modelling - Task B - Rental Model

Task B: Using SQL Developer Data Modeler

Using your model from last week for the "Property Rental System", map your Conceptual model (ERD) into a logical model in Oracle SQL Developer Data Modeler.

Engineer your Logical Model to a Relational Model and then create the tables etc in Oracle from the generated DDL. In doing so make use of at least one check clause and one sequence.

- 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 building 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. In addition the lease period and bond are recorded for each property or sub property rented. An owner may own several properties.
- Properties are subject to damage and the agency records all instance of damage to its properties - property, date, type of damage and repair cost are recorded. Repair costs are charged directly to tenants
- Normal property maintenance is also noted - property, date, type of maintenance and cost are recorded. Maintenance costs are charged to the property owner.
- 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.

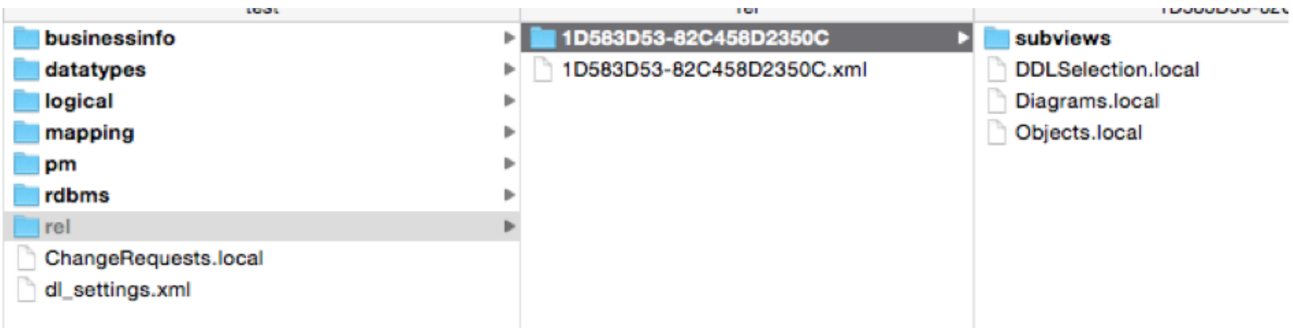
5.2 SQL Developer Data Modeller Issues

This document lists some of the issues you may experience when using SQL Developer and current work around's

Relational Model Disappears

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 re 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 linked XML file (https://goo.gl/Y5Zkzo) 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

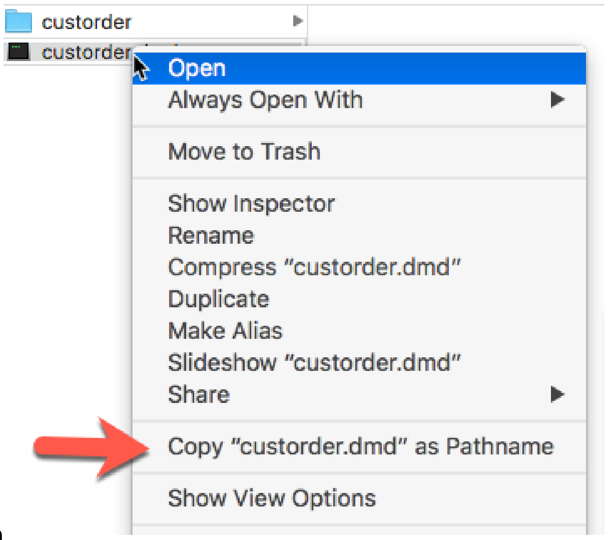
When you now open your model the relational model should be back.

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.

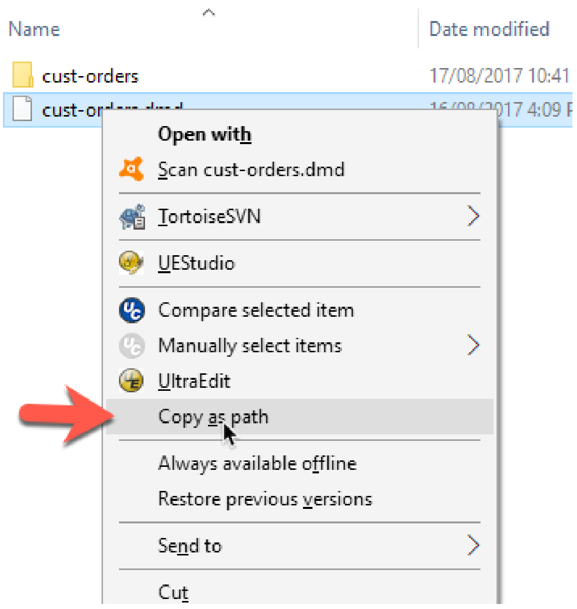
If this occurs, or as a standard alternative, you can type/obtain in the full path name that you wish to open or navigate to. For example, if you have a Customer-Orders project you wish to open use the following steps:

In your file manager navigate to where your .dmd file for the project is and obtain it's full path name. Under Mac OS, right click the dmd file and right click and while holding this, press the "option" key, this



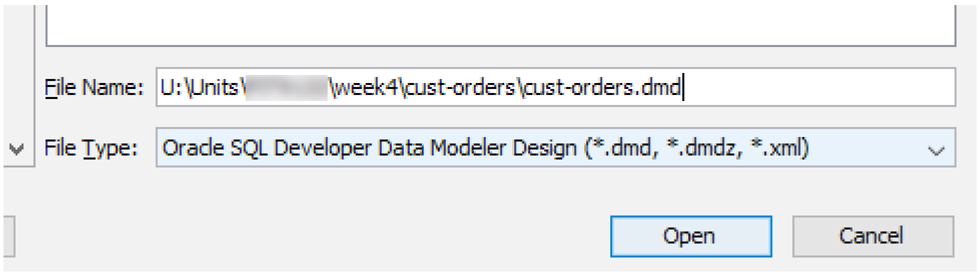
will result in

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 into the SQL Developer dialog box - to cause it to go to a particular folder or open a particular project. For example, in MS Windows:

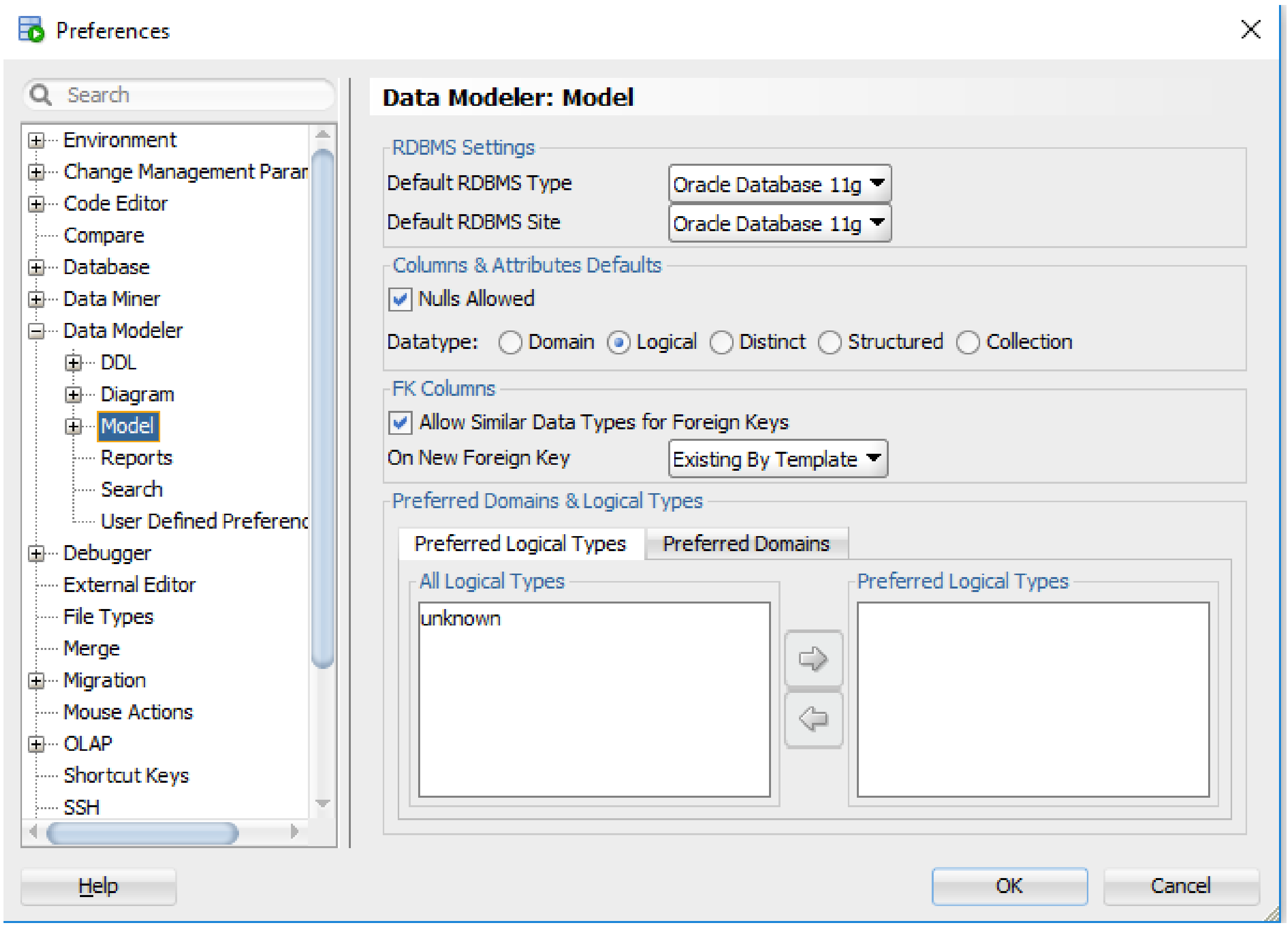

File Name: U:\Units\[redacted]\week4\cust-orders\cust-orders.dmd

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 path name.

