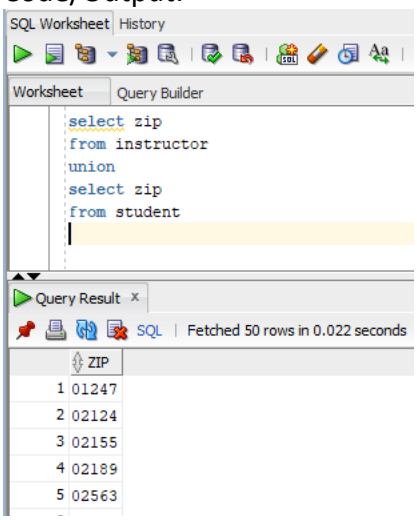
Ryan Paw
Week 3, Lab
ANA 660

Lab 9.1 Exercises (all the questions)
Lab 9.2 Exercises (all the questions)
Lab 13.3 Exercises (questions a, b, c, d, and e)
Lab 14.2 Exercises (questions a, b, and c)
Lab 15.1 Exercises (questions a, b, c, and h)
Lab 17.1 Exercises (all the questions)

I, Ryan Paw, finished and completed this week's reading requirements and lab activities. I created and ran the code in SQL Plus myself.

9.1, a)
The output for the code shows the first and last names of the instructors and students. The UNION ALL helps combine the "instructor" and "student" tables into one table. It combines the result set of two or more SELECT statements and allows duplicate values. In this example, the UNION ALL operation works because both "Instructor" and "Student are in the same position for each SELECT statement.

9.1, b)
Code/Output:



9.1, c)
The example code helps solve the problem of taking multiple tables and combining them into one, consolidated table. In the example, it combines "enrollment", "grade", "grade_type", "grade_conversion". The UNION operator helps combine these tables and eliminates any duplicates.

9.1, d)

The example code returns an error because the SQL command is not properly ended (ORA-00933). This is because an ORDER BY needs to be edited in this query, and needs to be added at the end of the statement.

Error:

Worksheet | Query Builder

```
select course_no, description
from course
where prerequisite is not null
order by 1
union
select course_no, description
from course
where prerequisite is null
```

Query Result

SQL | Executing:select course_no, descriptionfrom coursewhe

```
ORA-00933: SQL command not properly ended
00933. 00000 - "SQL command not properly ended"
*Cause:
*Action:
Error at Line: 5 Column: 1
```

Correct output:

Worksheet | Query Builder

```
select course_no, description
from course
where prerequisite is not null
union
select course_no, description
from course
where prerequisite is null
order by 1
```

Query Result

SQL | All Rows Fetched: 30 in 0.065 seconds

	COURSE_NO	DESCRIPTION
1	10	Technology Concepts
2	20	Intro to Information Systems
3	25	Intro to Programming
4	80	Programming Techniques
5	100	Hands-On Windows
6	120	Intro to Java Programming
7	122	Intermediate Java Programming

9.1, e)

The example code returns an error because the table or view does not exist (ORA-00942). This is because the data types are not the same from the two tables. The order of the columns needs to be revered so that the id field is the first, then last_name.

Error:

```
select instructor_id, last_name
from instructor
union
select last_name, student_id
from students
```

Query Result

SQL | Executing:select instructor_id, last_namefrom instr

```
ORA-00942: table or view does not exist
00942. 00000 - "table or view does not exist"
*Cause:
*Action:
Error at Line: 5 Column: 6
```

9.2, a)

The output links the course_no from the section and course tables, and will list the courses and their description. The MINUS operator takes away all the courses with sections from all courses. Therefore, the output will show courses that do not have matching sections.

```
select course_no, description
from course
minus
select s.course_no, c.description
from section s, course c
where s.course_no = c.course_no
```

Query Result

SQL | All Rows Fetched: 2 in 0.05 seconds

	COURSE_NO	DESCRIPTION
1	80	Programming Techniques
2	430	Java Developer III

9.2, b)

Code/Output:

