CST2355 - Database Systems

# Assignment #3 Database Construction and Processing [15%]

*This assignment relates to the following Course Learning Requirements:*

CLR 2 - Develop Advanced Database Design and Normalization

CLR 4 - Use Oracle Procedural programming language (PL/SQL) to write programs that contain SQL statements

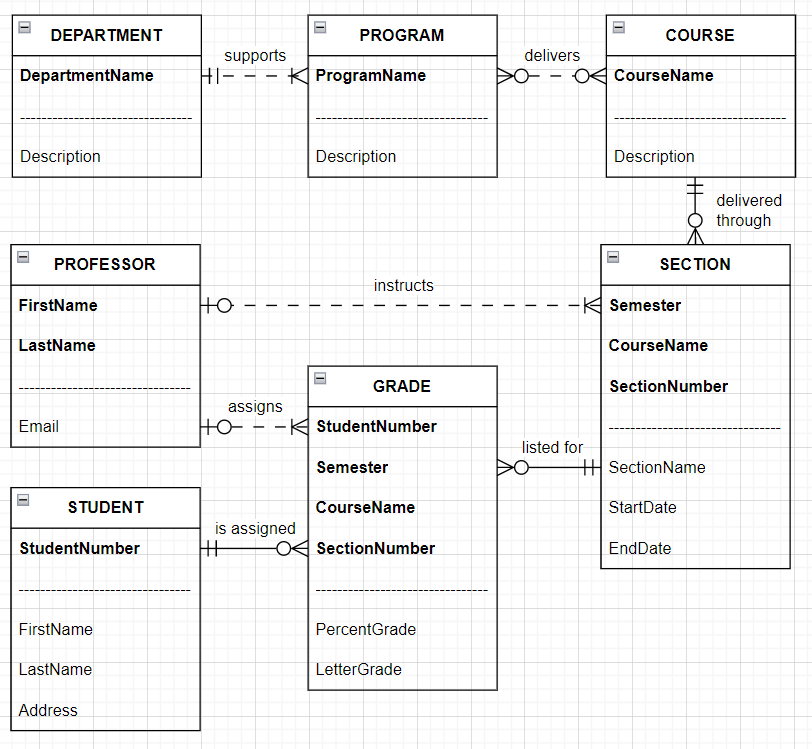
CLR5 - Develop advanced Database Queries

|  |
| --- |
| **adhere to the Academic Integrity policy - no sharing, no copying, no group work**  **provide ONLY your own work.** |

## Background

You are working for a college, enhancing a new grading system for students. From talking with the administration, you received a conceptual data model and a forward engineered script. You will use PL/SQL to add functionality such as data cleanup, business rules, and reports.

# Conceptual Data Model



## 

## Pre-Assignment Instructions

1 -- Read Modules 8 and 9 on Brightspace

2 -- Review the assignment support videos in the Additional Resources folder on Brightspace

3 -- Confirm Oracle Database, SQL Developer, SQLPLUS are installed

## 4 -- Review the scripts provided with this assignment

## 5 -- Use the ASSIGN\_3 user. Create the user if it doesn’t exist.

|  |
| --- |
| **Embed all screenshots in the document where requested.**  For full marks screenshots must have formatted output and complete showing ONLY the information requested. No format script is provided. Include multiple screenshots and clear your screen using the **CLEAR SCREEN** command when needed. |

## Instructions

|  |
| --- |
| SQLPLUS and SQL Developer each have their own connection to the database. They cannot see each other’s changes without you running COMMIT. If either tool hangs, try running COMMIT in both tools. If hanging still occurs, close both tools and reopen them. |

|  |
| --- |
| In the conceptual data model there is a SectionNumber. This column was implemented as a number data type. The school now wants SectionNumber to be an alpha-numeric (containing both numbers and letters). You are asked to change SectionNumber to a VARCHAR2. |

1 -- Download and extract all files from the submission folder. Run CST2355\_A3\_DB.SQL to create your database.

You will create a new script that changes the data type for a table column without using the MODIFY command. Refer to <https://www.techonthenet.com/oracle/tables/alter_table.php> for examples of using the following statements.

Add an ALTER TABLE SECTION RENAME statement to rename the SectionNumber column to SectionNumber\_old.

Add an ALTER TABLE SECTION ADD statement to add a SectionNumber column with the data type VARCHAR (10).

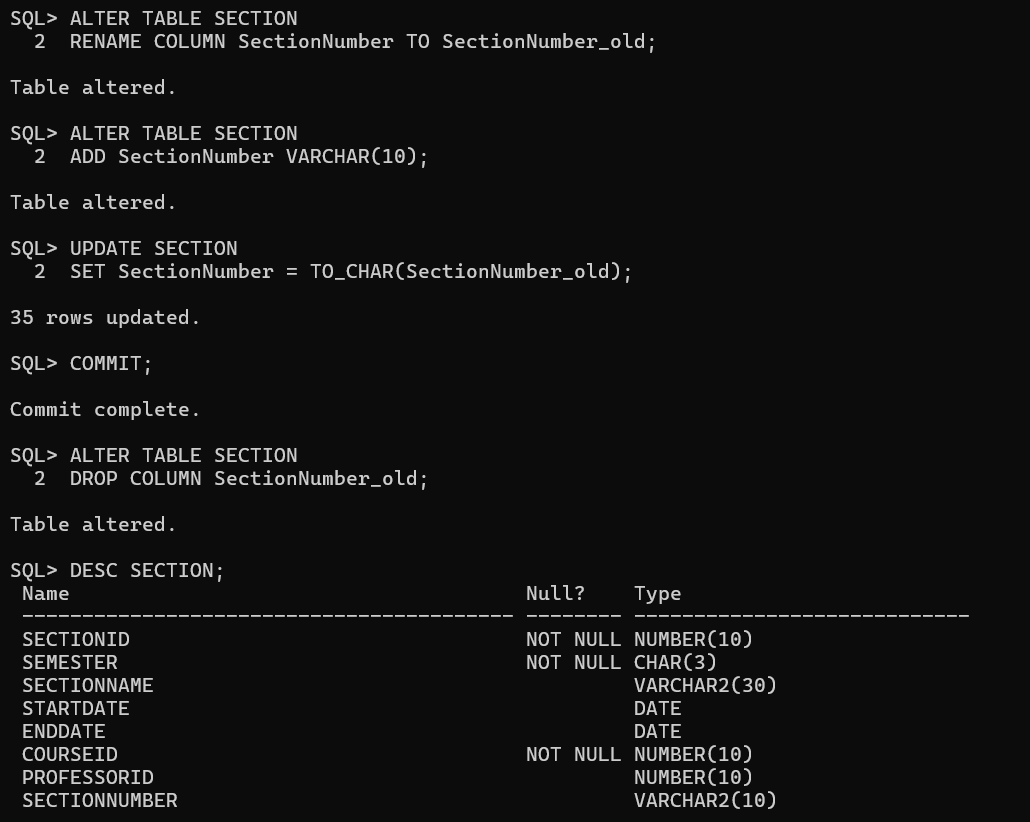
Add an UPDATE SECTION statement to copy the data from SectionNumber\_old to SectionNumber.

Add an ALTER TABLE SECTION DROP statement to remove the SectionNumber\_old column.

Add a DESC statement to show each column and data type for the SECTION table.

Add a comment line at the top of the script that includes your name, student number, and today’s date. Copy and paste your script into SQLPLUS to run it.

Provide screenshots from SQLPLUS of your script and each statement’s run results.



|  |
| --- |
| Often you will be asked to write code to reassign or renumber things in a database. Reassigning is when you re-associate an object. Professor Black is reassigned from Section 450 to Section 451. Renumbering is when Section 450 is renumbered or renamed to Section 451. You are asked to write a procedure to renumber sections for a specified course. |

2 -- Create a script that creates a new package and package body.