Logical Datatypes 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

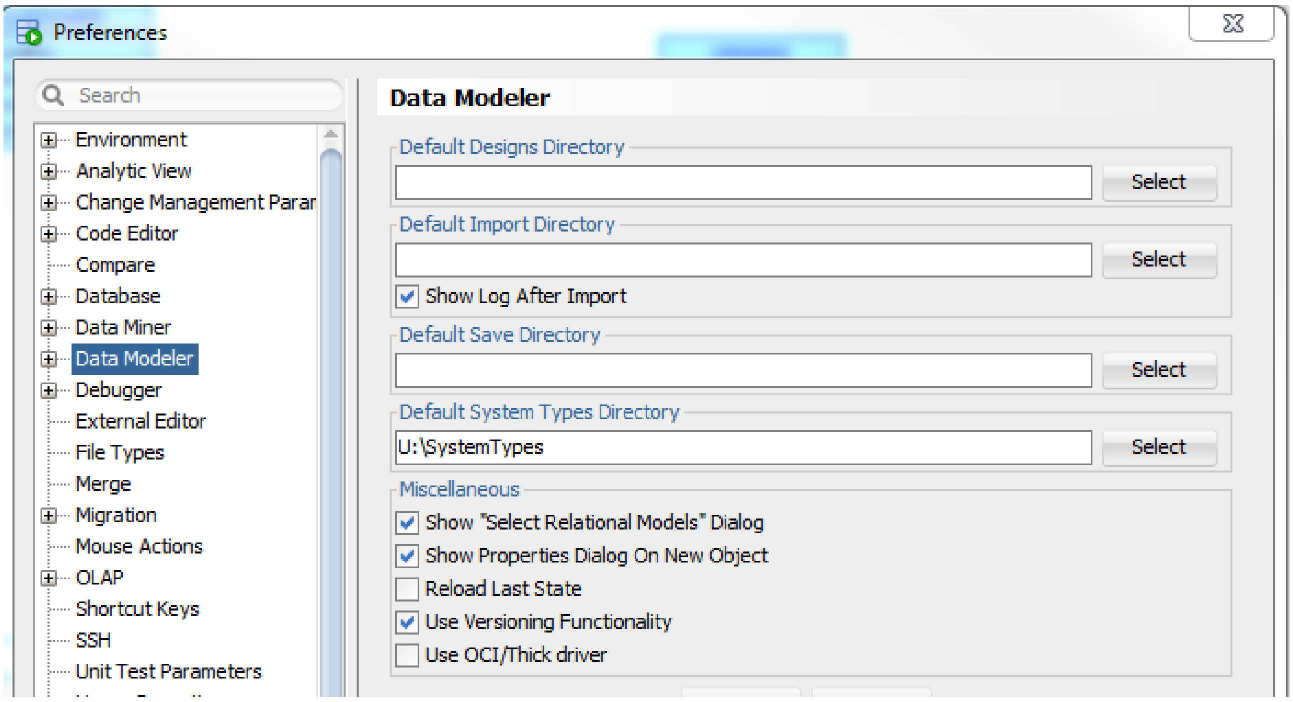
Preferred Domains & Logical Types

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.



Data Modeler

Default Designs Directory:

Default Import Directory:

[✓] Show Log After Import

Default Save Directory:

Default System Types Directory: U:\SystemTypes

Miscellaneous

[✓] Show "Select Relational Models" Dialog

[✓] Show Properties Dialog On New Object

[ ] Reload Last State

[✓] Use Versioning Functionality

[ ] Use OCI/Thick driver

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 pathnamee, or if SQL Developer has not correctly saved them at some stage.

To fix this:

- Go back to this setting and remove the current value in this entry ie. in the above, as used in the on-campus labs, it is U:\SystemTypes - make this entry blank (have no path)
- Save your settings and exit SQL Developer
- Reopen SQL Developer and the types will be back.

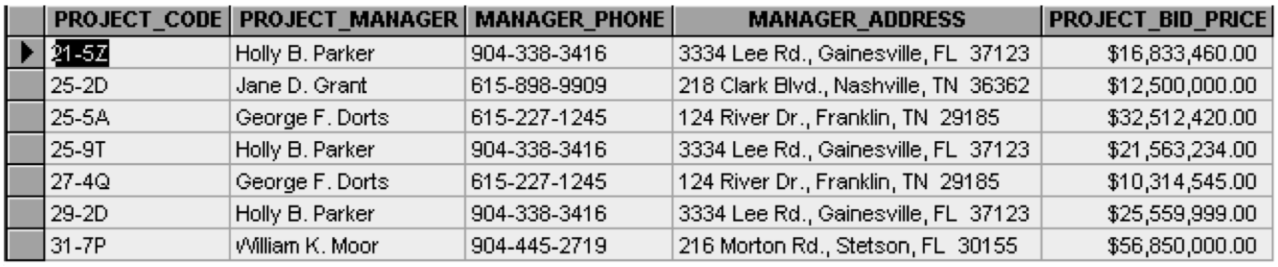
If you then have problems saving a model, you will need to reset up the Default System Types Directory as you previously did (ensure there is no space in the pathname).

6 Normalisation

Normalisation Tutorial Activities

Introduction

Normalisation process starts with identifying attributes and the possible existence of the repeating group. The starting set of attributes is called UNF (un-normalised form). During analysis, it is possible to have different sets of un-normalised form (UNF). This initial form will depend on what you perceive being represented in the table or relation. For example, consider the following table. You may consider the table is about PROJECT, others may see the table is data about PROJECT MANAGER. Both are correct perceptions but it will lead to a different UNF in the normalisation process.



PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
21-5Z	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	$16,833,460.00
25-2D	Jane D. Grant	615-898-9909	218 Clark Blvd., Nashville, TN 36362	$12,500,000.00
25-5A	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	$32,512,420.00
25-9T	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	$21,563,234.00
27-4Q	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	$10,314,545.00
29-2D	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	$25,559,999.00
31-7P	William K. Moor	904-445-2719	216 Morton Rd., Stetson, FL 30155	$56,850,000.00

A. UNF when perceiving table to represent PROJECT

PROJECT(project_code, project_manager, manager_phone, manager_address, project_bid_price)

B. UNF when perceiving table to represent PROJECT MANAGER

MANAGER(project_manager, manager_phone, manager_address (project_code, project_bid_price))

When you consider the table represents PROJECT MANAGER, you see that we have a repeating group because a PROJECT MANAGER can supervise more than one PROJECT. Notice the curly brackets enclosing project_code and project_bid_price attributes.

The PROJECT table/relation, on the other hand, does not have a repeating group because a project has only one project manager.

To complete the normalisation, you choose one of the UNFs as a starting point and follow the normalisation step to at least 3NF. The example steps to complete the normalisation provided below.

Normalisation process:

Legend: Primary Key is underlined, Foreign Key is in italic.

The identification of Foreign Key is not part of the formal steps of normalisation, however, it will give you a mechanism to check whether you have split the attributes of a relation into two separate relations correctly. When you split a relation into two, there should be at a common attribute (or attributes) exists in both relations. In one relation, it will be a Primary Key (PK). In the other relation, the attribute will be a Foreign Key (FK).

Approach A.

UNF

PROJECT(project_code, project_manager, manager_phone, manager_address, project_bid_price)

1NF

PROJECT(<u>project_code</u>, project_manager, manager_phone, manager_address, project_bid_price)

2NF

There is no partial dependency, the 2NF is the same as the 1NF.

PROJECT(<u>project_code</u>, project_manager, manager_phone, manager_address, project_bid_price)

3NF

PROJECT(<u>project_code</u>, project_bid_price, *project_manager*)

MANAGER(<u>project_manager</u>**,** manager_phone, manager_address)

Approach B

UNF

MANAGER(project_manager, manager_phone, manager_address (project_code, project_bid_price))

1 NF

MANAGER(<u>project_manager</u>, manager_phone, manager_address)

PROJECT(<u>project_code</u>, project_bid_price, *project_manager*)

2 NF

There is no partial dependency, the 2 NF is the same as the 1NF.

MANAGER(project_manager, manager_phone, manager_address)

PROJECT(project_code, project_bid_price, *project_manager*)

3 NF

There is no transitive dependency, the 3NF is the same as the 2NF.

MANAGER(project_manager, manager_phone, manager_address)

PROJECT(project_code, project_bid_price, *project_manager*)

Tutorial Questions.

A. This exercise is adapted from Connolly and Begg, Exercise 14.15 (p441)

staffNo	dentistName	patNo	patName	appointment		surgeryNo
				date	time	
S1011	Tony Smith	P100	Gillian White	12-Sep-08	10:00	S15
S1011	Tony Smith	P105	Jill Bell	12-Sep-08	12:00	S15
S1024	Helen Pearson	P108	Ian MacKay	12-Sep-08	10:00	S10
S1024	Helen Pearson	P108	Ian MacKay	14-Sep-08	14:00	S10
S1032	Robin Plevin	P105	Jill Bell	14-Sep-08	16:30	S15
S1032	Robin Plevin	P110	John Walker	15-Sep-08	18:00	S13

1. Describe possible insertion, deletion and update anomalies for the DENTIST relation.
2. Write the UNF (un-normalised Form) for each of the possible analysis below for the data depicted on the above table.
    1. The table contains data about APPOINTMENTs. An appointment is made for a dentist and a patient at an agreed appointment time.
    2. The table contains data about DENTISTs. A dentist has a number of appointments.
    3. The table contains data about DENTISTs. A dentist sees a number of patients. For each patient it may involve several appointments.
3. Perform the normalisation process up to the 3NF for the UNF you have created based on statement 1. After you produce the 1NF (the first normal form), draw the dependency diagram of the relation(s).
4. Perform the normalisation process up to the 3NF for the UNF you have created based on statement 2. After you produce the 1NF (the first normal form), draw the dependency diagram of the relation(s).
5. Interested in a challenge - perform the normalisation process up to the 3NF for the UNF you have created based on statement 3.

B. Normalise the forms to the 3NF.

UNITS CURRENTLY APPROVED **REPORT DATE:** 9/07/2015

Unit Number	Unit Name	Unit Description	Unit Value
FIT9131	Programming Foundations	Introduction to programming	6
FIT9132	Introduction to Databases	Database Fundamentals	6
FIT9134	Computer Architecture and Operating Systems	Fundamentals of computer systems and the computing environment	6
FIT9135	Data Communications	Fundamentals of data and computer communications	6

* Unit values may be either 3, 6 or 12 points

LECTURER DETAILS **REPORT DATE:** 29/07/2015
LECTURER'S NUMBER: 10234
LECTURER'S NAME: GUISEPPE BLOGGS
LECTURER'S OFFICE No.: 169
LECTURER'S PHONE No.: 99037111
UNIT ADVISER FOR:

UNIT NUMBER	UNIT NAME
FIT9131	Programming Foundations
FIT9134	Computer Architecture and Operating Systems

* A given unit may have several advisers

* Some lecturers share offices, although each have their own phone

STUDENT DETAILS **REPORT DATE:** 29/07/2015
STUDENT No.: 12345678
STUDENT NAME: Poindexter Jones
STUDENT ADDRESS: 23 Wide Road, Caulfield, 3162
COURSE ENROLLED: MIT
MODE OF STUDY: On-Campus
MENTOR NUMBER: 10234
MENTOR NAME: Guiseppe Bloggs
ACADEMIC RECORD:

UNIT NUMBER	UNIT NAME	YEAR / SEMESTER	GRADE
FIT9131	Programming Foundations	2014/2	N
FIT9131	Programming Foundations	2015/1	D
FIT9132	Introduction to Databases	2015/1	D

* Grade may have the value N, P, C, D or HD

* Mode of Study must be On-campus (O) or Distance Education (D)

In order to add a student, the lecturer who advises that student must already exist in the database. No lecturer may be deleted who advises any students which are currently in the database. If the lecturer number of a lecturer is changed, then the number would be changed for each student advised by that lecturer.

Note:

- To simplify the normalisation process that involves multiple forms, you should perform the normalisation one form at a time until all relations are in 3NF. Once you have done this process for all forms, consolidate the relations from the different forms by:
    - group together all relations with the same primary key, i.e representing the same entity.
    - choose a single name for synonyms. For example, **mentor** is the same as **lecturer.**

7 SQL Data Definition Language (DDL)

When creating schema files, you should always also create a drop file or add the drop commands to the top of your schema file. You should drop the tables using the:

drop table tablename purge;

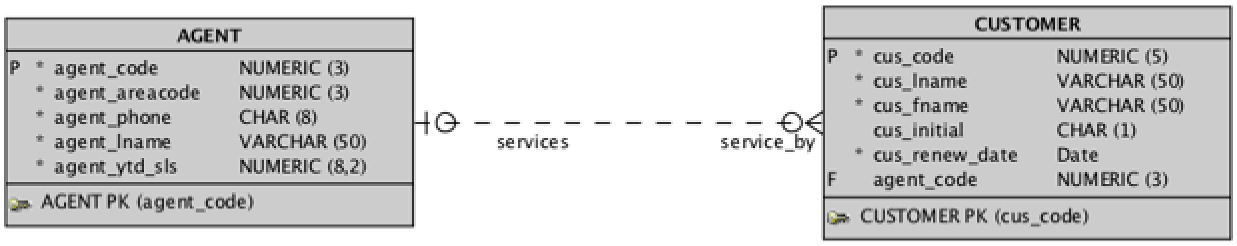
syntax.