Worksheet | Query Builder

```
select course_no, section_no, 'no enrollments' "status"
from section
minus
select course_no, section_no, 'no enrollments'
from section s
where exists
    (select section_id
    from enrollment e
    where e.section_id=s.section_id)
```

Query Result

All Rows Fetched: 14 in 0.037 seconds

	COURSE_NO	SECTION_NO	status
1	25	9	no enrollments
2	124	4	no enrollments
3	125	6	no enrollments
4	130	3	no enrollments
5	132	3	no enrollments
6	134	2	no enrollments
7	135	3	no enrollments
8	140	3	no enrollments
9	142	3	no enrollments
10	144	2	no enrollments
11	145	3	no enrollments

9.2, c)

Code/Output:

Worksheet | Query Builder

```
select zip
from instructor
intersect
select zip
from student
```

Query Result

All Rows Fetched: 1

	ZIP
1	10025

9.2, d)

Worksheet | Query Builder

```
select student_id
from student
intersect
select student_id
from enrollment
```

Query Result | SQL | Fetched 50 rows in 0.031 seconds

	STUDENT_ID
1	102
2	103
3	104
4	105
5	106
6	107
7	108
8	109

13.3, a)
Code/Output:

Worksheet | Query Builder

```
create or replace view long_distance_student as
select s.*, z.city, z.state
from student s, zipcode z
where s.zip = z.zip
and state not in ('NJ','NY','CT')
```

Query Result | Script Output | Task completed in 0.421 seconds

```
View LONG_DISTANCE_STUDENT created.
```

13.3, b)
Code/Output:

SQL Worksheet | History

Worksheet | Query Builder

```
select *
from course
where cost <= 1095
```

Script Output | Query Result

All Rows Fetched: 3 in 0.034 seconds

	COURSE_NO	DESCRIPTION	COST	PREREQUISITE	CREATED_BY	CREATED_DATE	MODIFIED_BY	MODIFIED_DATE
1	135	Unix Tips and Techniques	1095	134	DSCHERER	29-MAR-07	ARISCHER	05-APR-07
2	230	Intro to the Internet	1095	10	DSCHERER	29-MAR-07	ARISCHER	05-APR-07
3	240	Intro to the BASIC Language	1095	25	DSCHERER	29-MAR-07	ARISCHER	05-APR-07

13.3, c)

I notice that I can still insert data into the table, even though the data I inserted is higher than a cost of 1095. To change this, we need to insert a "check option constraint" into SQL, so whenever new data is entered that violates the constraint, SQL will show an error. The WHERE clause in SQL does not work for DML statements.

13.3, d)

Code/Output:

```
drop view long_distance_student;
drop view cheap_course;
```

Script Output | Query Result

Task completed in 0.239 seconds

```
View LONG_DISTANCE_STUDENT dropped.


View CHEAP_COURSE dropped.
```

13.3, e)

Code/Output:

Create table test_tab / create view test_tab_view / describe view (desc) shows description of test_tab_view:

Worksheet | Query Builder

```
1  create table test_tab
2      (col1 number)
3
4  create or replace view test_tab_view as
5  select *
6      from test_tab
7
8  alter table test_tab
9      add (col2 NUMBER)
10
11 desc test_tab_view
13 select text
14 from user_views
15 where view_name = 'TEST_TAB_VIEW'
```

Script Output | Query Result | Task completed in 0.389 seconds

```
*Action:

Table TEST_TAB created.

View TEST_TAB_VIEW created.

Table TEST_TAB altered.

Name Null? Type
---- ----- ------
COL1       NUMBER
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.084 seconds

	TEXT
1	select "COL1" from test_tab

Re-issue creation of the view statement:

```
17 create or replace view test_tab_view as
18 select *
19 from test_tab
20
21 desc test_tab_view
```

```
View TEST_TAB_VIEW created.