Add CREATE PACKAGE and CREATE PACKAGE BODY statements to create the SectionMaintenance package and package body. Refer to <https://www.tutorialspoint.com/plsql/plsql_packages.htm> for examples.

Add a stored procedure in the package and package body. Call it RenumberSectionNumber.

Add an IN\_COURSE input parameter to the procedure.

Add a NULL between the BEGIN and END for the procedure so you can compile the script.

PROCEDURE … AS

*BEGIN*

*NULL;*

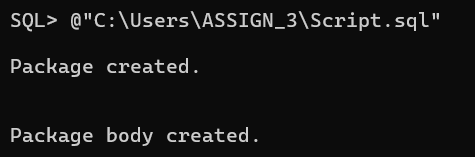
*END;*

*/*

End your script with a slash or **/** so Oracle compiles your code block.

Copy and paste your script into SQLPLUS to run your script.

Provide screenshots from SQLPLUS of your script code and its run results.



3 -- Update your script from step 2.

Declare a INT variable called NewSectionNumber. Refer to <https://www.oracletutorial.com/plsql-tutorial/plsql-variables/> for examples on using variables.

Declare a CURSOR variable called SectionList that returns the SectionID for a course with the CourseID = IN\_COURSE. Order the list according by SectionNumber. Refer to <https://www.techonthenet.com/oracle/cursors/declare.php> for examples on how to use cursors.

PROCEDURE … IS

*-- replace this with variable declarations*

*BEGIN*

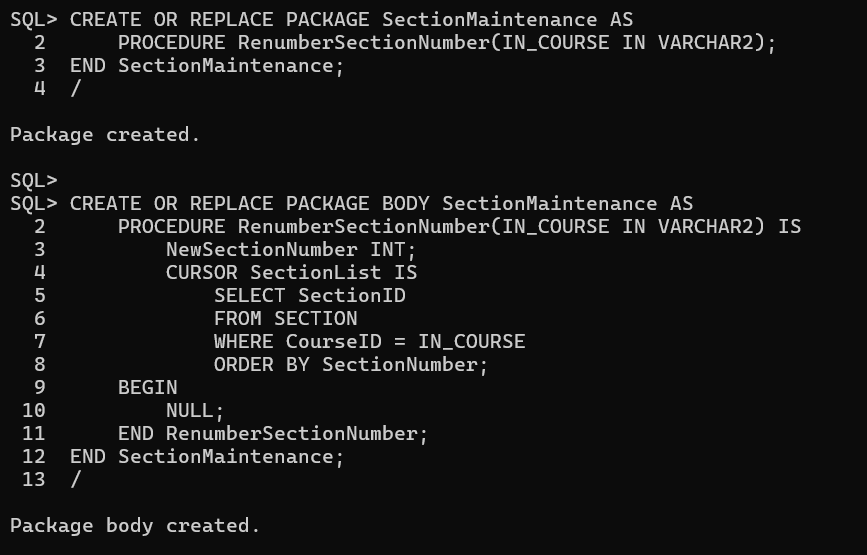
*NULL;*

*END;*

*/*

Copy and paste your script into SQLPLUS to run your script.

Provide screenshots from SQLPLUS of your script code and its run results.



4 --Update your script from step 3.

Remove the NULL

Add a statement that sets your NewSectionNumber to 450. Refer to <https://www.oracletutorial.com/plsql-tutorial/plsql-variables/> for examples.

Add a FOR LOOP to loop through the cursor. Refer to <https://www.oracletutorial.com/plsql-tutorial/plsql-cursor-for-loop/> for examples.

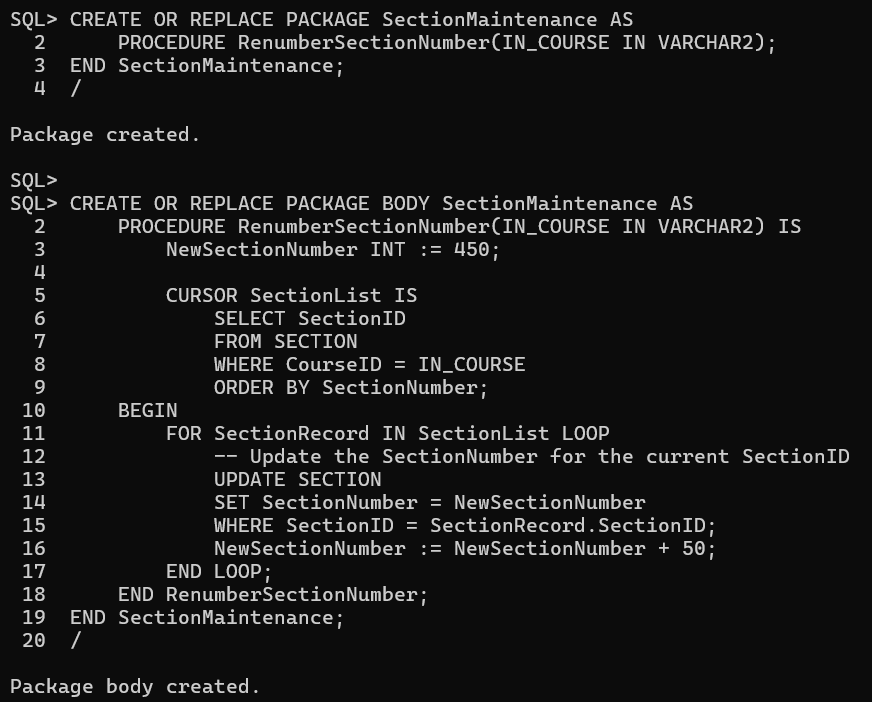
In the FOR LOOP add an UPDATE SECTION statement that updates one row to update the SECTION table with the new SectionNumber using the cursor’s SectionID.

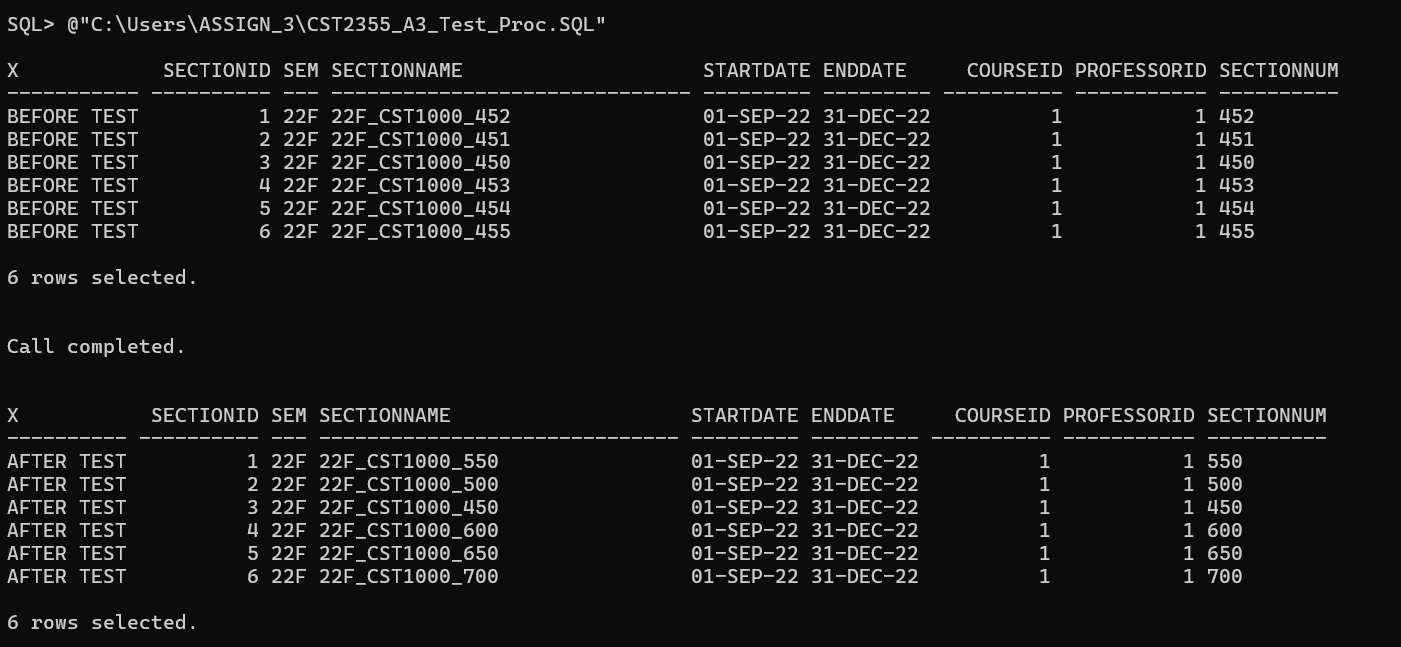
In the FOR LOOP, increment the NewSectionNumber by 50. (example: x = x + 50)

