

FIT2094-FIT3171 2019 S1 -- Week 2 eBook

Credits and Copyrights:

Authors: FIT3171 2018 S2 UG Databases

FIT Database Teaching Team

Maria Indrawan, Manoj Kathpalia, Lindsay Smith, et al

Copyright © Monash University, unless otherwise stated. All Rights Reserved, except for individual components (or items) marked with their own licence restrictions

Change Log:

- Formatted and Imported from Alexandria March 2019.

FIT2094-FIT3171 2019 S1 -- Week 1 eBook	1
1.0. Introduction to SQL Developer	3
1.0.1 SQL Developer Preference Settings	4
Display Line Numbers	4
SQL Code AutoFormat	5
PL/SQL Scope Identifiers	7
Navigation Filter (Optional)	8
Copying your configuration to a new PC/Laptop	10
Copying your configuration to a new PC/Laptop - Mac instructions	10
SQL Developer Language	11
1.0.1. Working in the On-Campus Labs	12
Basics - Windows 10 on Lab PCs	13
Follow-up for Mac users	17
Follow-up for Linux users	18
1.1. Connecting to Oracle database using SQL Developer	20
1. Adding a new connection	20
2. Changing your password	21
1.2. Pre-Lecture Notes	23

2.0. Using SQL Developer GUI to manage data

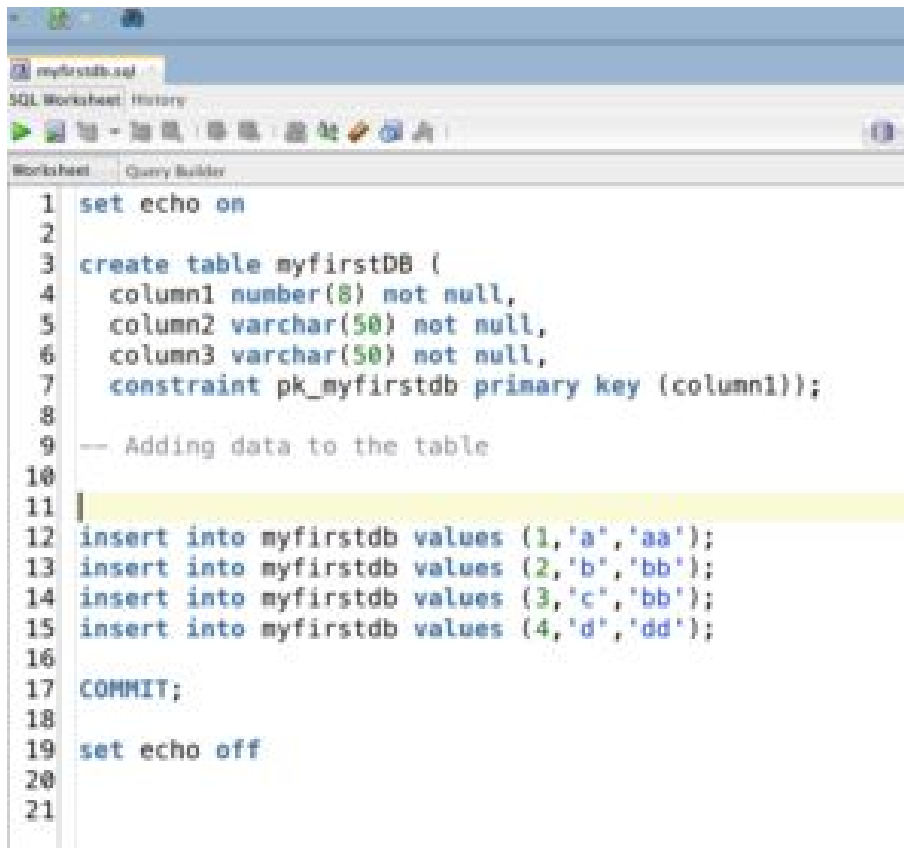
There are two main approaches to interact with Oracle database using SQL Developer, through the Graphical User Interface (GUI) and SQL Worksheet. In this module, you will learn to use the SQL Worksheet to create the database and to use the GUI to add, update and delete data from the database.

2.0.1. 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 the Oracle server.

