INF60014 Database Systems
Stored Procedures & Stored Functions
Major Assessment Task

Version 1.0.0

The assignment is to be done **individually**

Due Date/Time: **11:50pm Friday 7th of September, 2017**

Submission Requirements

You must submit your assignment via **Doubtfire**
https://doubtfire.ict.swin.edu.au/

This assignment requires only one submission.

- **The submission file must be called Ass1_SQLCode.sql**
- **Ensure that Ass1_SQLCode.sql contains all tasks that you have completed**
- **Submit Ass1_SQLCode.sql to MAT 1.1 P**

The SQL script **must** work with your Swinburne Oracle account.

***** This assignment is marked with an automated script. You must use the names specified in this assignment outline at all times. If you do not use the specified name you will not pass that part *****

Note:
The actual code writing within this assignment is not too difficult.
Each block of code contains a few lines of code and can be easily tested.

However there are a large number of these small blocks of code that need to be written and tested. This will take time.
Don't leave this work to the last couple of days before the due date, as you will most likely run out of time.

Background

You are to create tables to store product and customer data.

You will create a number of stored procedures and functions (SPFs) to insert / update / delete / query data.

These SPFs will be called from

- Additional stored procedures that can be executed from anonymous blocks via SQL Developer
- A host application written in VB or C#

Some SPFs may modify data in multiple rows in multiple tables
This will require you to demonstrate the use of handling database transactions.

Some SPFs may require data to be passed / returned using cursors

Requirements

Pass Task:

Parts 1,2, & 3 form the **PASS** task for this assignment

Credit Task:

Parts 4, 5, & 6 form the **CREDIT** task for this assignment.

Distinction Task:

Parts 7, 8, & 9 form the **DISTINCTION** task for this assignment.

Preparation

Begin by downloading the file named **ASS1_DDL.TXT** from Blackboard. Execute the contents of the file in SQL Developer.

******The database schema may not be altered in any way.******

Note: The first time that you run this script, you are likely to see errors for each of the Drop Table or Drop Sequence statements. It occurs because these objects do not yet exist in Oracle. If you run the script a second time, then those 'drop' errors should disappear.

Marking Guide

Part	Brief Description	Task
1	Develop and test SP/SFs to be executed within SQL Developer	Pass
2	Develop and test SP/SFs that use cursors executed within SQL Developer	Pass
3	Develop and test SP/SFs that utilise check constraints executed within SQL Developer	Pass
4	Develop and test SP/SFs that add and retrieve complex sale data and is executed within SQL Developer	Credit
5	Develop and test SP/SFs that deletes complex sale data and is executed within SQL Developer	Credit
6	Develop and test SP/SFs that Delete Customer and Delete Product executed within SQL Developer	Credit
7	Develop and test VB/C# code that calls SP/SFs from Part 1	Distinction
8	Develop and test VB/C# code to handle cursors from Part 2	Distinction
9	Develop and test VB/C# code to handle task 4,5,6 functionality	Distinction

NOTE: Tasks must be a minimum of 80% correct to be deemed complete.

TASK 1 (parts 1-3)

PART 1. Basic STORED PROCEDURES / FUNCTIONS and SQL DEVELOPER testing

Part 1.1. Create these stored procedures/function using SQL Developer

Name			**Type**	**Return Type**
ADD_CUST_TO_DB			Stored Procedure	None
Description	Add a new customer to Customer table.			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pcustname	Varchar2	Customer Name	
Requirements	Insert a new customer using parameter values. Set the SALES_YTD value to zero. Set the STATUS value to 'OK'			
Exceptions	**Type**		**Raise Application Error Details**	
	Duplicate primary key		-20010. Duplicate customer ID	
	pcustid outside range: 1-499		-20002. Customer ID out of range	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
ADD_CUSTOMER_VIASQLDEV			Stored Procedure	None
Description	Calls ADD_CUST_TO_DB			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pcustname	Varchar2	Customer Name	
Requirements	Display line '----------' using DOPL			
	Display parameter value in following format **before** Inserting row Adding Customer. ID: 999 Name: XXXXXXXXXXXXXXXXXXX			
	If row inserted successfully display "Customer Added OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Test Strategy.