Name Null? Type
---- ----- ------
COL1       NUMBER
COL2       NUMBER
```

Drop test_tab and test_tab_view:

```
23 drop table test_tab;
24 alter view test_tab_view COMPILE;
```

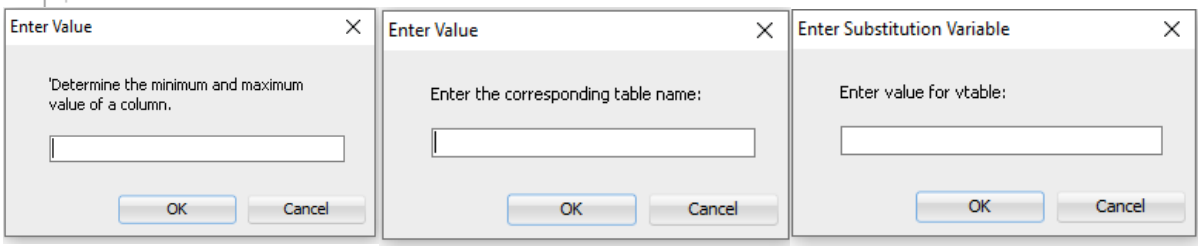
```
Table TEST_TAB dropped.

Warning: execution completed with warning

View TEST_TAB_VIEW altered.
```

14.2, a)
Code/Output:
The example code shows prompts that we can input values in (see screenshots below)

```
accept vcol char prompt 'Determine the minimum and maximum value of a column.
Enter the column name: '
accept ctable char prompt 'Enter the corresponding table name: '
set verify off
select min(&vcol), max(&vcol)
from &vtable
```



14.2, b)
The purpose of this code shows the compiled user-accessible tables and a row count for each table.

Line 1-8: This is a comment in SQL and it explains the file name, purpose, who it was created by, and who it was modified by.
Line 9: SET TERM OFF suppresses the display so you don't see the output on the screen
Line 10: PAGESIZE is set to 0. This tells SQL the number of printed lines that will fit on one page of the output.
Line 11: FEEDBACK is turned off. This gives feedback SELECT, UPDATE, INSERT statements.
Line 12: SPOOL is turned off. When SPOOL is turned off, it sends output of statements to a file called output.txt in the directory where the MaxL shell is located in your computer.
Line 13: This is a literal SELECT statement that uses 4 quotation marks. The quotation marks show the output of a single quotation mark in the file that was spooled. The name of the table is also displayed between them.
Line 14: The COUNT function in this line counts the rows. Char(10) function creates a new line for the spooled file.
Line 15: The results of the concatenation is combined here. The table name in lowercase letters is displayed with a semicolon.
Line 16: This line uses the FROM statement that references where the query is coming from. It is referencing "user_tables".
Line 17: This line ends the spooling.
Line 18-20: These lines resets the settings for FEEDBACK, PAGESIZE, and TERM to their defaults.
Line 21: This line runs the spooled file called "temp.lst" and issues the SQL statements

14.2, c)
I would execute a script like the example script if I need to re-create database objects. For example, if I need to make edits or changes to database objects this would be ideal, instead of redoing and re-running code.

15.1, a)

```
SQL> create user teacher identified by subject container=current;

User created.

SQL> grant connect, resource to teacher;

Grant succeeded.
```

```
SQL> create table account
  2     (account_num NUMBER(15),
  3     type    varchar2(10),
  4     status  varchar2(6),
  5  constraint account_pk PRIMARY KEY(account_num));

Table created.

SQL> insert into account
  2  (account_num, type, status)
  3  values
  4  (1001, 'Checking','Active');

1 row created.

SQL> COMMIT
  2  ;

Commit complete.
```

15.1, b)

User_role_privs lists what roles are available, and their corresponding privileges

```
SQL> select username, granted_role, admin_option
  2  from user_role_privs;

USERNAME
--------------------------------------------------------------------------------
GRANTED_ROLE
--------------------------------------------------------------------------------
ADM
---
SYSTEM
AQ_ADMINISTRATOR_ROLE
YES

SYSTEM
DBA
NO

USERNAME
--------------------------------------------------------------------------------
GRANTED_ROLE
--------------------------------------------------------------------------------
ADM
---
```

Session_privs shows what privileges are available to the current user.