4. Drag the **myfirstDB.sql** to the SQL Worksheet area.



```
1 set echo on
2
3 create table myfirstDB (
4     column1 number(8) not null,
5     column2 varchar(50) not null,
6     column3 varchar(50) not null,
7     constraint pk_myfirstdb primary key (column1));
8
9 -- Adding data to the table
10
11
12 insert into myfirstdb values (1,'a','aa');
13 insert into myfirstdb values (2,'b','bb');
14 insert into myfirstdb values (3,'c','bb');
15 insert into myfirstdb values (4,'d','dd');
16
17 COMMIT;
18
19 set echo off
20
21
```

5. Run the SQL script by pressing the “run script” icon.

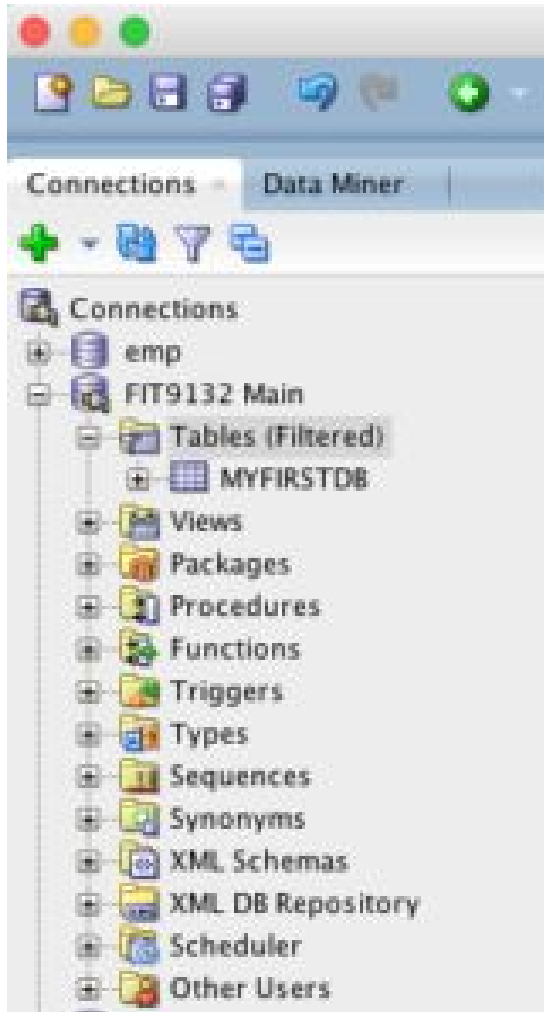


2.0.2. Viewing table structure and data inside a table.

1. To view the table using the graphical user interface, expand the **Tables** option in the Connection tab and find “**myfirstdb**” from the list.

NOTE: Table names are often capitalised in Oracle. More details? Read the footnote :-)¹

¹ <https://community.oracle.com/thread/1000437?start=0&tstart=0>

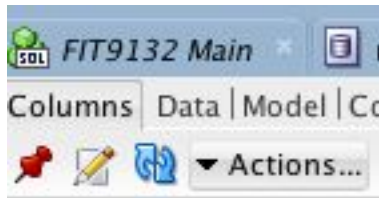


Double click on the “**myfirstdb**” entry.

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

2.0.3. 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	z	aa
2	2 b		bb
3	3 c		bb
4	4 d		dd


- 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.

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

2.0.4. Adding a new row to the table.

- Click on the insert new row icon. 
- 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)

- Replace the (null) values with 5, e, and ee, respectively.
- 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

2.0.5. 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 populate the data. To perform this task, you need to make sure you have completed the module **Introduction to SQL Developer** (from Week 1).

2.1.1. 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 Developer².
5. Run the **student_poor.sql**.
6. Run the **student_good.sql**.

² If you have forgotten how to do so, refer to Section 2.0.1 above!

2.1.2 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**

2.1.3. Tasks

- Locate the listing of these tables in SQL Developer to check whether you have completed Step 5 and 6 of S2.1.1 (Creating a database) correctly.
- Perform 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 actions, perform the action in both the poorly designed and well designed databases. Compare the observations when the action took place in those databases.
- Remember to take your own notes as you perform the steps and answer the self-observation questions below.

DATABASE OPERATIONS:

UPDATE

1. Change the name of **FIT9131** for student number **1111** into Foundation of Java in the **student_poor** table ('**poorly designed database**').
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 need 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-designed 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-designed 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-designed 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 with name 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-designed 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?

2.1.4. 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 designed 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 the Relational model, its theory and design methodology.

2.1.5. Advanced Question

If your tutor has time at the end of the class, let's apply your knowledge from student_good and student_poor above to see if you can solve a real-world problem.

Let's say you are advising a popular artist in setting up yet another music streaming service (to compete with Spotify, Apple Music, TIDAL, etc :-). You have a simple database containing song information, a portion of which is provided below as a screenshot from SQL Developer.