The drop table statements should list tables in the reverse order of your create table order so that FK relationships will be able to be removed successfully. Should a syntax error occur while testing your schema, you simply need to run the drop commands to remove any tables which may have been created.

An excellent summary of the Oracle data types and version restrictions is available from:

https://www.techonthenet.com/oracle/datatypes.php

For this unit, we make use of CHAR, VARCHAR2 (or VARCHAR), NUMBER (or NUMERIC) and DATE



The data model above represents figure 3.3 from Coronel & Morris. There are two different ways of coding this model as a set of create table statements.

Using table constraints

SQL constraints are classified as column or table constraints; depending on which item they are attached to:

```
create table agent
  (
    agent_code     number (3) constraint agent_pk primary key,
    agent_areacode number (3) not null ,
    agent_phone    char (8) not null ,
    agent_lname    varchar2 (50) not null ,
    agent_ytd_sls  number (8,2) not null
  ) ;
```

This is a declaration of the primary key as a column constraint

```
create table agent
  (
    agent_code     number (3) not null ,
    agent_areacode number (3) not null ,
    agent_phone    char (8) not null ,
    agent_lname    varchar2 (50) not null ,
    agent_ytd_sls  number (8,2) not null,
    constraint agent_pk primary key ( agent_code )
  ) ;
```

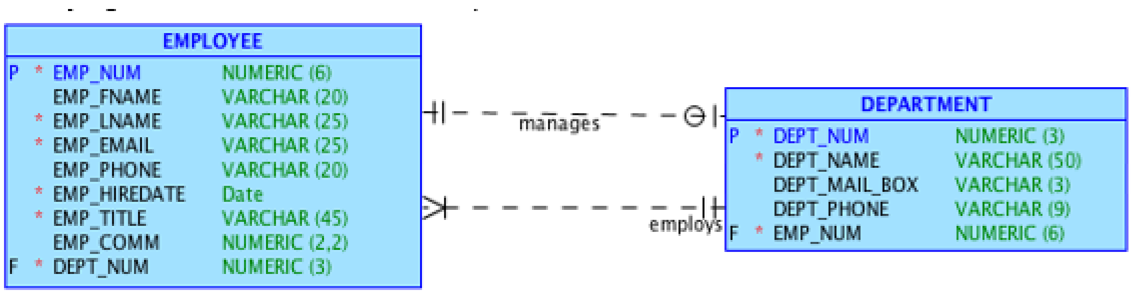
Here the primary key has been declared as a table constraint, at the end of the table after all column declarations have been completed. In some circumstances, for example, a composite primary key you must use a table constraint since a column constraint refers only to a single column.

The create table statements for the two tables in fig 3-3 would be:

```
create table agent
  (
    agent_code     number (3) not null ,
    agent_areacode number (3) not null ,
    agent_phone    char (8) not null ,
    agent_lname    varchar2 (50) not null ,
    agent_ytd_sls  number (8,2) not null,
    constraint agent_pk primary key ( agent_code )
  ) ;


create table customer
  (
    cus_code       number (5) not null ,
    cus_lname      varchar2 (50) not null ,
    cus_fname      varchar2 (50) not null ,
    cus_initial    char (1) ,
    cus_renew_date date not null ,
    agent_code     number (3),
    constraint customer_pk primary key ( cus_code ),
    constraint customer_agent_fk foreign key ( agent_code)
       references agent ( agent_code ) on delete set null
  ) ;
```

The inclusion of the referential integrity rule *on delete set null* in the above create table statement is appropriate in this scenario - when an agent leaves a reasonable approach would be to set the foreign key for that agent's customers to null. The default on delete restrict (which you do not specify, simply omit an on delete clause) would also be an alternative approach. Using *on delete cascade* would not since this would cause the customers of the agent who left to also be deleted. When coding a foreign key definition you **must always consider what is a suitable on delete approach (RESTRICT, CASCADE, NULLIFY) given the scenario you are working with**.

In some circumstances, this approach of defining the foreign keys as part of the table definitions cannot be used. Can you see what the issue is with trying to create the two tables depicted below?



In such a situation an alternative approach to declaring constraints needs to be adopted.

Using ALTER table commands

In this approach, the tables are declared without constraints and then the constraints are applied via the ALTER TABLE command (see section 7.5 of Coronel & Morris).

```
create table agent
  (
    agent_code     number (3) not null ,
    agent_areacode number (3) not null ,
    agent_phone    char (8) not null ,
    agent_lname    varchar2 (50) not null ,
    agent_ytd_sls  number (8,2) not null
  ) ;

alter table agent add constraint agent_pk primary key
   ( agent_code ) ;

create table customer
  (
    cus_code       number (5) not null ,
    cus_lname      varchar2 (50) not null ,
    cus_fname      varchar2 (50) not null ,
    cus_initial    char (1) ,
    cus_renew_date date not null ,
    agent_code     number (3)
  ) ;

alter table customer add constraint customer_pk primary key
    ( cus_code ) ;

alter table customer add constraint customer_agent_fk foreign key
   ( agent_code ) references agent ( agent_code )
   on delete set null;
```

Remember, from above, when coding a foreign key definition you **must always consider what is a suitable on delete approach (RESTRICT, CASCADE, NULLIFY) given the scenario you are working with**.

After creating the tables we need to insert the data, for AGENT the insert will have the form:

```
insert into agent values (501,713,'228-1249','Alby',132735.75);
```

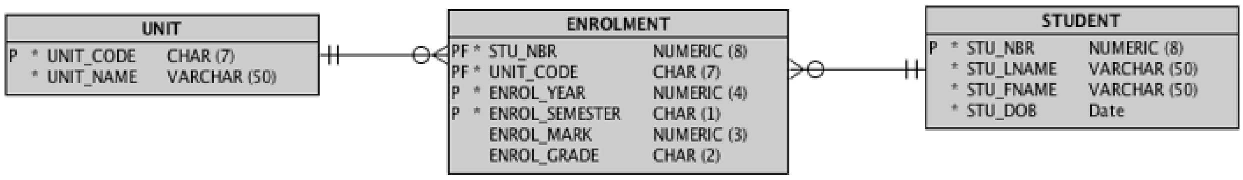
for customer:

```
insert into customer values(10010,'Ramas','Alfred','A',
    '05-Apr-2014',501);
```

It is important to note that for the insert into customer we are using the default Oracle date format of dd-mon-yyy - in the near future we will correct this and allow any date format via the oracle function *todate*.

Lab Tasks

Using the model from the DDL lecture for student, unit and enrolment:



Creating tables from scratch.

- Code a schema file to create these three tables, noting the following ***extra*** constraints:
    1. stu_nbr > 10000000
    2. unit_name is unique in the UNIT table
    3. enrol_semester can only contain the value of 1 or 2 or 3.
- In implementing these constraints you will need to make use of CHECK clauses (see Coronel & Morris section 7.2.6).
- Ensure your script file has appropriate comments in the header, includes the required drop commands and includes echo on and echo off commands.
- Run your script and create the three required tables.
- Save the output from this run.

As an alternative to using echo on/off and having to save the output, a simpler approach is through the use of the inbuilt Oracle SPOOL command.

To use SPOOL, place as the top line in your schema file:

```
spool ./myoutput.txt
```

and as the last line in your script file

```
spool off
```

This will produce a file, in the same folder that your script is saved in, called myoutput.txt which contains the full run of your SQL script.

7.1 INSERTing data into the database

1. Basic INSERT statement.

In this exercise, you will enter the data into the database using INSERT statements with the following assumptions:

- the database currently does not have any existing data.
- the primary key is not generated automatically by the DBMS.

TASKS

Insert the following data into the tables specified using the SQL INSERT statement:

STUDENT

stu_nbr	stu_lname	stu_fname	stu_dob
11111111	Bloggs	Fred	01-Jan-1990
11111112	Nice	Nick	10-Oct-1994
11111113	Wheat	Wendy	05-May-1990
11111114	Sheen	Cindy	25-Dec-1996

UNIT

unit_code	unit_name
FIT9999	FIT Last Unit
FIT5132	Introduction to Databases
FIT5016	Project
FIT5111	Student's Life

ENROLMENT

stu_nbr	unit_code	enrol_year	enrol_semester	enrol_mark	enrol_grade
11111111	FIT5132	2013	1	35	N
11111111	FIT5016	2013	1	61	C
11111111	FIT5132	2013	2	42	N
11111111	FIT5111	2013	2	76	D
11111111	FIT5132	2014	2		
11111112	FIT5132	2013	2	83	HD
11111112	FIT5111	2013	2	79	D
11111113	FIT5132	2014	2		
11111113	FIT5111	2014	2		
11111114	FIT5111	2014	2		

- Ensure you make use of COMMIT to make your changes permanent.
- Check that your data has inserted correctly by using the SQL command SELECT * FROM *tablename* **and** by using the SQL GUI (select the table in the right-hand list and then select the Data tab).

2. Using SEQUENCEs in an INSERT statement.

In the previous exercises, you have entered the primary key value manually in the INSERT statements. In the case where a SEQUENCE is available, you should use the sequence mechanism to generate the value of the primary key.

TASKS

Create a sequence for the STUDENT table called STUDENT_SEQ

- Create a sequence for the STUDENT table called STUDENT_SEQ that starts at 11111115 and increases by 1.
- Check that the sequence exists in two ways (using SQL and browsing your SQL Developer connection objects).

Add a new student (MICKEY MOUSE)

- Use the student sequence - pick any STU_DOB you wish.
- Check that your insert worked.
- Add an enrolment for this student to the unit FIT5132 in semester 2 2016.

3. Advanced INSERT.

We have learned how to add data into the database in the previous exercises through the use of INSERT statements. In those exercises, the INSERT statements were created as a single script assuming that data is all added at the same time, such as at the beginning when the tables are created. On some occasions, new data is added after some data already exists in the database. In this situation, it is a good idea to use a combination of INSERT and SELECT statements.

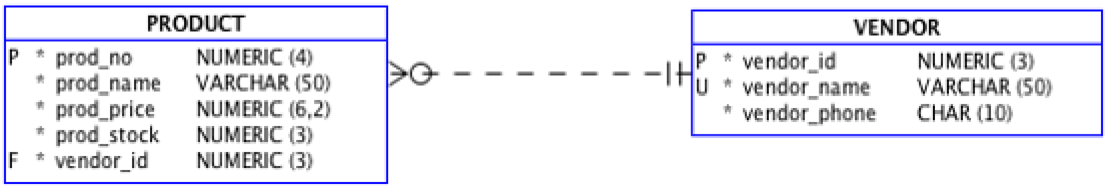
A SELECT statement is an SQL statement that we use to retrieve data from a database. An example of a SELECT statement would be:

```
SELECT vendor_id
FROM vendor
WHERE vendor_name = 'Seagate';
```

The above SQL statement consists of three SQL clauses SELECT, FROM and WHERE. The SELECT clause is used to declare which column(s) are to be displayed in the output. The FROM clause is used to declare from which table the data needs to be retrieved. The WHERE clause is used to declare which rows are to be retrieved. In the above SQL select, any row that has the vendor_name equal to 'Seagate' will be retrieved. The SQL SELECT statement will be covered in more detail in the future module, retrieving data from the database.