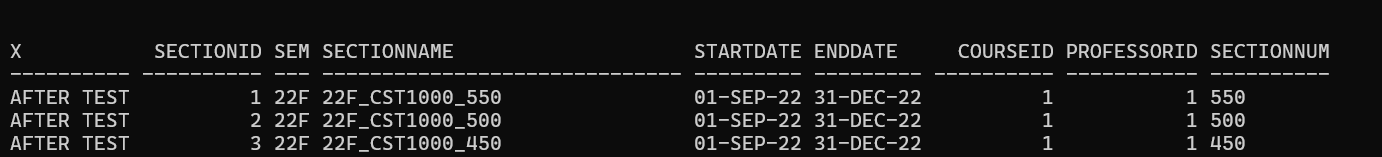
Add a comment line at the top of the script that includes your name, student number, and today’s date.

Copy and paste your script into SQLPLUS to run your script. Then run in SQLPLUS the CST2355\_A3\_Test\_Proc.SQL using the @ command to test your procedure. After the test, the first 3 rows should be SectionNumber 550, 500, 450 in that order.

Provide screenshots from SQLPLUS of your code script, its run results, and the test run results.







|  |
| --- |
| The benefits of a mandatory parent optional child is that no additional code needs to be added to enforce the relationship. It is the way databases work from out of the the box. However, the conceptual data model shows **PROFESSOR -- SECTION** are **optional parent - mandatory child** relationships. You will create a trigger that adds a new professor when the section is assigned a professor. |

5 -- You will create a new script to for an INSTEAD OF INSERT trigger.

Add a CREATE VIEW V\_SECTION statement using SELECT \* FROM SECTION.

Add a CREATE TRIGGER statement to create an INSTEAD OF INSERT trigger on V\_SECTION called SECTION\_INSERT. Set the trigger to be done for every row. Refer to

<https://www.oracletutorial.com/plsql-tutorial/oracle-instead-of-triggers/> for examples.

Add a NULL between the BEGIN and END for the trigger so you can compile the script.

*BEGIN*

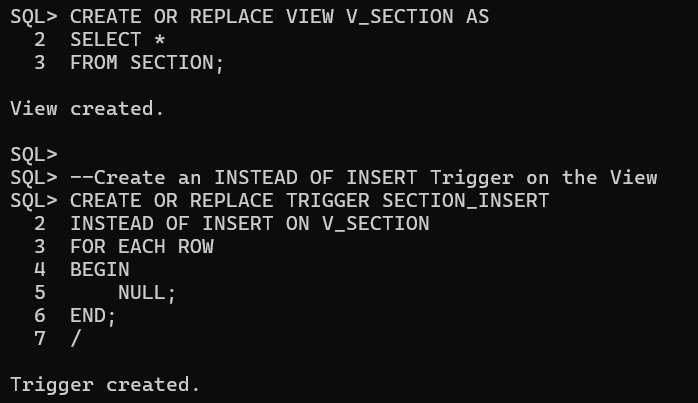
*NULL;*

*END;*

*/*

Copy and paste your script into SQLPLUS to run your script.

Provide screenshots from SQLPLUS of your script code and its run results.



6 -- Update your script from step 5.

Remove the NULL

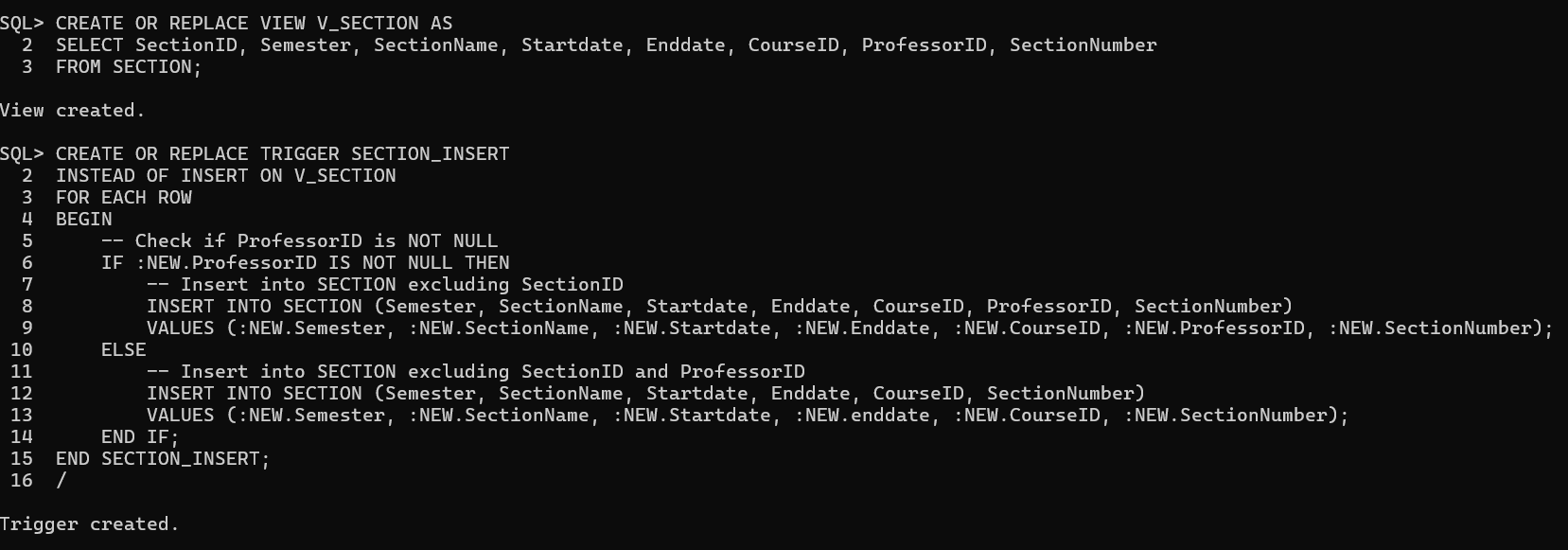
Add an IF… ELSE statement that executes when the **new** ProfessorID IS NOT NULL. Refer to <https://www.tutorialspoint.com/plsql/plsql_if_then_elsif.htm> for examples.