Ensure the Customer Table has been created.
Ensure the Customer Table has no rows.
Run this anonymous block:

```
Begin
  ADD_CUSTOMER_VIASQLDEV(1,'Fred Smith');
  ADD_CUSTOMER_VIASQLDEV(2,'Sue Davis');
  ADD_CUSTOMER_VIASQLDEV(3,'Emma Jones');
  ADD_CUSTOMER_VIASQLDEV(1,'John Brown');
  ADD_CUSTOMER_VIASQLDEV(500,'Helen Nolan');
End;
```

Check that the correct messages are displayed in the Script Output Window

```
----------
Adding Customer 1 Fred Smith
Added OK
----------
Adding Customer 2 Sue Davis
Added OK
----------
Adding Customer 3 Emma Jones
Added OK
----------
Adding Customer 1 John Brown
ORA-20001: Error: Duplicate Customer ID
----------
Adding Customer 500 Helen Nolan
ORA-20002: Error: Customer ID out of range
```

If your code does not work correctly, then you may need to delete all rows from the customer table before running the anonymous script again.

Part 1.2. Create these stored procedures/functions using SQL Developer

Name			**Type**	**Return Type**
DELETE_ALL_CUSTOMERS_FROM_DB			Stored Function	Number
Description	Delete all customers from Customer table.			
Parameters	**Name**	**Type**	**Description**	
Requirements	Delete all customers from Customer table. Return the number of rows deleted			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
DELETE_ALL_CUSTOMERS_VIASQLDEV			Stored Procedure	None
Description	Calls DELETE_ALL_CUSTOMERS_FROM_DB.			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** deleting row using DOPL Display "Deleting all Customer rows" using DOPL			
	Display "999 rows deleted" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Test Strategy.

Run this anonymous block:

```
begin
DELETE_ALL_CUSTOMERS_VIASQLDEV;
ADD_CUSTOMER_VIASQLDEVELOPER(1,'Fred Smith');
ADD_CUSTOMER_VIASQLDEVELOPER(2,'Sue Davis');
ADD_CUSTOMER_VIASQLDEVELOPER(3,'Emma Jones');
end;
```

Run the block again.

If successful:

- you should not get messages about Duplicate Primary Keys
- three customer rows will be inserted into the table

Part 1.3. Create these stored procedures/functions using SQL Developer

Name			**Type**	**Return Type**
ADD_PROD_TO_DB			Stored Procedure	None
Description	Add a new product to Product table.			
Parameters	**Name**	**Type**	**Description**	
	pprodid	Number	Product Id	
	pprodname	Varchar2	Product Name	
	pprice	Number	Price	
Requirements	Insert a new product using parameter values. Set the SALES_YTD value to zero.			
Exceptions	**Type**		**Raise Application Error Details**	
	Duplicate primary key		-20010. Duplicate product ID	
	pprodid outside range: 1000 - 2500		-20012. Product ID out of range	
	pprice outside range: 0 – 999.99		-20013. Price out of range	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
ADD_PRODUCT_VIASQLDEV			Stored Procedure	None
Description	Calls ADD_PROD_TO_DB			
Parameters	**Name**	**Type**	**Description**	
	pprodid	Number	Product Id	
	pprodname	Varchar2	Product Name	
	pprice	Number	Price	
Requirements	Display line '----------' using DOPL			
	Display parameter value in following format **before** Inserting row Adding Product. ID: 9999 Name: XXXXXXXXXXXXXXXXXXX Price: 999.99			
	If row inserted successfully display "Product Added OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Name			**Type**	**Return Type**
DELETE_ALL_PRODUCTS_FROM_DB			Stored Function	Number
Description	Delete all products from Product table.			
Parameters	**Name**	**Type**	**Description**	
Requirements	Delete all products from Product table. Return the number of rows deleted			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
DELETE_ALL_PRODUCTS_VIASQLDEV			Stored Procedure	None
Description	Calls DELETE_ALL_PRODUCTS_FROM_DB.			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
Requirements	Display the following **before** deleting row using DOPL "Deleting all Product rows" using DOPL			
	Display "999 rows deleted" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Test Strategy.

Write an anonymous block to test these stored procedures/functions.