For our exercise on using the advanced INSERT statement, consider the following model depicting VENDOR and PRODUCT.

Assume we want to add vendors and the products they supply into a set of tables represented by:



A suitable schema would be:

```
DROP TABLE PRODUCT PURGE;
DROP TABLE VENDOR PURGE;
DROP SEQUENCE PRODUCT_prod_no_SEQ;
DROP SEQUENCE VENDOR_vendor_id_SEQ;

CREATE TABLE PRODUCT
  (
    prod_no          NUMBER (4) NOT NULL ,
    prod_name        VARCHAR2 (50) NOT NULL ,
    prod_price       NUMBER (6,2) NOT NULL ,
    prod_stock       NUMBER (3) NOT NULL ,
    VENDOR_vendor_id NUMBER (3) NOT NULL
  ) ;

ALTER TABLE PRODUCT ADD CONSTRAINT PRODUCT_PK PRIMARY KEY ( prod_no ) ;

CREATE TABLE VENDOR
  (
    vendor_id    NUMBER (3) NOT NULL ,
    vendor_name  VARCHAR2 (50) NOT NULL ,
    vendor_phone CHAR (10) NOT NULL
  ) ;

ALTER TABLE VENDOR ADD CONSTRAINT VENDOR_PK PRIMARY KEY ( vendor_id ) ;

ALTER TABLE VENDOR ADD CONSTRAINT VENDOR_UN UNIQUE ( vendor_name ) ;

ALTER TABLE PRODUCT ADD CONSTRAINT PRODUCT_VENDOR_FK FOREIGN KEY (
VENDOR_vendor_id ) REFERENCES VENDOR ( vendor_id ) ON
DELETE CASCADE ;

CREATE SEQUENCE PRODUCT_prod_no_SEQ START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE VENDOR_vendor_id_SEQ START WITH 1 INCREMENT BY 1;
```

There are two ways in which we can perform the INSERT.

1. Use the nextval and currval of the sequences.

```
-- Add Vendor 1 and the products they supply
 insert into vendor values (VENDOR_vendor_id_SEQ.nextval,
        'Western Digital', '1234567890');
 insert into product values (PRODUCT_prod_no_SEQ.nextval,
        '2TB My Cloud Drive',195,5,VENDOR_vendor_id_SEQ.currval);
 insert into product values (PRODUCT_prod_no_SEQ.nextval,
        '1TB Portable Hard Drive',76,4,VENDOR_vendor_id_SEQ.currval);
 insert into product values (PRODUCT_prod_no_SEQ.nextval,
        'Live Media Player',119,2,VENDOR_vendor_id_SEQ.currval);
 commit;
 -- Add Vendor 2 and the products they supply
 insert into vendor values (VENDOR_vendor_id_SEQ.nextval,'Seagate',
        '2468101234');
 insert into product values (PRODUCT_prod_no_SEQ.nextval,
        '2TB Desktop Drive',94,12,VENDOR_vendor_id_SEQ.currval);
 insert into product values (PRODUCT_prod_no_SEQ.nextval,
        '4TB 4 Bay NAS',76,4,VENDOR_vendor_id_SEQ.currval);
 insert into product values (PRODUCT_prod_no_SEQ.nextval,
        '2TB Central Personal Storage' ,169,5,
      VENDOR_vendor_id_SEQ.currval);
 commit;
```

2. Use the nextval in combination with the SELECT statement.

- Add a new product for a vendor at a subsequent time (vendor names will be unique - note the U in the model above and the vendor_un constraint in the schema)

```
insert into product values (PRODUCT_prod_no_SEQ.nextval,
        'GoFlex Thunderbolt Adaptor',134,2,
        (select vendor_id from vendor where vendor_name = 'Seagate'));
```

In subsequent weeks you will see that the same concept can be used with other data manipulation statements such as UPDATE and DELETE.

TASKS

- A new student has started a course and needs to enrol into "Introduction to databases". Enter the new student's details and his/her enrolment to the database using the nextval in combination with a SELECT statement. You can make up details of the new student and when they will attempt "Introduction to databases".
    - You must not do a manual lookup to find the unit code of the "Introduction to databases".

4. Creating a table and inserting data as a single SQL statement.

A table can also be created based on an existing table, and immediately populated with contents by using a SELECT statement within the CREATE TABLE statement.

For example, to create a table called FIT5132_STUDENT which contains the enrolment details of all students who have been or are currently enrolled in FIT5132, we would use:

```
create table FIT5132_STUDENT
as select *
from enrolment
where unit_code = 'FIT5132';
```

Here, we use the SELECT statement to retrieve all columns (the wildcard "*" represents all columns) from the table enrolment, but only those rows with a value of the unit_code equal to FIT5132.

TASKS

- Create a table called FIT5111_STUDENT. The table should contain all enrolments for the unit FIT51111.
- Check the table exists.
- List the contents of the table.

5. Changing a table's structure.

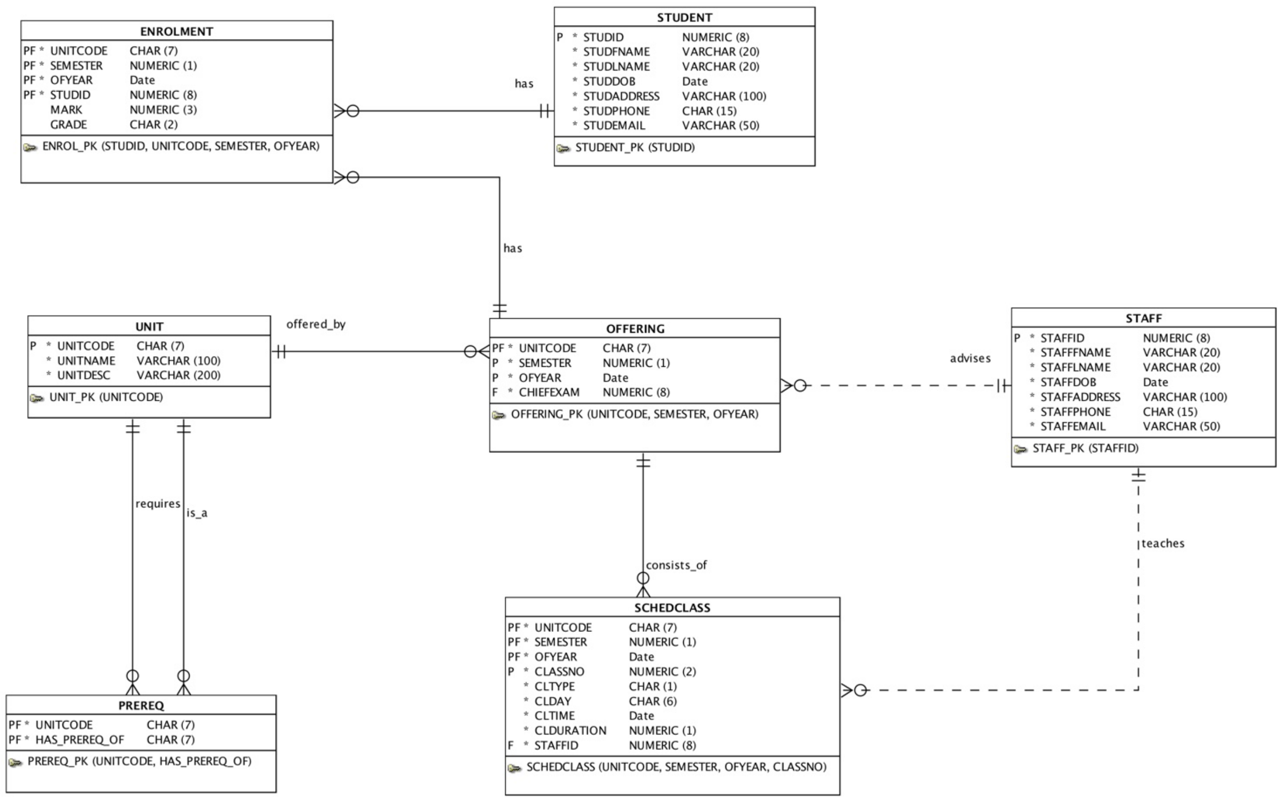
TASKS

- Add a new column to the UNIT table which will represent credit points for the unit (hint use the ALTER command). The default value for the new column should be 6 points.
- Insert a new unit after you have added the new column. You can make up the details of the new unit.
- Check that the new insert has worked correctly.

8 SQL Part I - SQL Basic

The following exercises will allow you to be familiar with writing basic SQL statements.

Use the UNIVERSITY database to complete the exercises. Figure 1.0 depicts the data model for the UNIVERSITY database.



University Data model

In the Monash Oracle database, this UNIVERSITY set of tables has been created under the user "UNI". To use these tables you need to add the prefix "UNI" to the table names that you use in an SQL statement. For example, if you want to retrieve data from UNIT table you need to write:

```
SELECT unitcode, unitname
FROM uni.unit;
```

instead of

```
SELECT unitcode, unitname
FROM unit;
```

This week we make use of Oracle dates - to use these correctly you should note the following:

- The Oracle date data type contains both date and time, however, you can choose to use just a date, just a time, both or parts of a date depending on the format strings used
- to_date: converts from a string to a date according to a format string
- to_char: converts from a date to a string according to a format string

The Oracle documentation links are:

- Format models (http://goo.gl/6IFTqP)
- to_date (http://goo.gl/hhrCAo)
- to_char (http://goo.gl/7yvmwp)

Part A. Retrieving data from a single table

1. List all students and their details.
2. List all units and their details.
3. List all students who have the surname 'Smith'.
4. List the student's details for those students who have surname starts with the letter "S". In the display, rename the columns studfname and studlname to firstname and lastname.
5. List the student's surname, firstname and address for those students who have surname starts with the letter "S" and firstname contains the letter "i".
6. List the unit code and semester of all units that are offered in the year 2014.
   To complete this question you need to use the Oracle function to_char to convert the data type for the year component of the offering date into text. For example, to_char(ofyear,'yyyy') - here we are only using the year part of the date.
7. List the unit code of all units that are offered in semester 1 of 2014.
8. Assuming that a unit code is created based on the following rules:
   - The first three letters represent faculty abbreviation, eg FIT for the Faculty of Information Technology.
   - The first digit of the number following the letter represents the year level.List the unit details of all first year units in the Faculty of Information Technology.
9. List the unit code and semester of all units that were offered in either semester 1 or summer of 2013. Note: summer semester is recorded as semester 3.
10. List the student number, mark, unit code and semester for those students who have passed any unit in semester 1 of 2013.

Part B. Retrieving data from multiple tables.

Note: remember to use the foreign key and the primary key when joining two or more tables.

1. List the name of all students who have marks in the range of 60 to 70.
2. List all the unit codes, semester and name of the chief examiner for all the units that are offered in 2014.
3. List the name (firstname and surname), unit names, the year and semester of enrolment of all units taken so far.
4. List all the unit codes and the unit names and their year and semester offerings. To display the date correctly in Oracle, you need to use to_char function. For example, to_char(ofyear,'YYYY').
5. List the unit code, semester, class type (lecture or tutorial), day and time for all units taught by Albus Dumbledore in 2013. Sort the list according to the unit code.
6. Create a study statement for Mary Smith. A study statement contains unit code, unit name, semester and year study was attempted, the mark and grade.
7. List the unit code, unit name and the unit code and unit name of the pre-requisite units of all units in the database.
8. List the unit code and unit name of the pre-requisite units of 'Advanced Data Management' unit.
9. Find all students (list their id, firstname and surname) who have a failed unit in the year 2013.
10. List the student name, unit code, semester and year for those students who do not have marks recorded.