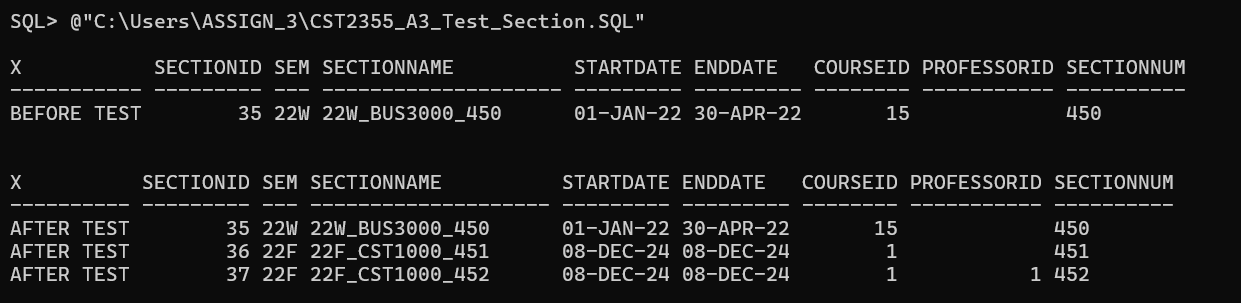
Under the IF part, add an INSERT INTO SECTION statement. The statement inserts each **new** V\_SECTION column value into SECTION except for the SectionID.

Under the ELSE part, add an INSERT INTO SECTION statement. The statement inserts each **new** V\_SECTION column value into SECTION except for the SectionID and ProfessorID.

Copy and paste your script into SQLPLUS to run your script. Then run in SQLPLUS the CST2355\_A3\_Test\_Section.SQL using the @ command to test your insert. After the test all columns for section 451 and 452 are inserted.

Provide screenshots from SQLPLUS of your script code, its run results, and the test run results.





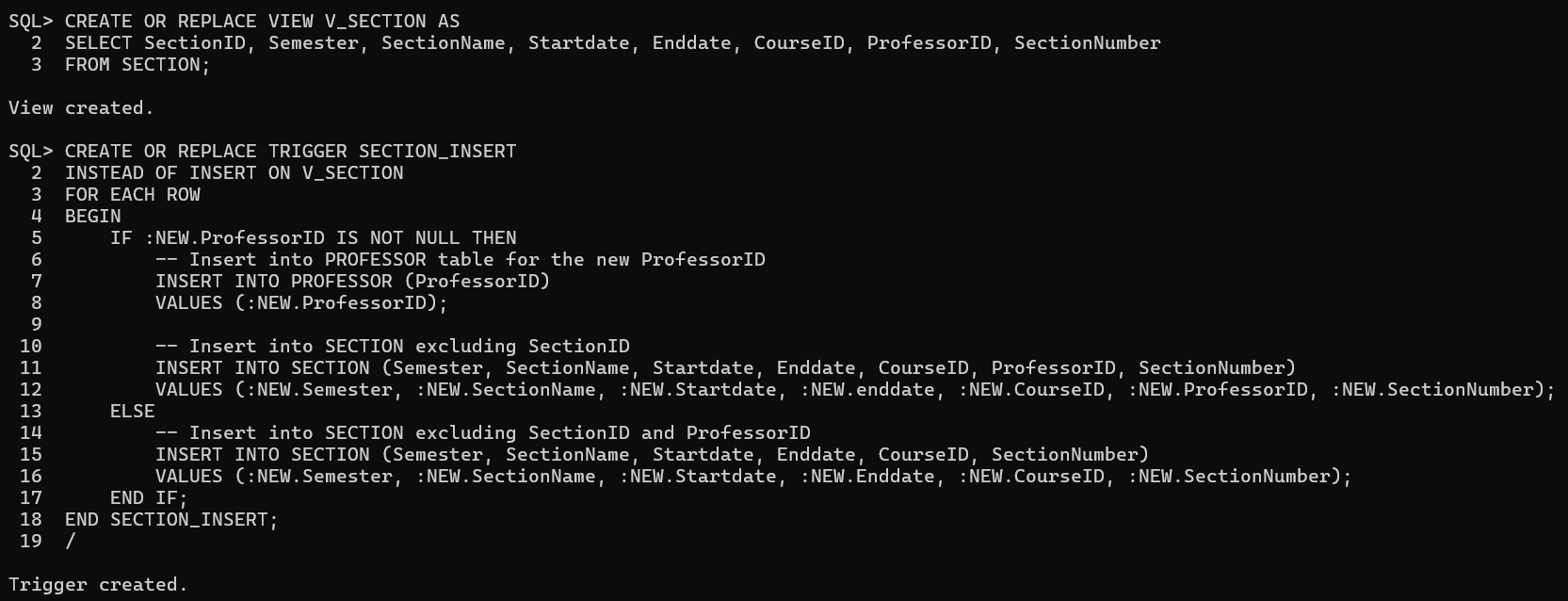
7 -- Update your script from step 6.

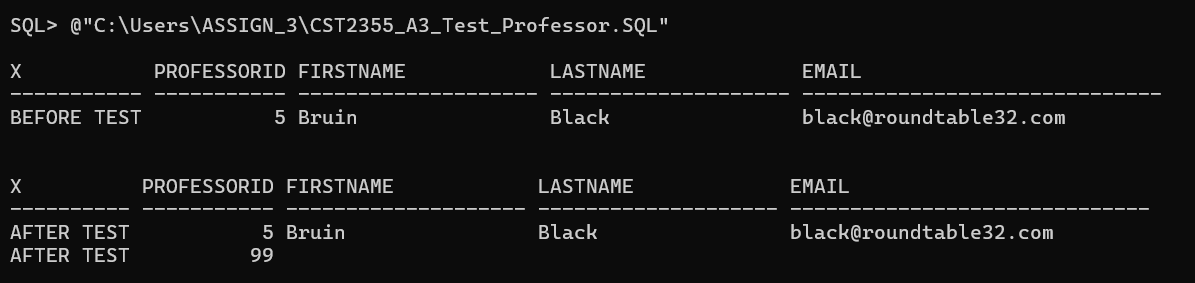
Within the IF statement and before the INSERT INTO SECTION add an INSERT INTO PROFESSOR statement. The statement inserts only the **new** V\_SECTION ProfessorID value into the ProfessorID in the PROFESSOR table. Refer to <https://www.tutorialspoint.com/plsql/plsql_if_then_elsif.htm> for examples.

Add a comment line at the top of the script that includes your name, student number, and today’s date.

Copy and paste your script into SQLPLUS to run your script. Then run in SQLPLUS the CST2355\_A3\_Test\_Professor.SQL using the @ command to test your insert. After the test a professor with ProfessorID equal to 99 should be inserted.

Provide screenshots from SQLPLUS of your script code, its run results, and the test run results.





|  |
| --- |
| Views are objects created on base tables. A view can be treated like a table in SELECT statements. You will create a view that shows how course, section, professor, and student information are all tied together. |

8 -- Open the CST2355\_A3 \_View.SQL script. Copy the CREATE VIEW portion of the script to create a new script.

Add the following columns -- SectionID, CourseID, ProfessorID, SectionName, CourseName, ProfessorFirstName, and ProfessorLastName. Add the inner joins on the tables COURSE, SECTION and PROFESSOR to get the data for these columns. Inner join PROFESSOR to SECTION. Do not remove pre-existing columns and tables.