Part 1.4. Create these stored procedures/functions using SQL Developer

Name			**Type**	**Return Type**
GET_CUST_STRING_FROM_DB			Stored Function	Varchar2
Description	Get one customers details from customer table			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
Requirements	Return a single string using the format: Custid: 999 Name:XXXXXXXXXXXXXXXXXXXX Status XXXXXXX SalesYTD:99999.99			
Exceptions	**Type**		**Raise Application Error Details**	
	No matching customer id found		-20021. Customer ID not found	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
GET_CUST_STRING_VIASQLDEV			Stored Procedure	None
Description	Calls GET_CUST_STRING_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
Requirements	Display line '----------' using DOPL			
	Display the following **before** getting customer details using DOPL "Getting Details for CustId 999" using DOPL			
	Display the return value of GET_CUST_STRING_FROM_DB via DOPL			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Name			**Type**	**Return Type**
UPD_CUST_SALESYTD_IN_DB			Stored Procedure	None
Description	Update one customer's sales_ytd value in the customer table			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pamt	Number	Change Amount	
Requirements	Change one customer's SALES_YTD value by the pamt value.			
Exceptions	**Type**		**Raise Application Error Details**	
	No rows updated		-20021. Customer ID not found	
	pamt outside range: -999.99 to 999.99		-20032. Amount out of range	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
UPD_CUST_SALESYTD_VIASQLDEV			Stored Procedure	None
Description	Calls UPD_CUST_SALESYTD_IN_DB			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pamt	Number	Change Amount	
Requirements	Display line '----------' using DOPL			
	Display the following **before** updating row using DOPL "Updating SalesYTD. Customer Id: 999 Amount: 999.99" using DOPL			
	If row updated successfully display "Update OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Test Strategy.

Write an anonymous block to test these stored procedures/functions.

Part 1.5. Create these stored procedures/functions using SQL Developer

Name			**Type**	**Return Type**
GET_PROD_STRING_FROM_DB			Stored Function	Varchar2
Description	Get one products details from product table			
Parameters	**Name**	**Type**	**Description**	
	pprodid	Number	Product Id	
Requirements	Return a single string using the format: Prodid: 999 Name:XXXXXXXXXXXXXXXXXXXX Price 999.99 SalesYTD:99999.99			
Exceptions	**Type**		**Raise Application Error Details**	
	No matching product id found		-20041. Product ID not found	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
GET_PROD_STRING_VIASQLDEV			Stored Procedure	None
Description	Calls GET_PROD_STRING_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
	pprodid	Number	Product Id	
Requirements	Display line '----------' using DOPL			
	Display the following **before** getting product details using DOPL "Getting Details for Prod Id 999" using DOPL			
	Display the return value of GET_PROD_STRING_FROM_DB via DOPL			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Name			**Type**	**Return Type**
UPD_PROD_SALESYTD_IN_DB			Stored Procedure	None
Description	Update one product's sales_ytd value in the product table			
Parameters	**Name**	**Type**	**Description**	
	pprodid	Number	Product Id	
	pamt	Number	Change Amount	
Requirements	Change one product's SALES_YTD value by the pamt value.			
Exceptions	**Type**		**Raise Application Error Details**	
	No rows updated		-20041. Product ID not found	
	pamt outside range: -999.99 to 999.99		-20052. Amount out of range	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
UPD_PROD_SALESYTD_VIASQLDEV			Stored Procedure	None
Description	Calls UPD_PROD_SALESYTD_IN_DB			
Parameters	**Name**	**Type**	**Description**	
	pprodid	Number	Product Id	
	pamt	Number	Change Amount	
Requirements	Display line '----------' using DOPL			
	Display the following **before** updating row using DOPL "Updating SalesYTD Product Id: 999 Amount: 9999.99" using DOPL			
	If row updated successfully display "Update OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Test Strategy.

Write an anonymous block to test these stored procedures/functions.

Part 1.6. Create these stored procedures/functions using SQL Developer

Name			**Type**	**Return Type**
UPD_CUST_STATUS_IN_DB			Stored Procedure	None
Description	Update one customer's status value in the customer table			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pstatus	Varchar2	New status	
Requirements	Change one customer's status value.			
Exceptions	**Type**		**Raise Application Error Details**	
	No rows updated		-20061. Customer ID not found	
	Invalid status (not either OK or SUSPEND)		-20062. Invalid Status value	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
UPD_CUST_STATUS_VIASQLDEV			Stored Procedure	None
Description	Calls UPD_CUST_STATUS_IN_DB			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pstatus	Varchar2	New status	
Requirements	Display line '----------' using DOPL			
	Display the following **before** updating row using DOPL "Updating Status. Id: 999 New Status: XXXXXXX" using DOPL			
	If row updated successfully display "Update OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Test Strategy.

Write an anonymous block to test these stored procedures/functions.