9
SQL Part II - Maintaining data

Using DELETE and UPDATE

1. UPDATE

It is common for data to change value across time. In a database, we use the SQL UPDATE statement to change the value of a cell or cells in a table.

The UPDATE statement consist of three main components:

- The name of the table where the data will be updated.
- The row or the set of rows where the value will be updated.
- The new value to replace the old value.

An example of an UPDATE statement for data in the database we have created by following the exercises in the module Tutorial 6 SQL Data Definition Language DDL is as follow:

```
UPDATE enrolment
SET enrol_mark = 60
WHERE stud_nbr = 11111111 AND
unit_code = 'FIT5132' AND
enrol_semester = '2' AND
enrol_year = 2014;
```

TASKS

1. Update the unit name of FIT9999 from 'FIT Last Unit' to 'place holder unit'.
2. Enter the mark and grade for the student with the student number of 11111113 for the unit code FIT5132 that the student enrolled in semester 2 of 2014. The mark is 75 and the grade is D.
3. The university introduced a new grade classification. The new classification are:
    1. 45 - 54 is P1.
    2. 55 - 64 is P2.
    3. 65 - 74 is C.
    4. 75 - 84 is D.
    5. 85 - 100 is HD.

   Change the database to reflect the new grade classification.

2. DELETE

The DELETE statement is used to remove data from the database. It is important to consider the referential integrity issues when writing a DELETE statement. In Oracle, a table can be created with FOREIGN KEY constraints with a reference_clause ON DELETE action. The action can be set to CASCADE and SET NULL. When the reference_clause is not specified, the action will be set to RESTRICT, in other words, the deletion of row in a parent table (table contains the PRIMARY KEY being referred to by a FOREIGN KEY) will not be allowed when there are rows in the child table (the table with the FOREIGN KEY).

TASKS

1. A student with student number 11111114 has taken intermission in semester 2 2014, hence all the enrolment of this student for semester 2 2014 should be removed. Change the database to reflect this situation.
2. Assume that Wendy Wheat (student number 11111113) has withdrawn from the university. Remove her details from the database.
3. Add Wendy Wheat back to the database (use the INSERT statements you have created when completing module Tutorial 6 SQL Data Definition Language DDL). Change the FOREIGN KEY constraints definition for table STUDENT so it will now include the references_clause ON DELETE CASCADE. Hint: You need to use the ALTER TABLE statement to drop the FOREIGN KEY constraint first and then put it back using ALTER TABLE with ADD CONSTRAINT clause. A brief description of using ALTER to drop a constraint is available here (https://www.techonthenet.com/oracle/foreign_keys/drop.php), the ADD CONSTRAINT was covered in tutorial 6. For more details, you can check the SQL Reference Manual (available from Moodle) for the full syntax and a range of examples. Once you have changed the table, now, perform the deletion of the Wendy Wheat (student number 11111113) row in the STUDENT table. Examine the ENROLMENT table. What happens to the enrolment records of Wendy Wheat?

9.1 Transactions Management

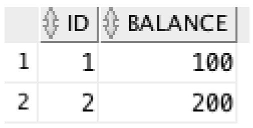
Transaction Management

In these exercises, you will examine the issues involved in updating shared data.

You will work in pairs. Suppose one user1 is User1, and the other is User2. *Replace User1 and User2 with your respective usernames when reading the tutorial exercises*.

User1 will create a table called account which will be shared with User2, that is, both users will be allowed to select data from the table and also update data in the table. This table keeps the account balances of customers, where each customer is identified by a unique id.

Q1. User1 should create the account table. The table will have 2 attributes, id and balance. Both attributes will have datatype number. After creating the table, User1 should insert two rows of data so that the table looks as below:



	ID	BALANCE
1	1	100
2	2	200

Q2. User1 can now make the account table available to User2 using the following command:

```
grant select, update on account to User2;
```

Q3. In order for User2 to access the account table, they would normally have to prefix the account table with the name of the owner, e.g:

```
select * from User1.account;
```

However, we can remove the need to do this if we create a synonym (essentially a system maintained alias for the table) as follows (User2 issues this command):

```
create synonym account for User1.account;
```

Q4. Make sure both users have the autocommit feature turned OFF - i.e. both users should issue the command

```
set autocommit off
```

Q5. Now, try the following (maintain the order of the operations).

- User1 updates the balance of customer 1 from 100 to 110 (without issuing a commit).
- User2 views the contents of the account table (do they see the new value? - if not, why not?)
- User1 issues a commit command.
- User2 views the contents of the account table (do you notice any difference?)

Explain what is happening in the results of the above queries, in the context of atomic transactions.

Q6. Now we will try and see what happens when we try some concurrent updates of the table (keep the order of transactions the same as below)

- User1 updates the balance of customer 2 from 200 to 150 (without issuing a commit).
- User2 tries to update the balance of customer 2 to 100 (what happens?)

Explain what is happening here. What should be done to allow the User2 update to proceed?

Q7. Now try the following:

- User1 updates the balance of customer 2 from 200 to 150 (without issuing a commit).
- User2 tries to update the balance of customer 1 to 125 (what happens?)

How does this differ from the results of the transactions in part 6? What does this tell you about the granularity of locking in Oracle? What must be done in order for the results of both updates to be visible to both users?

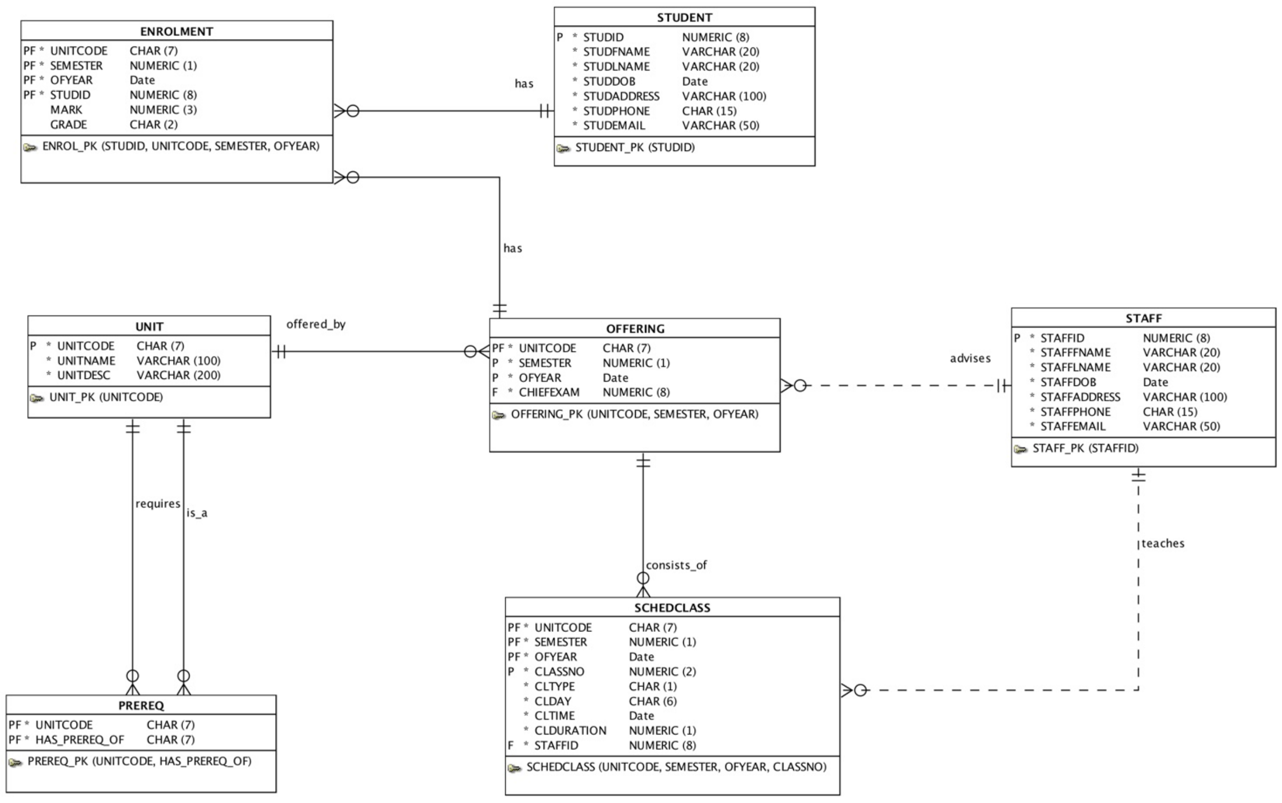
Q8. Try and generate a deadlock between the two users (hint: you will need to set up another shared table). Remember that a deadlock occurs when User1 holds a lock on table A and requests a lock on table B, but table B is locked by User2 who is also requesting a lock on table A.

10
SQL Part III - SQL Intermediate

The following exercises will allow you to be familiar with:

- All the SQL constructs used in last weeks exercises.
- Aggregate functions such as min, max, avg
- The GROUP BY clause, and
- Subqueries

This week we will continue to use the UNIVERSITY database model:



Use the Uni database from the previous exercises to complete the following queries.

1. Find the maximum mark for FIT1004 in semester 1, 2013.
2. Find the average mark of FIT1040 in semester 2, 2013.
3. List the average mark for each offering of FIT1040. In the listing, you need to include the year and semester number. Sort the result according to the year.
4. Find the number of students enrolled in the unit FIT1040 in the year 2013, under the following conditions:
    - Repeat students are counted each time
    - Repeat students are only counted once
5. Find the total number of enrolment per semester for each unit in the year 2013. The list should include the unitcode, semester and year. Order the list in increasing order of enrolment numbers.
6. Find the total number of prerequisite units for FIT2077.
7. Find the total number of prerequisite units for each unit. In the list, include the unitcode for which the count is applicable.

8. For each pre-requisite unit, calculate how many times it has been used as prerequisite. Include the name of the prerequisite unit in the listing .
9. Find the unit with the highest number of enrolments in a given offering in the year 2013.
10. Who is the oldest student in FIT1004? Display the student full name and the date of birth.
11. Find all students enrolled in FIT1004 in semester 1, 2013 who have scored more than the average mark of FIT1004 in the same offering? Display the students' name and the mark. Sort the list in the order of the mark from the highest to the lowest.

10.1
PL/SQL Triggers FIT3171

PL_SQL Triggers

A trigger may be created by typing the code for the trigger directly into the SQL Developer SQL window and finishing the code with a / on a line by itself in column 1, or alternatively via the SQL Developer Trigger item. Using the latter approach allows you to make use of the trigger debug environment in SQL Developer.