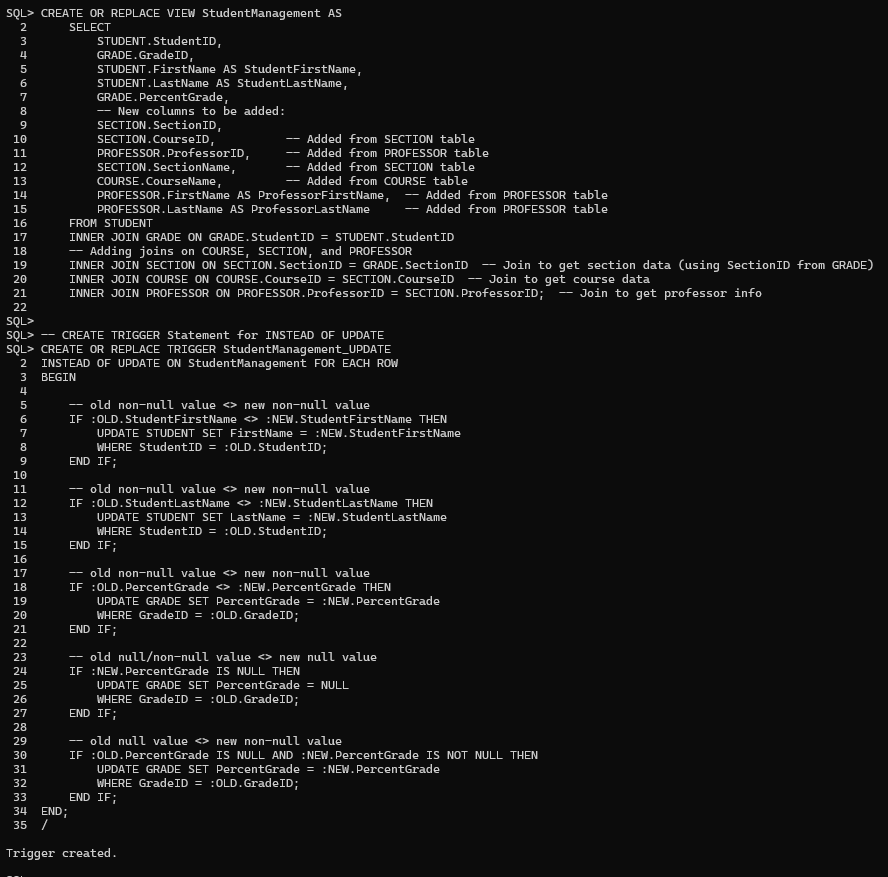
Add a comment line at the top of the script that includes your name, student number, and today’s date.

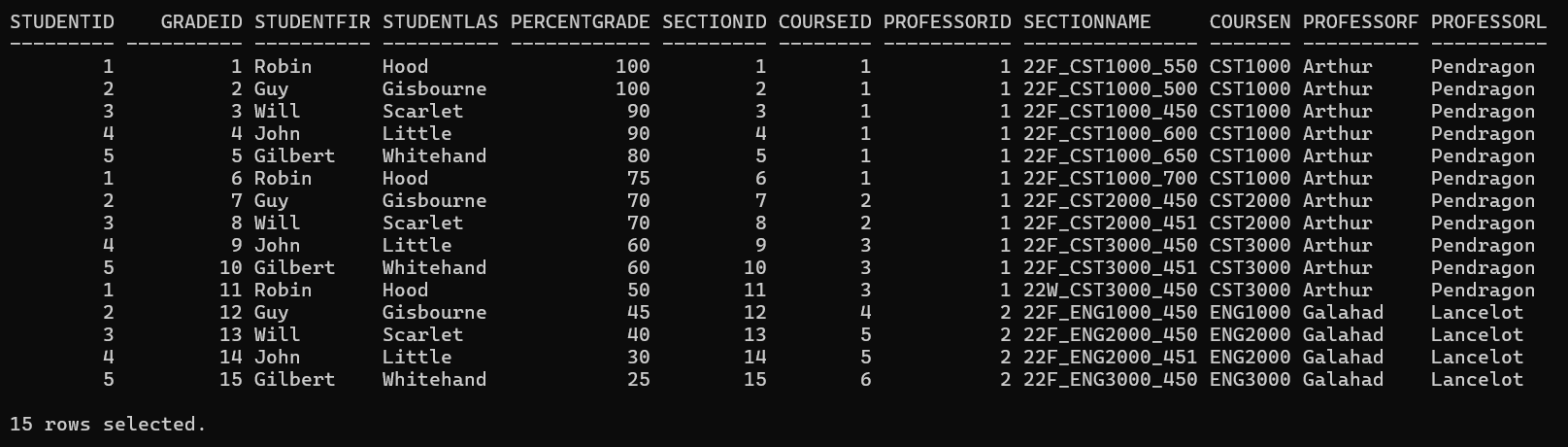
Copy and paste in SQLPLUS to run your script. Run @CST2355\_A3\_Test\_View.SQL to test your view. After the test you should have 15 rows.

If you receive an error creating your view, you will need to login using SQLPLUS / AS SYSDBA and run GRANT CREATE VIEW TO ASSIGN\_3;

Once you have permission, rerun your CREATE VIEW code.

Provide screenshots from SQLPLUS of your script and SELECT results.





|  |
| --- |
| You can use views an an interface for batch imports and for data entry. To do this with a view you need to make the view updatable. This is done using INSTEAD OF triggers. If in your view you make a change to a student’s FirstName, on its own your view doesn’t know where the FirstName is in the STUDENT table. So you specify in the trigger to locate the correct record in the STUDENT table. Then update that student’s FirstName in the STUDENT table using the UPDATE statement.  The college want to be able to quickly unassociate professors from sections when that professor is no longer available. |

9 -- Open the CST2355\_A3\_View.SQL script. Copy the TRIGGER portion of the script to create a new script.

Add an IF statement to the trigger that executes when both **new** ProfessorFirstName and **new** ProfessorLast are blank (NULL). Add this IF statement at the bottom of the trigger just before the END.

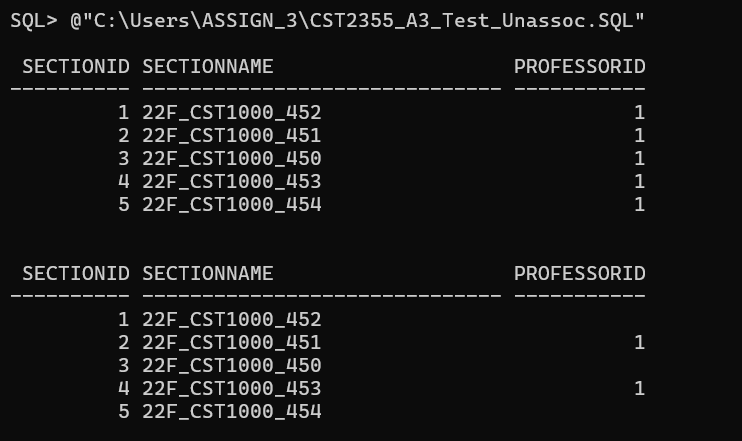
Under the IF statement add an UPDATE SECTION statement that unasociates the professor from the section. The unassociation is only done where the SectionID in SECTION is the same as the **old** SectionID for the trigger.

Add a comment line at the top of the script that includes your name, student number, and today’s date.

Copy and paste into In SQLPLUS to run the CREATE TRIGGER portion of your script. Run the CST2355\_A3\_Test\_Unasssoc.SQL using the @ command to test your trigger. After the test two sections should be assigned to the professor with ProfessorID = 1.

Provide screenshots from SQLPLUS of your script and test results.





|  |
| --- |
| It is common to use a function to store logic that will be used across various reports and queries. You will create a function when given a percentage it return a letter grade. Having a letter grade table will allow business users to easily change the ranges for letter grades at a later date. |

10 -- Create a new function called GradeLookup. Give it the input parameter IN\_Percentage with the type NUMBER. Have the function return the type VARCHAR. Refer to <https://www.tutorialspoint.com/plsql/plsql_functions.htm> for examples.