Part 1.7. Create these stored procedures/functions using SQL Developer

Name			**Type**	**Return Type**
ADD_SIMPLE_SALE_TO_DB			Stored Procedure	None
Description	Update one customer's status value in the customer table			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pprodid	Number	Product Id	
	pqty	Number	Sale Qty	
Requirements	Check if customer status is 'OK'. If not raise an exception. Check if quantity value is valid. If not raise an exception. Update both the Customer and Product SalesYTD values Note: The YTD values must be increased by pqty * the product price Calls UPD_CUST_SALES_YTD_IN_DB and UPD_PROD_SALES_YTD_IN_DB			
Exceptions	**Type**		**Raise Application Error Details**	
	Sale Quantity range 1 - 999		-20071. Sale Quantity outside valid range	
	Invalid customer status (status is not 'OK')		-20072. Customer status is not OK	
	No matching customer id found		-20073. Customer ID not found	
	No matching product id found		-20076. Product ID not found	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
ADD_SIMPLE_SALE_VIASQLDEV			Stored Procedure	None
Description	Calls ADD_SIMPLE_SALE_TO_DB			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pprodid	Number	Product Id	
	pqty	Number	Sale Qty	
Requirements	Display line '----------' using DOPL			
	Display the following **before** adding sale row using DOPL "Adding Simple Sale. Cust Id: 999 Prod Id 9999 Qty: 999 using DOPL			
	If row updated successfully display "Added Simple Sale OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm **Ensure that if any exception is raised that no data in the database is modified.**	

Test Strategy.

Write an anonymous block to test these stored procedures/functions.

Part 1.8. Create these stored procedures/functions using SQL Developer

Name			**Type**	**Return Type**
SUM_CUST_SALESYTD			Stored Function	Number
Description	Sum and return the SalesYTD value of all rows in the Customer table			
Parameters	**Name**	**Type**	**Description**	
Requirements	Sum and return the SalesYTD value of all rows in the Customer table			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
SUM_CUST_SALES_VIASQLDEV			Stored Procedure	None
Description	Calls SUM_CUST_SALESYTD			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** calculation using DOPL "Summing Customer SalesYTD using DOPL			
	If successful display in this format "All Customer Total: 99999.99" via DOPL			
Exceptions	**Type**		**Action**	
	NO ROWS FOUND		**Ignore Error. Display 0 total**	
	Other		use DOPL to show value of sqlerrm	

Name			**Type**	**Return Type**
SUM_PROD_SALESYTD_FROM_DB			Stored Function	Number
Description	Sum and return the SalesYTD value of all rows in the Product table			
Parameters	**Name**	**Type**	**Description**	
Requirements	Sum and return the SalesYTD value of all rows in the Product table			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
SUM_PROD_SALES_VIASQLDEV			Stored Procedure	None
Description	Calls SUM_PROD_SALESYTD_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** calculation using DOPL "Summing Product SalesYTD using DOPL			
	If successful display in this format "All Product Total: 99999.99"			
Exceptions	**Type**		**Action**	
	NO ROWS FOUND		**Ignore Error. Display 0 total**	
	Other		use DOPL to show value of sqlerrm	

Test Strategy.

Write an anonymous block to test these stored procedures/functions.

Part 1.9.

Copy and paste the stored procedure/function code from the tasks 1.1 to 1.8 into a file name **Ass1_SQLCode.sql**.

- Complete the student information at the top of the file
- ******Ensure that a / character appears on a single between each of the stored procedures and functions******

Part 1.10.