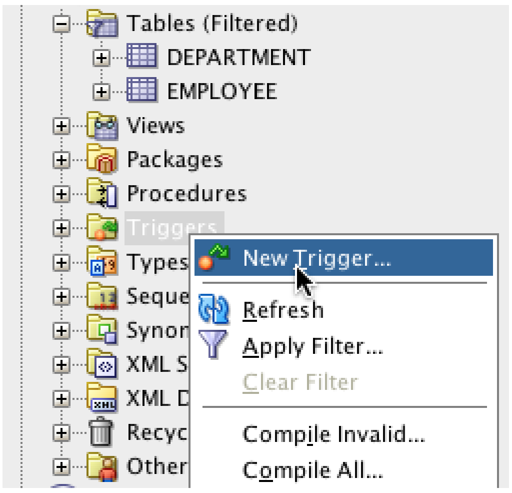
All work carried out for this module must be completed under SVN.

Task A: Employee-Department

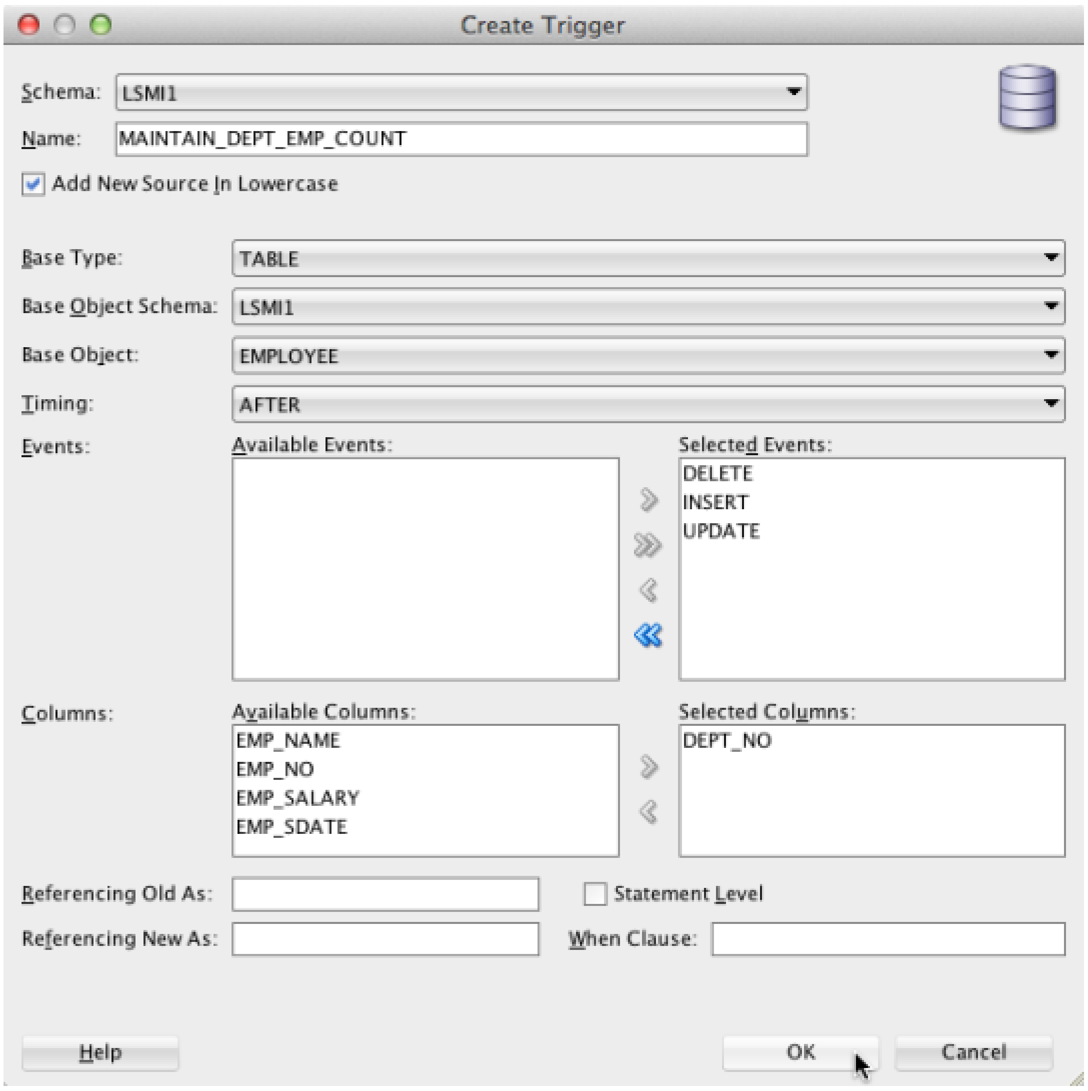
Given an EMPLOYEE table and a DEPARTMENT table with the structure: DEPARTMENT (dept_no, dept_name dept_empcnt), where dept_empcnt is the count of employees in department. We will create a trigger, which automatically maintains this value:

- insert an employee increases the department count for the department the employee is being added to
- delete an employee decreases the department count for the department the employee is being removed from
- update an employee's department (ie. transfer an employee from their current department to a new department) decreases the current department count and increases the new department count

a. Create a new trigger:



b. Select/Enter the new trigger details:



Schema: LSMI1

Name: MAINTAIN_DEPT_EMP_COUNT

[✓] Add New Source In Lowercase

Base Type: TABLE

Base Object Schema: LSMI1

Base Object: EMPLOYEE

Timing: AFTER

Events:

Available Events:

Selected Events:
DELETE
INSERT
UPDATE

Columns:

Available Columns:
EMP_NAME
EMP_NO
EMP_SALARY
EMP_SDATE

Selected Columns:
DEPT_NO

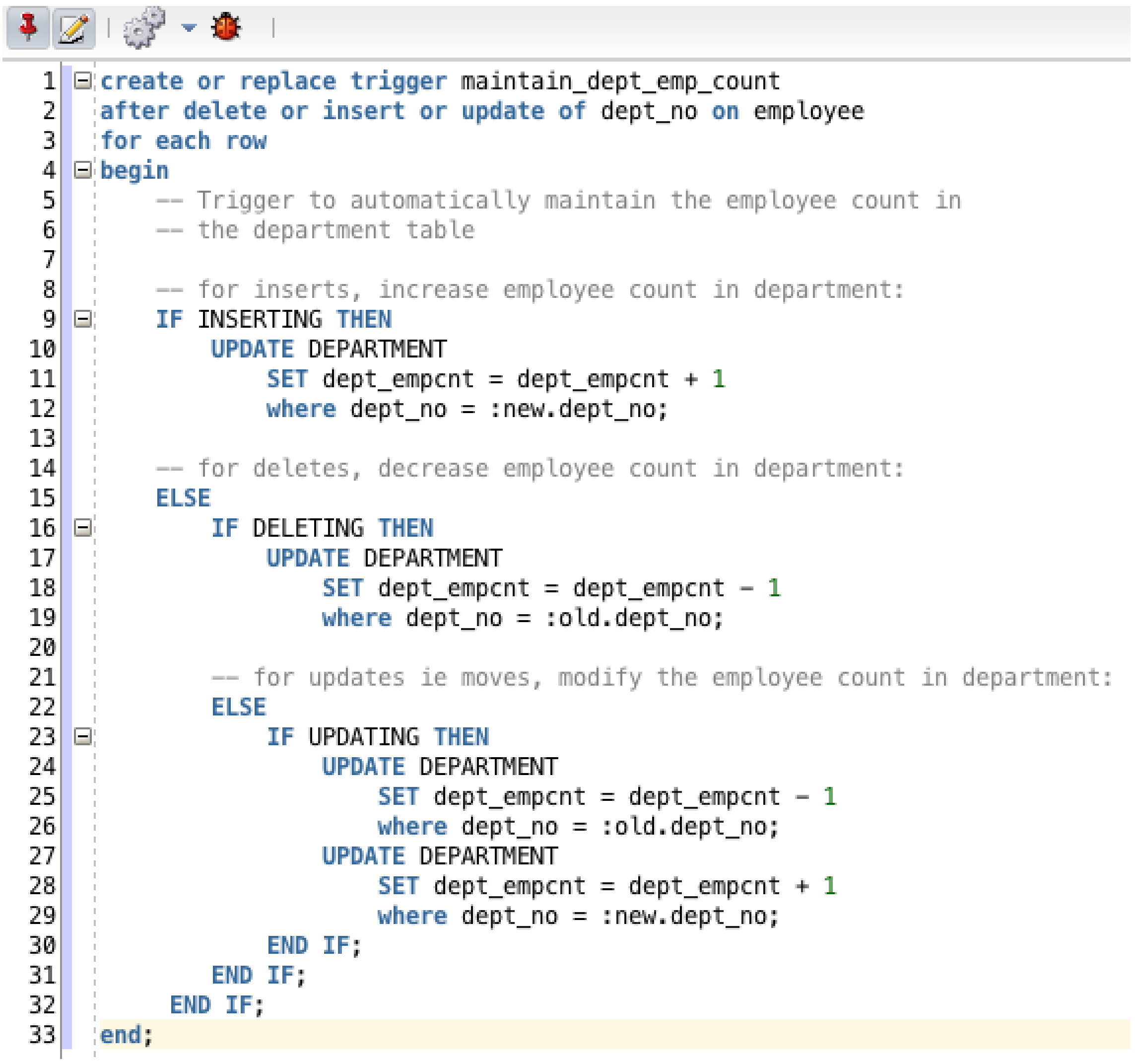
Referencing Old As:

[ ] Statement Level

Referencing New As:

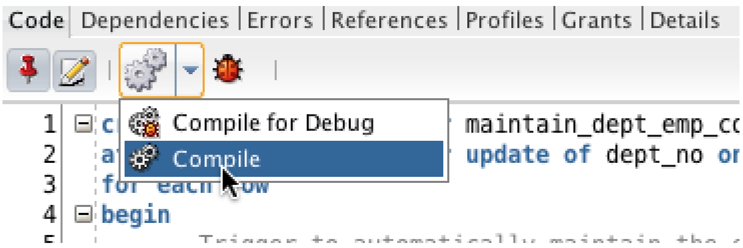
When Clause:

c. Select OK and then enter the trigger BODY:



```
create or replace trigger maintain_dept_emp_count
after delete or insert or update of dept_no on employee
for each row
begin
    -- Trigger to automatically maintain the employee count in
    -- the department table

    -- for inserts, increase employee count in department:
    IF INSERTING THEN
        UPDATE DEPARTMENT
            SET dept_empcnt = dept_empcnt + 1
            where dept_no = :new.dept_no;

    -- for deletes, decrease employee count in department:
    ELSE
        IF DELETING THEN
            UPDATE DEPARTMENT
                SET dept_empcnt = dept_empcnt - 1
                where dept_no = :old.dept_no;

        -- for updates ie moves, modify the employee count in department:
        ELSE
            IF UPDATING THEN
                UPDATE DEPARTMENT
                    SET dept_empcnt = dept_empcnt - 1
                    where dept_no = :old.dept_no;
                UPDATE DEPARTMENT
                    SET dept_empcnt = dept_empcnt + 1
                    where dept_no = :new.dept_no;
            END IF;
        END IF;
   END IF;
end;
```

d. finally compile the trigger, correcting any errors which appear:

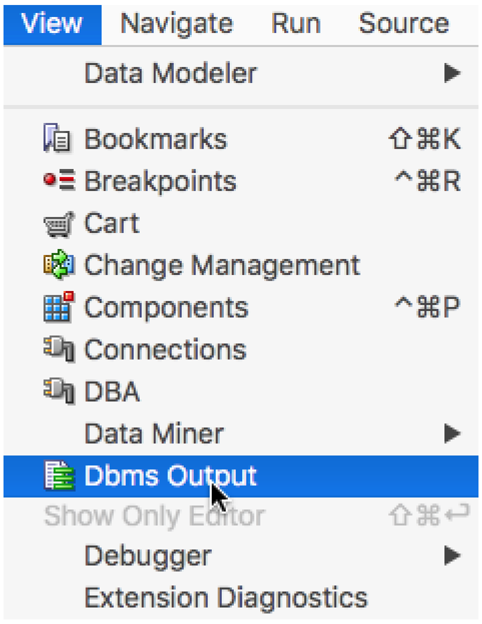


The schema and trigger body code for this example are available in the archive dept-emp.zip available from Moodle.