Declare a NewLetterGrade variable with the type VARCHAR (3).

Add an SELECT LetterGrade INTO GRADE\_LOOKUP statement to put the value of the LetterGrade into the NewLetterGrade variable. Refer to <https://www.oracletutorial.com/plsql-tutorial/plsql-select-into/> for examples.

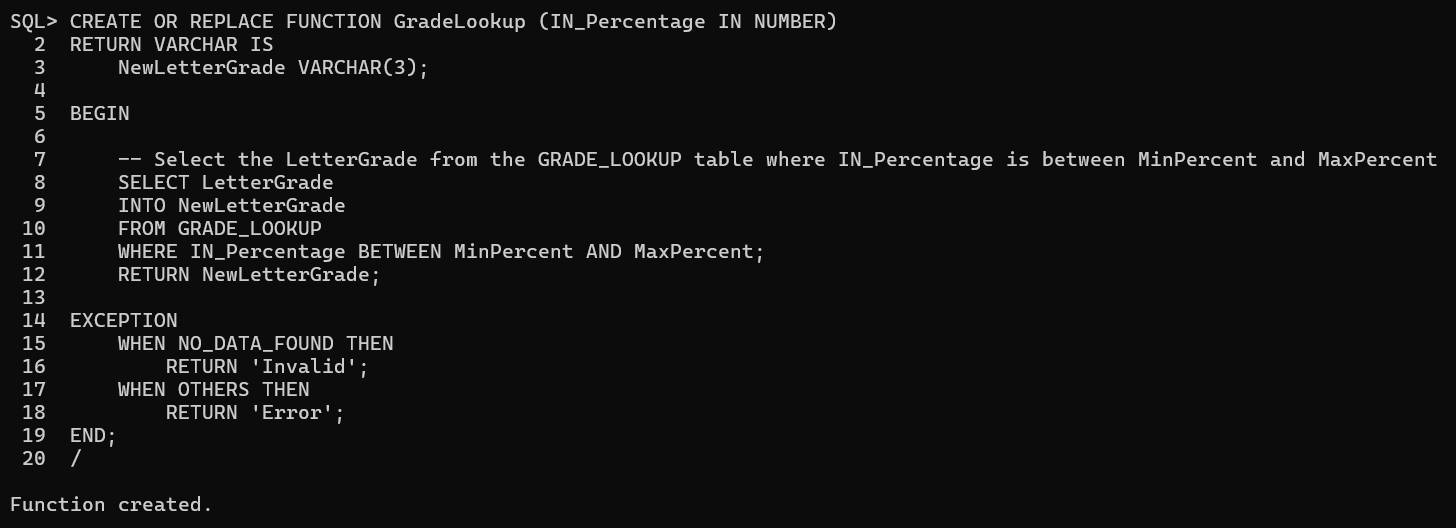
Add a WHERE to the SELECT so the statement returns the letter grade where IN\_Percentage is BETWEEN the MinPercent and MaxPercent values in the GRADE\_LOOKUP table. Refer to <https://www.oracletutorial.com/oracle-basics/oracle-between/> for examples.

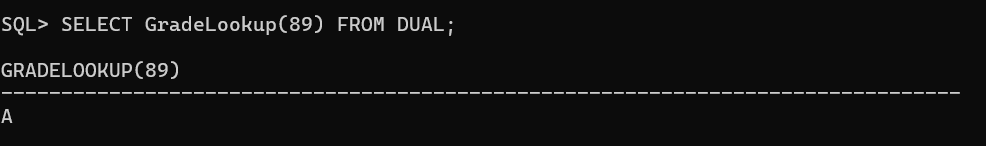
Add a RETURN statement to return NewLetterGrade.

Add a comment line at the top of the script that includes your name, student number, and today’s date.

Copy and paste Into SQLPLUS to run your script. Run SELECT GradeLookup(89) FROM DUAL to test your function.

Provide screenshots from SQLPLUS of your script and SELECT results.





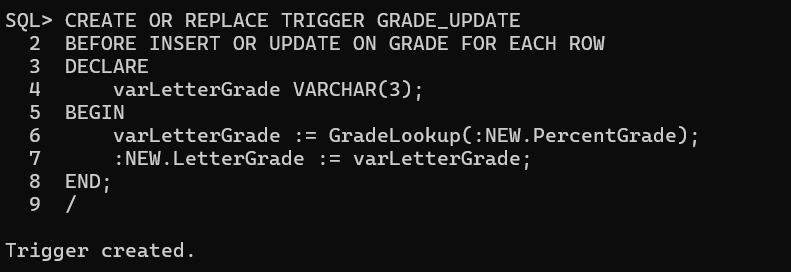
|  |
| --- |
| Anytime the PercentGrade is modified you want the corresponding LetterGrade stored in the GRADE table. |

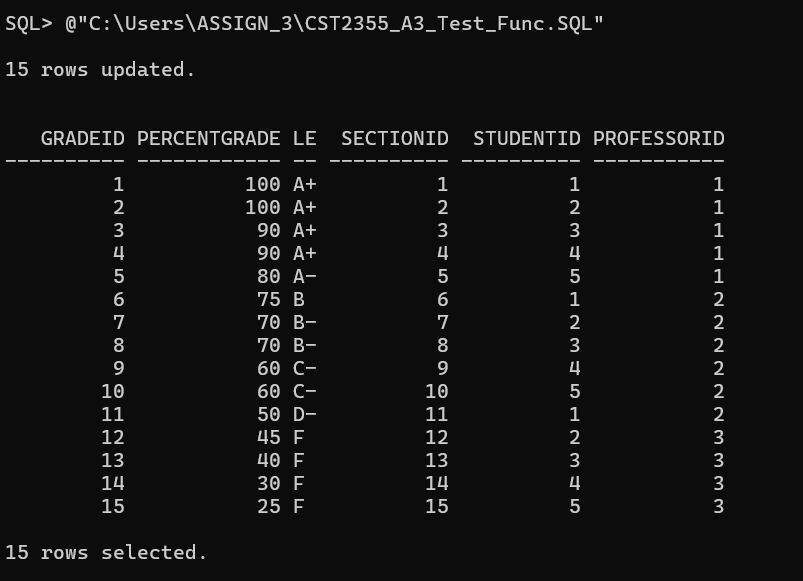
11 -- Open the CST2355\_A3\_Grade.SQL script. Modify the UPDATE and INSERT trigger so the **new** LetterGrade is updated using GradeLookup. For the input parameter for the function use **new** PercentageGrade.

Add a comment line at the top of the script that includes your name, student number, and today’s date.

Copy and paste into SQLPLUS to run your script. Run CST2355\_A3\_Test\_Func.SQL using @ command to test your trigger. After the test you should see a letter grades ranging from A+ to F.