```
SQL> select *
  2  from session_privs;

PRIVILEGE
----------------------------------------
ALTER SYSTEM
AUDIT SYSTEM
CREATE SESSION
ALTER SESSION
RESTRICTED SESSION
CREATE TABLESPACE
ALTER TABLESPACE
MANAGE TABLESPACE
DROP TABLESPACE
UNLIMITED TABLESPACE
CREATE USER

PRIVILEGE
----------------------------------------
BECOME USER
ALTER USER
DROP USER
CREATE ROLLBACK SEGMENT
ALTER ROLLBACK SEGMENT
DROP ROLLBACK SEGMENT
CREATE TABLE
CREATE ANY TABLE
```

15.1, c)

Note: After troubleshooting using online resources, I could not create connections which allows me to switch between users. I am currently running a Mac with Parallels, and I've run into error ORA-12541 multiple times that has prevented me from creating connections. Even though I've run into technical issues, I would like to explain what I should be observing as my output for this question.

In this question, the example code tries to insert data into the teacher.account table. However, it will result an error because the privileges do not align. Therefore, in SQL Plus, while logged on the TEACHER account, we need to grant permission for the STUDENT account. From the STUDENT account, we can then allow the same privileges to other users. The code should look like this:

CONN teacher/subject

GRANT SELECT ON account TO student WITH GRANT OPTION

--This code connects to the TEACHER user, and grants permission to the STUDENT user.

CONN student/learn

*SELECT **
FROM teacher.account

--This code connects to the STUDENT user, and the student queries to see the rows in the teacher.account table. If the STUDENT user tries to insert rows into the teacher account, their permission will be denied because they do not have permission to edit that account.

15.1, h)

Note: After troubleshooting using online resources, I could not create connections which allows me to switch between users. I am currently running a Mac with Parallels, and I've run into error ORA-12541 multiple times that has prevented me from creating connections. Even though I've run into technical issues, I would like to explain what I should be observing as my output for this question.

In this question, the following code can be used to create the roles and grant permissions:

CREATE ROLE student_admin

GRANT INSERT, UPDATE ON course TO student_admin

GRANT student_admin TO TEACHER

--After the code successfully runs, the student_admin role should be created and is granted permission to insert/update data for the course table. The last line of code grants permission to the TEACHER role.

17.1, a)

Code/Output:

Worksheet | Query Builder

```
select *
from
    (select cost, count(*) num_rows
    from course
    group by cost) course_cost
pivot
    (sum(num_rows)
    for cost in (1095, 1195, 1595, null))
```

Script Output | Query Result 2

SQL | All Rows Fetched: 1 in 0.047 seconds

	1095	1195	1595	NULL
1	3	25	1	1

17.1, b)

The example code was modified to display the top three revenue-generating courses. Any ties in revenue will show duplicates.

```sql
select course_no, revenue, rev_dense_rank
from
    (select course_no, revenue,
    dense_rank() over
        (order by revenue desc) rev_dense
    from course_revenue)
where rev_dense <=3
```

17.1, c)
The result of the AVG column is achieved by taking the average for each partition of the code, and then computing the cumulative average. When the values in the partition cluse change, the average is reset.

17.1, d)
The current query uses ENROLL_DATE to find the difference in days between the enroll date. However, it does not label the “days” in the DIFF and CUM_SUM columns. This is because DIFF is the difference in days between each value, and CUM_SUM is the cumulative sum of days.

17.1, e)
The example query shows the courses and number of students for each course. The number of students is the average enrollment per course, not including courses with the maximum number of enrollments. This code uses NUM_ENROLL and AVG_STUD_ENROLL to help generate the output. NUM_ENROLL calculates the number of enrolled students per course. AVG_STUD_ENROLL calculates the average number of students enrolled, not including the maximum number of enrollments.