SONG_ID	TITLE	ARTIST	ALBUM	GENRE	YEAR	NUMBER_IN_ALBUM
1	Eastside	benny blanco, Halsey & Khalid	Friends Keep Secrets	dance, pop	2018	1
2	Never Gonna Give You Up	Rick Astley	Whenever You Need Somebody	pop	1987	1
3	All Star	Smash Mouth	Astro Lounge	rock	1999	5
4	Lose Yourself	Eminem	8 Mile Soundtrack	rap	2002	1
5	Teardrops on My Guitar	Taylor Swift	Taylor Swift	country	2006	3
6	Bad Blood	Taylor Swift feat. Kendrick ...	1989	POP	2014	8

What is bad about the design? Hints:

- Multiple artists per song?
- Multiple genres per song?
- pop vs POP?
- Same artist for multiple songs?

Copyright © Monash University, unless otherwise stated. All Rights Reserved, except for individual components (or items) marked with their own licence restrictions.

1.2. Pre-Lecture Notes

These are notes you may find useful to read through before the lecture, adapted from Lindsay Smith's lecture material. **NOTE: THESE ARE NOT THE FINAL LECTURE SLIDES.**

As a start, do read up about Database Relational Models on Wikipedia
https://en.wikipedia.org/wiki/Relational_model

Then, read up below:

The Relational Model

- Introduced by CODD in 1970 - the fundamental basis for relational DBMS's
- Basic structure is the mathematical concept of a RELATION mapped to the 'concept' of a table (tabular representation of relation)
 - Relation - abstract object
 - Table - pictorial representation
 - Storage structure - "real thing" - eg. isam file
- Relational Model Terminology
 - DOMAIN - set of atomic (indivisible) values
 - specify
 - name
 - data type
 - data format
- Examples:
 - customer_number domain - 5 character string of the form xxxdd
 - name domain - 20 character string
 - address domain - 30 character string containing street, town & postcode
 - credit_limit domain - money in the range \$1,000 to \$99,999

A Relation

- A relation consists of two parts
 - heading
 - body
- Relation Heading
 - Also called Relational **Schema** consists of a fixed set of attributes
 - $R(A_1, A_2, \dots, A_n)$
 - R = relation name, A_i = attribute i
 - Each attribute corresponds to one underlying domain:
 - Customer relation heading:
 - CUSTOMER (custno, custname, custadd, credlimit)
 - » $\text{dom}(\text{custno}) = \text{customer_number}$
 - » $\text{dom}(\text{custname}) = \text{name}$
 - » $\text{dom}(\text{custadd}) = \text{address}$
 - » $\text{dom}(\text{credlimit}) = \text{credit_limit}$

custno	custname	custadd	credlimit
--------	----------	---------	-----------

Relation Body

- Relation Body
 - Also called Relation Instance (state)
 - $r(R) = \{t_1, t_2, t_3, \dots, t_m\}$
 - consists of a time-varying set of n -tuples
 - Relation R consists of tuples $t_1, t_2, t_3 \dots t_m$
 - m = number of tuples = **relation cardinality**
 - each n -tuple is an ordered list of n values
 - $t = \langle v_1, v_2, \dots, v_n \rangle$
 - n = number of values in tuple (no of attributes) = **relation degree**
 - In the tabular representation:
 - Relation heading \Rightarrow column headings
 - Relation body \Rightarrow set of data rows

custno	custname	custadd	credlimit
SMI13	SMITH	Wide Rd, Clayton, 3168	2000
JON44	JONES	Narrow St, Clayton, 3168	10000
BRO23	BROWN	Here Rd, Clayton, 3168	10000

Relation Properties

- No duplicate tuples
 - by definition sets do not contain duplicate elements
 - hence tuples are unique
- Tuples are unordered within a relation
 - by definition sets are not ordered
 - hence tuples can only be accessed by content
- No ordering of attributes within a tuple
 - by definition sets are not ordered

Relational Keys

- A candidate key K of a relation R is an attribute or set of attributes which exhibits the following properties:
 - No two tuples of R have the same value for K
(Uniqueness property)
 - No proper subset of K has the uniqueness property
(Minimality or Irreducibility property)

Relational Database

- A relational database is a collection of normalised relations.
- Normalisation is part of the design phase of the database and will be discussed in a later lecture.

Example relational database:

order (order_id, orderdate,)
order-line (order_id, product_id, quantity)
product (product_id, description, unit_price)

Relational Algebra

- Relationally complete.
- Procedural.
- Operators only apply to at most two relations at a time.
- 8 basic operations:
 - single relation: selection, projection
 - cartesian product, join
 - union
 - intersection
 - difference
 - division

EOF.