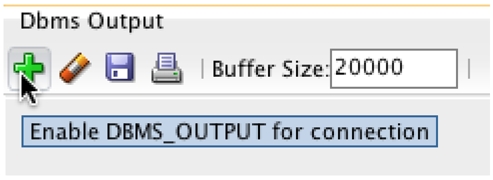
You can export completed trigger PL/SQL code (right click the trigger and select export) or simply copy and paste the code directly into a SQL script file.

If your output includes a message which will be written to the screen using DBMS_OUTPUT.PUT_LINE you will need to turn on DBMS Output.

To obtain DBMS_OUTPUT in SQL Developer you need to:
(a) Select View - Dbms Output



(b) In the output window which appears, select the green cross and then the connection you wish to receive output from:



Using the archive available from Moodle (dept-emp.zip), create the tables with the schema file, and create the trigger using the above process, copying the triggers body code from the text file in the archive (copy and paste it into the trigger editor).

After you have successfully created the trigger perform the following tasks using SQL:

1. Insert two departments, each with an employee count of zero.
2. Now insert an employee and observe what happens with the DEPARTMENT.DEPT_EMPCNT attribute (the count of the employees in a department).
3. Move the employee to the other department (update the employee dept_no), again check the effect on the DEPARTMENT.DEPT_EMPCNT attribute.

Lastly delete the employee, again check the effect on the DEPARTMENT.DEPT_EMPCNT attribute. Drop the employee and department tables.

Task B: Unit-Enrolment-Student

Using the archive student-enrol.zip supplied on Moodle create and load the tables for task B.

1. Create a trigger unit_upd_cascade that updates matching rows in the ENROLMENT table whenever a unit code is updated in the UNIT table. The changes in the enrolment table should be announced via dbms_output.put_line

Check the trigger works using SQL, record your SQL commands and the result. Hint - update a unit code in the unit table and observe what happens in the enrolment table.
2. Create a trigger as discussed in the lecture, to

- Maintain the integrity of the total number of students in a given unit in the UNIT table
- Record a delete operation in the audit_log

Check the trigger works using SQL, record your SQL commands and the result. Hint - enrol and remove an enrolment (delete) for a student.

For students who would like a challenge consider how you might code the trigger to maintain the student's average mark, and what the problems are.

3. Create a trigger called calculate_grade that calculates the student's grade (enrolment.enrol_grade) whenever a mark is updated for an enrolment or a new enrolment is added with a mark. Hint: you can avoid the "mutating table" error here by directly manipulating the new value of enrol_grade in your trigger based on the new enrol_mark value.

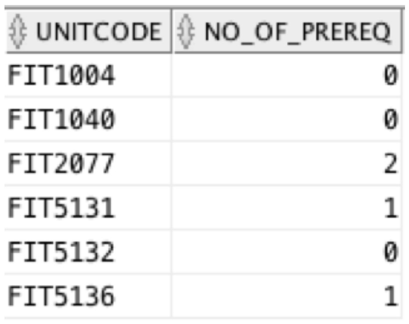
- enrol_mark >= 80 is a HD grade
- enrol_mark >= 70 and enrol_mark < 80 is a D grade
- enrol_mark >= 60 and enrol_mark < 70 is a C grade
- enrol_mark >= 50 and enrol_mark < 60 is a P grade
- enrol_mark < 50 is a N grade

Check the trigger works using SQL, record your SQL commands and the result.

11
SQL Part IV - SQL Advanced

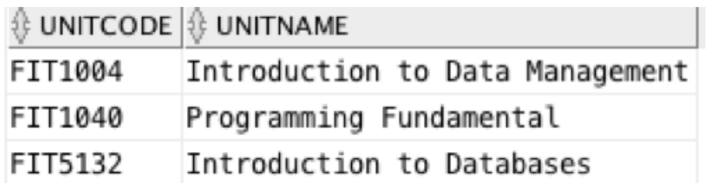
This week we will continue to use the UNIVERSITY database model.

Q1. Find the total number of prerequisite units for each unit. Include in the list the unitcode of units that do not have prerequisite. Hint: use an outer join.



UNITCODE	NO_OF_PREREQ
FIT1004	0
FIT1040	0
FIT2077	2
FIT5131	1
FIT5132	0
FIT5136	1

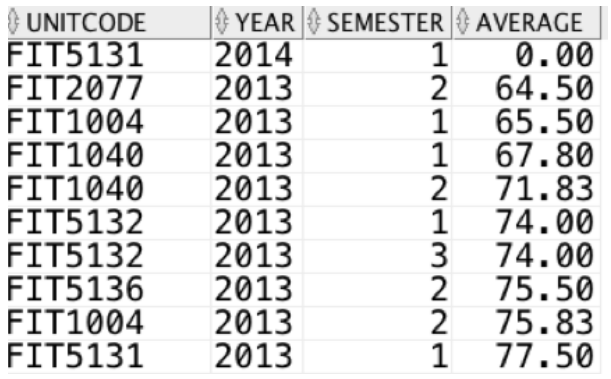
Q2. Display unitcode and unitname for units that do not have prerequisite.



UNITCODE	UNITNAME
FIT1004	Introduction to Data Management
FIT1040	Programming Fundamental
FIT5132	Introduction to Databases

There are many approaches that you can take in writing an SQL statement to answer this query. You can use the SET OPERATORS, OUTER JOIN and a SUBQUERY. Write SQL statements based on *all* of these approaches.

Q3. List the unit code and the average mark for each offering. Round the average to 2 digits after the decimal points. If the average result is 'null', display the average as 0.00. All values must be shown with two decimal digits:



UNITCODE	YEAR	SEMESTER	AVERAGE
FIT5131	2014	1	0.00
FIT2077	2013	2	64.50
FIT1004	2013	1	65.50
FIT1040	2013	1	67.80
FIT1040	2013	2	71.83
FIT5132	2013	1	74.00
FIT5132	2013	3	74.00
FIT5136	2013	2	75.50
FIT1004	2013	2	75.83
FIT5131	2013	1	77.50

12
Database Web Interfaces - Querying Data

Now that you are familiar with designing, creating and managing tables we will look at the manner in which such data can be accessed. To date you have been using SQL Developer, clearly, in practice, your users will not have access to/use this item of software. Access to your created tables by normal users will be via an application or web front end. For this tutorial, we are going to develop a ***basic*** web front end. To do this we are going to make use of PHP (http://php.net/) (recursive acronym for PHP: Hypertext Preprocessor) one of the most widely used programming languages (http://www.tiobe.com/tiobe-index/), especially for web development. Please note that this is a *very basic* introduction so that you become aware of the possibilities. PHP is a full OO language with a wide range of features. Development under PHP is commercially carried out via PHP Frameworks (https://www.searchenginejournal.com/guide-popular-php-frameworks-beginners/180922/). Monash offers several units in which you can further advance your understanding of PHP, and PHP is often used in final year industrial projects within a course.

PHP enables the mixing of PHP code (marked between <?php and ?>) and standard HTML code within a single file. When accessed via the web server the PHP code is handled via the PHP processor on the web server and replaced with appropriate output. For this unit we are not expecting you to become PHP experts, this is simply an exercise to increase your awareness of how a database can be accessed via the web. If you wish to delve further into PHP immediately there are a large number of good tutorials available on the web. A good starting point is https://www.w3schools.com/php/

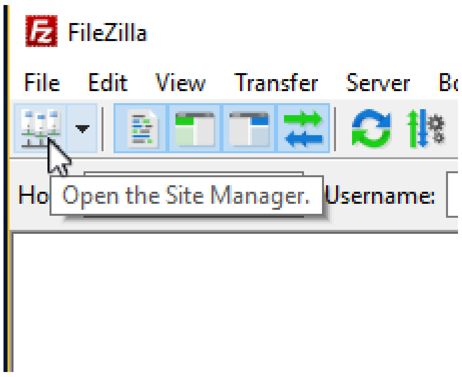
The steps in using PHP to access table data are:

- connect to the database
- define a SQL query string
- parse the SQL query against the database
- execute the statement
- fetch and display the data, and finally
- free the resources being used and close the connection.

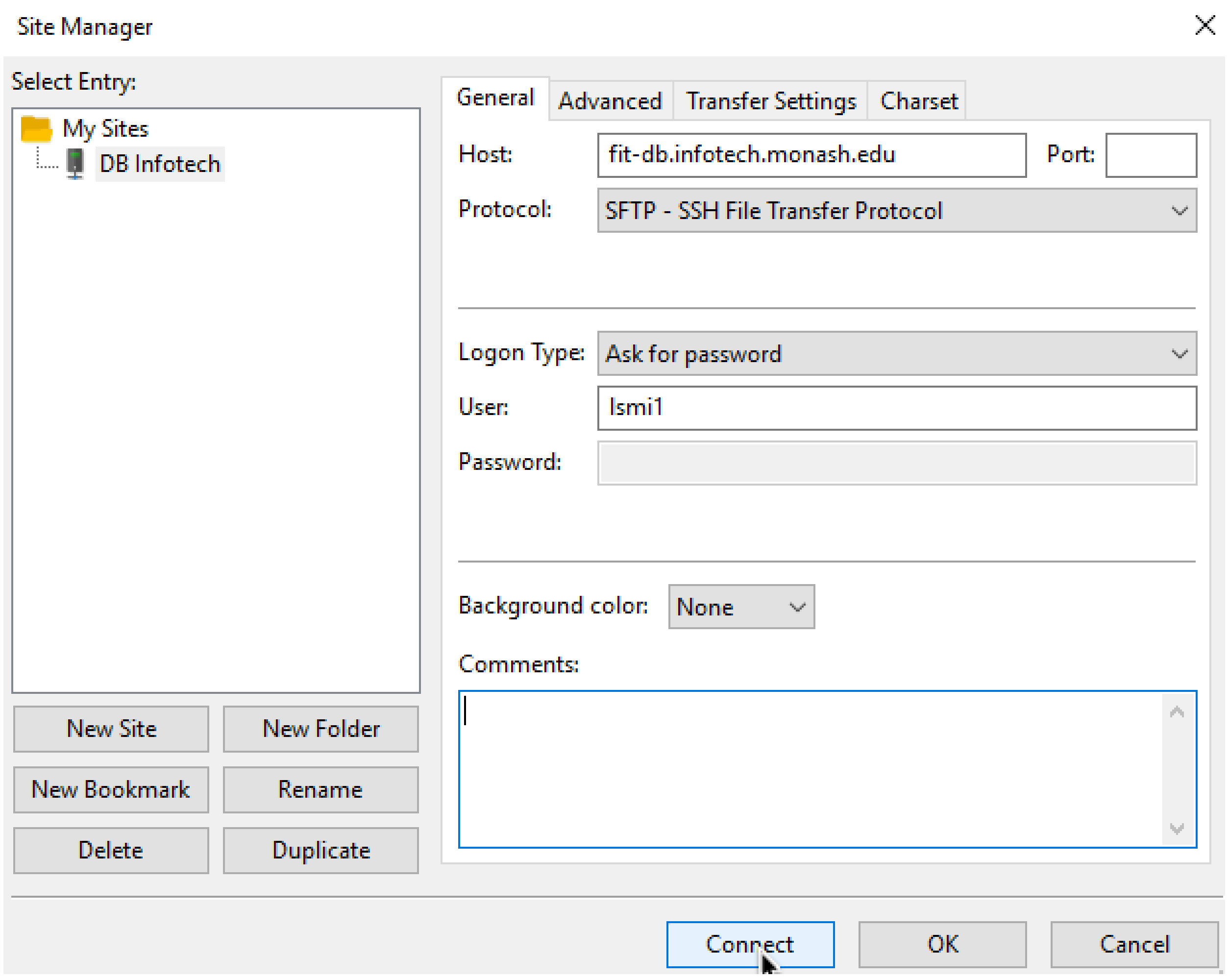
You have been given an account on the server http://fit-db.infotech.monash.edu/ - this server can be accessed in two ways:

- via the web by using a URL of the form: http://fit-db.infotech.monash.edu/~*yourauthcateusername*/ or
- via SFTP for transferring files to the server.