Provide screenshots from SQLPLUS of your script and test results.





|  |
| --- |
| The ROLLUP option for GROUP BY allows you to store a total in the same column as the values that determine that total. This simplifies the formatting logic for reporting tools and web pages. |

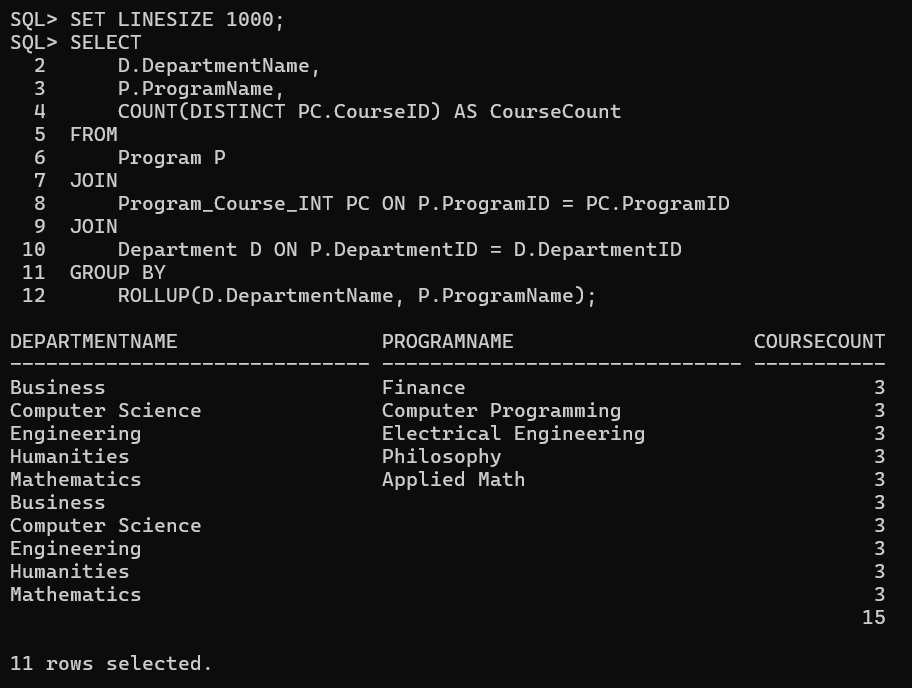
12 -- Open the CST2355\_A3\_Report.SQL script. Copy the ROLLUP portion of the script to create a new script. Modify your script so it produces the following output. Count IDs using COUNT(DISTINCT).

|  |  |  |
| --- | --- | --- |
| **DEPARTMENTNAME** | **PROGRAMNAME** | **COURSECOUNT** |
| Business | Finance | 3 |
| Computer Science | Computer Programming | 3 |
| Engineering | Electrical Engineering | 3 |
| Humanities | Philosphy | 3 |
| Mathematics | Applied Math | 3 |
| Business |  | 3 |
| Computer Science |  | 3 |
| Engineering |  | 3 |
| Humanities |  | 3 |
| Mathematics |  | 3 |
|  |  | 15 |

(the order of your rows may differ)

Add a comment line at the top of the script that includes your name, student number, and today’s date. On the next line after the comment add the command SET LINESIZE 200

In SQLPLUS copy/paste to run your script. Provide screenshots from SQLPLUS of your script and its results.



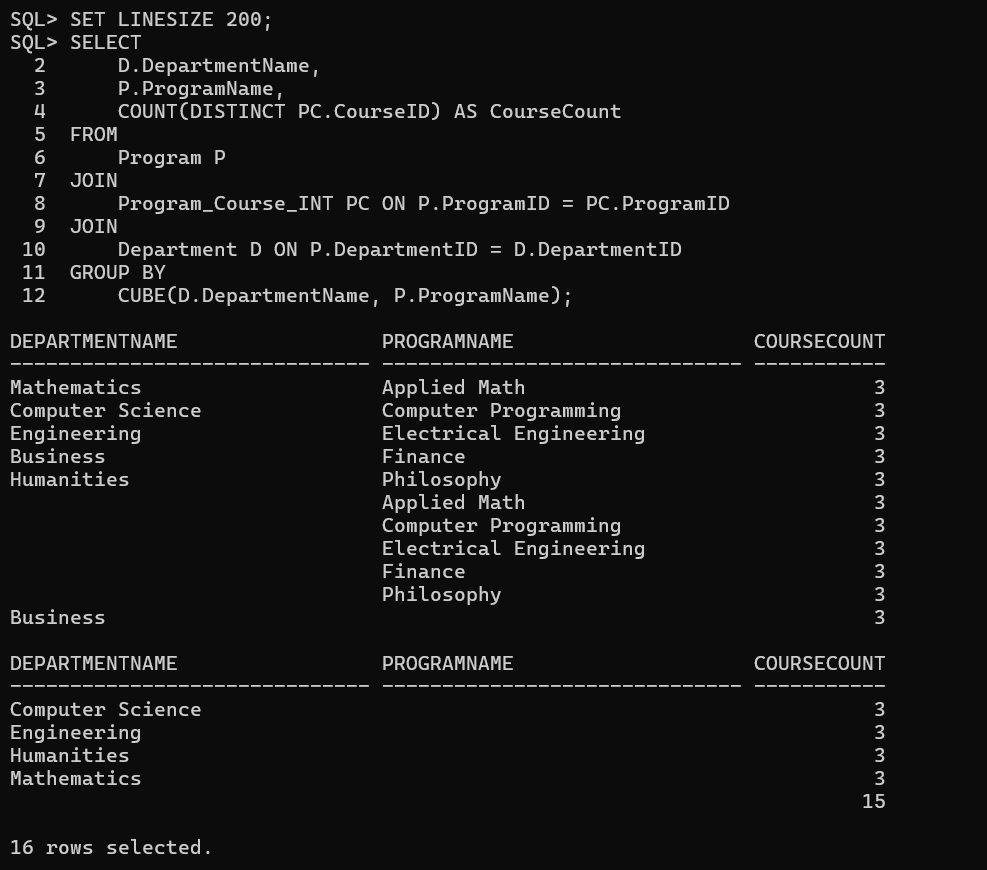
|  |
| --- |
| CUBE is used to retrieve multiple ROLLUPs within one query. |

13 -- Copy your script in step 12 and modify it so it produces the following output using a CUBE. Refer to <https://www.oracletutorial.com/oracle-basics/oracle-cube/> for examples.

In SQLPLUS copy/paste to run your script. Provide screenshots of your script and its results.

|  |  |  |
| --- | --- | --- |
| **DEPARTMENTNAME** | **PROGRAMNAME** | **COURSECOUNT** |
| Mathematics | Applied Math | 3 |
| Computer Science | Computer Programming | 3 |
| Engineering | Electrical Engineering | 3 |
| Business | Finance | 3 |
| Humanities | Philosophy | 3 |
|  | Applied Math | 3 |
|  | Computer Programming | 3 |
|  | Electrical Engineering | 3 |
|  | Finance | 3 |
|  | Philosophy | 3 |
| Business |  | 3 |
| Computer Science |  | 3 |
| Engineering |  | 3 |
| Humanities |  | 3 |
| Mathematics |  | 3 |
|  |  | 15 |

(the order of your rows may differ)



|  |
| --- |
| A PIVOT is where you take data that is by rows and convert it to show that data by columns. Pivoting query results is heavily used for reporting. It allows business users to compare numbers and see patterns. |

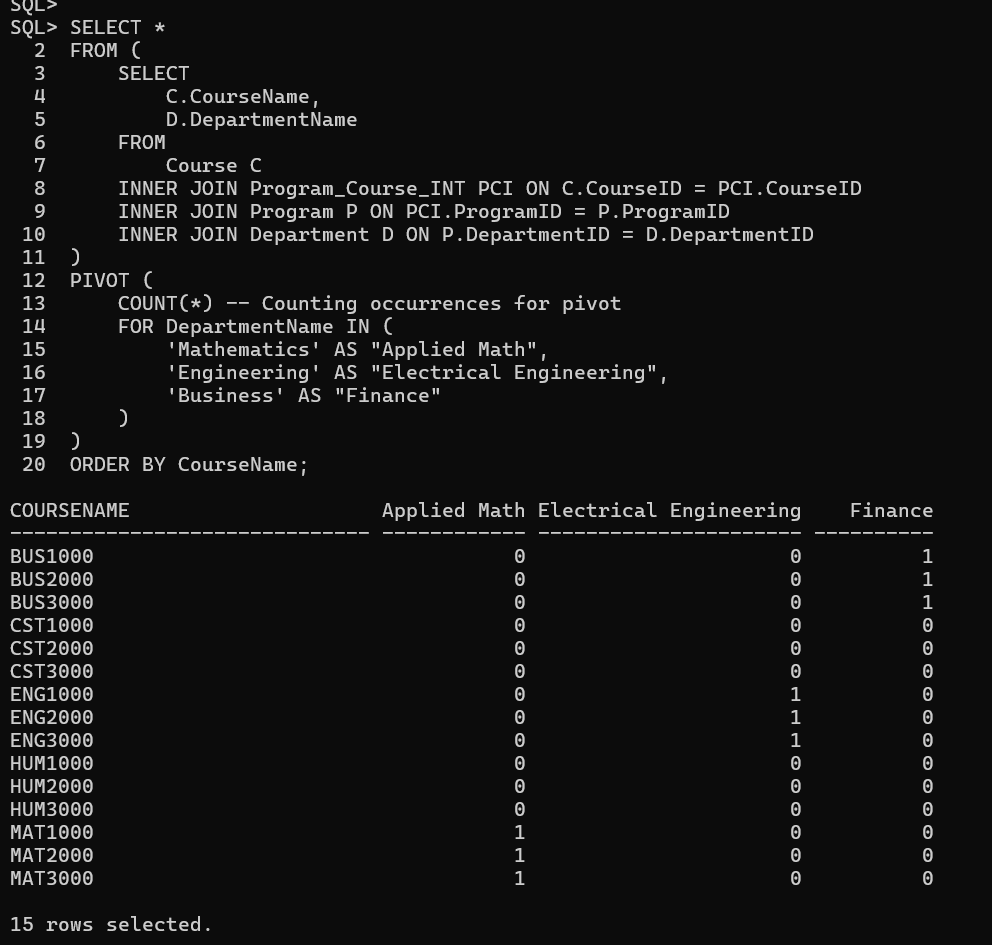
14 -- Open the CST2355\_A3\_Report.SQL script. Copy the PIVOT portion of the script to create a new script. Modify your script so it produces the following output. Reference [Oracle Pivot](https://www.oracletutorial.com/oracle-basics/oracle-pivot/) from OracleTutorial if needed. Count IDs using COUNT(DISTINCT).

|  |  |  |  |
| --- | --- | --- | --- |
| **COURSENAME** | **Applied Math** | **Electrical Engineering** | **Finance** |
| BUS1000 | 0 | 0 | 1 |
| BUS2000 | 0 | 0 | 1 |
| BUS3000 | 0 | 0 | 1 |
| CST1000 | 0 | 0 | 0 |
| CST2000 | 0 | 0 | 0 |
| CST3000 | 0 | 0 | 0 |
| ENG1000 | 0 | 1 | 0 |
| ENG2000 | 0 | 1 | 0 |
| ENG3000 | 0 | 1 | 0 |
| HUM1000 | 0 | 0 | 0 |
| HUM2000 | 0 | 0 | 0 |
| HUM3000 | 0 | 0 | 0 |
| MAT1000 | 1 | 0 | 0 |
| MAT2000 | 1 | 0 | 0 |
| MAT3000 | 1 | 0 | 0 |

(the order of your rows may differ)

Add a comment line at the top of the script that includes your name, student number, and today’s date. On the next line right after the comment add the command SET LINESIZE 200

In SQLPLUS copy/paste to run your script. Provide screenshots from SQLPLUS of your script and its results.



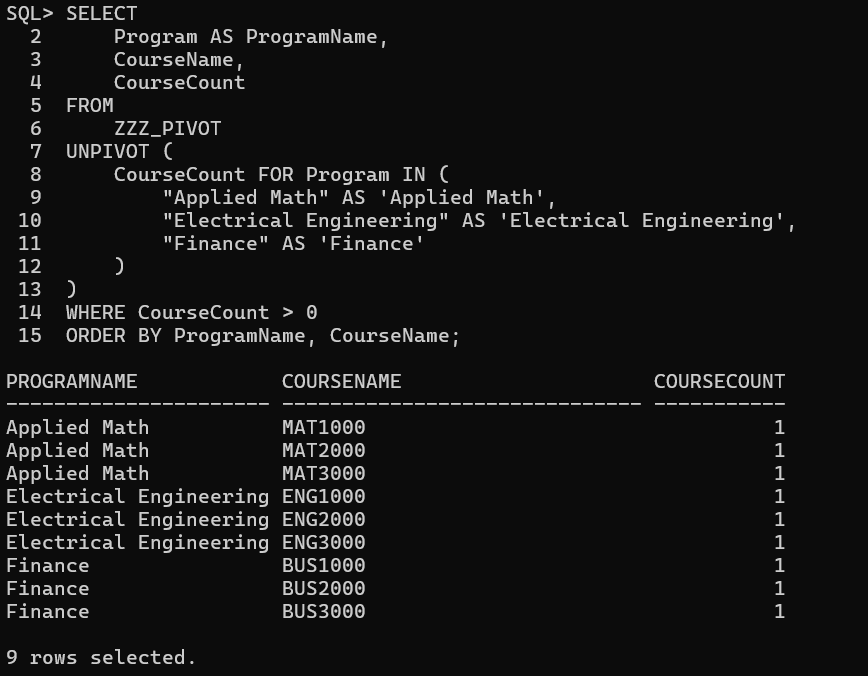
|  |
| --- |
| An UNPIVOT is where you take data that is by columns and convert it to show that data by rows. |

15 -- Create a table using your PIVOT script in step 14 using CREATE TABLE ZZZ\_PIVOT AS. Open CST2355\_A3\_Create\_Reports.SQL. Copy the UNPIVOT portion of the script and modify it so it produces the following output using an UNPIVOT. Reference [Oracle UnPivot](https://www.oracletutorial.com/oracle-basics/oracle-unpivot/) from OracleTutorial if needed.

|  |  |  |
| --- | --- | --- |
| **PROGRAM** | **COURSENAME** | **COURSECOUNT** |
| Finance | BUS1000 | 1 |
| Finance | BUS2000 | 1 |
| Finance | BUS3000 | 1 |
| Electrical Engineering | ENG1000 | 1 |
| Electrical Engineering | ENG2000 | 1 |
| Electrical Engineering | ENG3000 | 1 |
| Applied Math | MAT1000 | 1 |
| Applied Math | MAT2000 | 1 |
| Applied Math | MAT3000 | 1 |

(the order of your rows may differ)

Add a comment line at the top of the script that includes your name, student number, and today’s date. In SQLPLUS copy/paste to run your script. Provide screenshots from SQLPLUS of your script and its results.



|  |
| --- |
| Confirm all screenshots are correct and not clipped. Save your assignment as a PDF and submit it in the assignment submission folder. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment 3 Rubric** | | | |
| **Criteria** | **Accomplished** | **Developing** | **Incomplete** |
|  | **1 points** | **0.5 point** | **0 points** |
| 1 -- Data Type Change | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 2 -- Procedure Declaration | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 3 -- Parameter and Cursor Declaration | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 4 -- Cursor Loop and Increment | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 5 -- Trigger Declaration | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 6 -- Section Insert | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 7 -- Professor Insert | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 8 -- View | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 9 -- Unassociate Professor from Section | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 10 -- LetterGrade Function | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 11 -- GRADE LetterGrade Insert / Update | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 12 -- ROLLUP | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 13 -- CUBE | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 14 -- PIVOT | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |
| 15 -- UNPIVOT | Completed all instructions and submitted correct results | SQL follows most of the instructions and produced unexpected run results | Doesn't compile, SQL not per instructions, incomplete or omitted steps |