To create a connection to the server open an SFTP client such as FileZilla (https://filezilla-project.org/) (installed in the on-campus labs). With FileZilla select Open the Site Manager:



and enter the server name (host) and access type (protocol):



Host: fit-db.infotech.monash.edu
Port:
Protocol: SFTP - SSH File Transfer Protocol
Logon Type: Ask for password
User: lsmi1
Password:
Background color: None

Before selecting "Connect", click on the NewSite under My Sites on the left and give your connection an appropriate name (here DB Infotech). Also please ensure you use the "Logon Type" "Ask for password" so that FileZilla will prompt you for your password at each connection and not remember it. After clicking on "Connect", your connection will open and FileZilla will show your local files on the left and the server (remote) files on the right. Navigate using the right until you are located in your public_html folder. Documents placed in this folder will be published via the web server running on http://fit-db.infotech.monash.edu/.

To create the PHP files we will need you should make use of a text editor such as Notepad++ (https://notepad-plus-plus.org/) (MS Windows) or TextWrangler (http://www.barebones.com/products/TextWrangler/) (Mac)

The first PHP script we wish to create is a file containing your Oracle connection details, we will include this into PHP scripts we create to access the database. The file should be called connection.php and have the form:

```
<?php

$dbUserName = "myauthcateusername";
$dbPassword = "mypassword";
$dbSID = "myOracleSID";

?>
```

Each of the three values needs to be replaced with an appropriate value for you (make sure the " remain, just replace the actual values). Save this file as connection.php, if you are in the on-campus lab this should be on your mapped network drive eg. U:

Now we wish to create the actual PHP script which will access and display the student data in the student table of the UNIVERSITY database that you used in weeks 8,9 and 10. The following code contains all the required material to carry out a select of the student table and display the results in an HTML table (https://www.w3schools.com/TAGs/tag_table.asp). You should look through the code and understand the details of what has been coded:

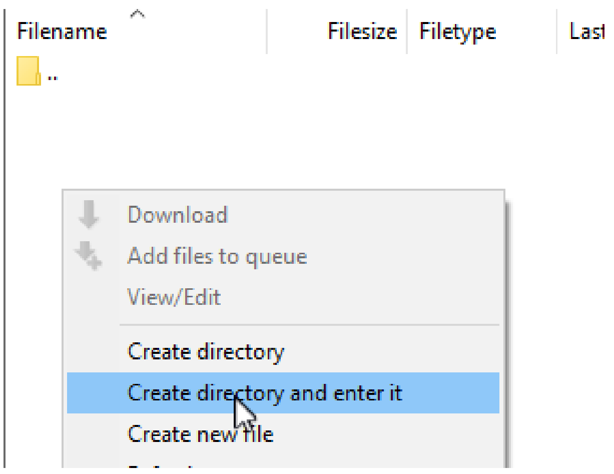
```
<!--
 BASIC PHP/Oracle script
 FIT Databsse Teaching Team
-->

<html>
<head>
 <title>Student List</title>
</head>

<body>
<h1>Student list UNIVERSITY database</h1>

<?php

// Include connection details
include("connection.php");

//Disable error reporting to HTML page, however
//DO NOT comment this out until the script has been debugged
error_reporting(0);

?>

<?php
```

```
// Set up the Oracle connection string
// dbSID comes from the connection.php file
$dbInstance = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)
   (HOST=fit2094.corp-prd.aws.monash.edu)(PORT=1521))
   (CONNECT_DATA=(SID=$dbSID)))";

// Connect to the database - Open Oracle connection
// dbUserName and dbPassword come from the connection.php file
$conn = oci_connect($dbUserName, $dbPassword, $dbInstance);
if (!$conn) {
 $e = oci_error();
 print "Error connecting to the database:<br>" ;
 print $e['message'] ;
 exit;
}
?>

<!-- create HTML table header to display output -->
<table border =1 width=600>
<tr>
 <th width=100><b>Student ID</b></td>
 <th width=200><b>Name</b></td>
 <th width=100><b>Birth Date</b></td>
 <th width=200><b>Email</b></td>
</tr>

<?php
//SQL query statement
$query = "SELECT studid,
 rtrim(studfname) || ' ' || rtrim(studlname) as sname,
 to_char(studdob,'dd-Mon-yyyy') as sbdate,
 studemail
 FROM uni.student
 ORDER BY studid";

//Parse statement
$stmt = oci_parse($conn,$query);
if (!$stmt) {
 $e = oci_error($conn);
 print "Error on parse of statement:<br>" ;
 print $e['message'] ;
 exit;
}

// oci_define_by_name maps SQL Columns in a queryto PHP variable names
// NB - MUST be done before executing, Oracle names must be in UPPER
case
oci_define_by_name($stmt,"STUDID",$studid);
oci_define_by_name($stmt,"SNAME",$stuname);
oci_define_by_name($stmt,"SBDATE",$stubdate);
oci_define_by_name($stmt,"STUDEMAIL",$stuemail);

// Execute the STATEMENT
$r = oci_execute($stmt);
```

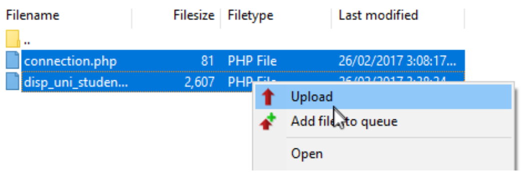
```
if (!$r) {
 $e = oci_error($stmt);
 print "Error execute of statement:<br>" ;
 print $e['message'] ;
 exit;
}

// Fetch the results of the query
while (oci_fetch($stmt)) {
 print("
 <tr>
 <td width=100>$studid</td>
 <td width=100>$stuname</td>
 <td width=100>$stubdate</td>
 <td width=100>$stuemail</td>
 </tr>");
}

// Close table
print ("</table>");

$no_rows = oci_num_rows($stmt);
print "<p>Rows found:" . $no_rows . "</p>";

// Free resources associated with Oracle statement
oci_free_statement($stmt);
// Close the Oracle connection
oci_close($conn);

?>

</body>
</html>
```

After you have checked through this code, copy and paste the code into a new text document called disp_uni_students.php saved in the same location as your connection.php file.

We now wish to transfer both of these files to the remote server into a folder below public_html called uni. Right click in your remote public_html folder and create a uni folder (select "Create … enter" in Filezilla):



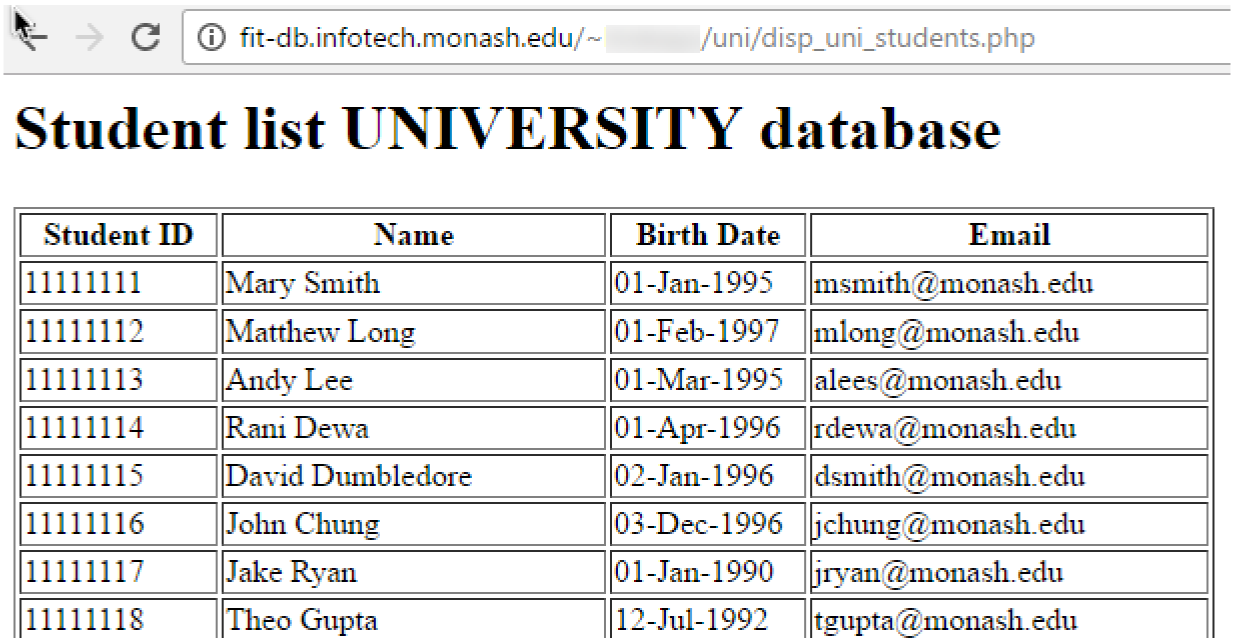
If you are not using Filezilla, navigate to this folder. Finally, locate your connection.php and disp_uni_students.php files in the right-hand file browser of Filezilla and transfer them to your remote unit folder via upload:



Well done, you are now ready to examine your work. Go to the url:

http://fit-db.infotech.monash.edu/~*yourauthcateusername*/uni/disp_uni_students.php

(replace *yourauthcateusername* with your actual username), and you should see something like:


fit-db.infotech.monash.edu/~[redacted]/uni/disp_uni_students.php

Student list UNIVERSITY database

Student ID	Name	Birth Date	Email
11111111	Mary Smith	01-Jan-1995	msmith@monash.edu
11111112	Matthew Long	01-Feb-1997	mlong@monash.edu
11111113	Andy Lee	01-Mar-1995	alees@monash.edu
11111114	Rani Dewa	01-Apr-1996	rdewa@monash.edu
11111115	David Dumbledore	02-Jan-1996	dsmith@monash.edu
11111116	John Chung	03-Dec-1996	jchung@monash.edu
11111117	Jake Ryan	01-Jan-1990	jryan@monash.edu
11111118	Theo Gupta	12-Jul-1992	tgupta@monash.edu

Tutorial Task

Create a *new* PHP script which will report all unit offerings from the UNIVERSITY database - the report should show the unit code, unit name, semester and year of offering (in the form "S2 2017") and the staff name of the chief examiner (in the form "Leroy Brown").