Modify the student number in line 2 below and then execute the following anonymous block:

```
begin
dbms_output.put_line('Student ID: 1234567');
DELETE_ALL_CUSTOMERS_VIASQLDEV;
DELETE_ALL_PRODUCTS_VIASQLDEV;
dbms_output.put_line('==========TEST ADD CUSTOMERS ==========================');
ADD_CUSTOMER_VIASQLDEV(1,'Colin Smith');
ADD_CUSTOMER_VIASQLDEV(2,'Jill Davis');
ADD_CUSTOMER_VIASQLDEV(3,'Dave Brown');
ADD_CUSTOMER_VIASQLDEV(4,'Kirsty Glass');
ADD_CUSTOMER_VIASQLDEV(1,'Jenny Nighy');
ADD_CUSTOMER_VIASQLDEV(-3,'Emma Jones');
ADD_CUSTOMER_VIASQLDEV(666,'Peter White');
dbms_output.put_line('==========TEST ADD PRODUCTS=========================');
ADD_PRODUCT_VIASQLDEV(1001,'ProdA', 10);
ADD_PRODUCT_VIASQLDEV(1002,'ProdB', 20);
ADD_PRODUCT_VIASQLDEV(1003,'ProdC', 35);
ADD_PRODUCT_VIASQLDEV(1001,'ProdD', 10);
ADD_PRODUCT_VIASQLDEV(3333,'ProdD', 100);
ADD_PRODUCT_VIASQLDEV(1004,'ProdD', 1234);
dbms_output.put_line('===========TEST STATUS UPDATES ==========================');
UPD_CUST_STATUS_VIASQLDEV(3,'SUSPEND');
UPD_CUST_STATUS_VIASQLDEV(4,'QWERTY');
dbms_output.put_line('===========TEST CUSTOMER RETREIVAL ==========================');
GET_CUST_STRING_VIASQLDEV(1);
GET_CUST_STRING_VIASQLDEV(2);
GET_CUST_STRING_VIASQLDEV(22);
dbms_output.put_line('===========TEST CUSTOMER RETREIVAL ==========================');
GET_PROD_STRING_VIASQLDEV(1001);
GET_PROD_STRING_VIASQLDEV(1002);
GET_PROD_STRING_VIASQLDEV(2222);
dbms_output.put_line('===========TEST SIMPLE SALES ==========================');
ADD_SIMPLE_SALE_VIASQLDEV(1,1001,15);
ADD_SIMPLE_SALE_VIASQLDEV(2,1002,37);
ADD_SIMPLE_SALE_VIASQLDEV(3,1002,15);
ADD_SIMPLE_SALE_VIASQLDEV(4,1001,100);
SUM_CUST_SALES_VIASQLDEV;
SUM_PROD_SALES_VIASQLDEV;
dbms_output.put_line('===========MORE TESTING OF SIMPLE SALES ==========================');
ADD_SIMPLE_SALE_VIASQLDEV(99,1002,60);
ADD_SIMPLE_SALE_VIASQLDEV(2,5555,60);
ADD_SIMPLE_SALE_VIASQLDEV(1,1002,6666);
SUM_CUST_SALES_VIASQLDEV;
SUM_PROD_SALES_VIASQLDEV;
dbms_output.put_line('==========LIST ALL CUSTOMERS AND PRODUCTS=========================');
GET_CUST_STRING_VIASQLDEV(1);
GET_CUST_STRING_VIASQLDEV(2);
GET_CUST_STRING_VIASQLDEV(3);
GET_CUST_STRING_VIASQLDEV(4);
GET_PROD_STRING_VIASQLDEV(1001);
GET_PROD_STRING_VIASQLDEV(1002);
GET_PROD_STRING_VIASQLDEV(1003);
end;
```

Copy and paste the output generated by this script into a file named Ass1_Output.TXT. This file is used for debugging purposes only. It does not have to be submitted.

PART 2. Cursors and SQL DEVELOPER testing

Part 2.1. Create these stored procedures/function using SQL Developer

Name			**Type**	**Return Type**
GET_ALLCUST			Stored Function	SYS_REFCURSOR
Description	Get all customer details and return as a SYS_REFCURSOR			
Parameters	**Name**	**Type**	**Description**	
Requirements	Get all customer details and return as a SYS_REFCURSOR			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
GET_ALLCUST_VIASQLDEV			Stored Procedure	None
Description	Calls GET_ALLCUST			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** listing any rows using DOPL "Listing All Customer Details			
	Display each customer using the following format via DOPL Custid: 999 Name:XXXXXXXXXXXXXXXXXXXX Status XXXXXXX SalesYTD:99999.99 If no customers exist, then display No rows found.			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Name			**Type**	**Return Type**
GET_ALLPROD_FROM_DB			Stored Function	SYS_REFCURSOR
Description	Get all product details and return as a SYS_REFCURSOR			
Parameters	**Name**	**Type**	**Description**	
Requirements	Get all product details and return as a SYS_REFCURSOR			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
GET_ALLPROD_VIASQLDEV			Stored Procedure	None
Description	Calls GET_ALLPROD_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** listing any rows using DOPL "Listing All Product Details			
	Display each product using the following format via DOPL Prodid: 999 Name:XXXXXXXXXXXXXXXXXXXX Price 999.99 SalesYTD:99999.99 If no products exist, then display No rows found.			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Part 2.2.

Copy and paste the stored procedure/function code above into the file named **Ass1_SQLCode.sql**

Part 2.3.

Execute the following block of code in SQL Developer. (Change the student id on line 2)

```
begin
dbms_output.put_line('Student ID: 1234567');
dbms_output.put_line('==========PART 2 TEST CURSOR=========================');
GET_ALLCUST_VIASQLDEV;
GET_ALLPROD_VIASQLDEV;
end;
```

Append the output generated by the above block into a file named **Ass1_Output.TXT**

PART 3. Check Constraints and SQL DEVELOPER testing

Part 3.1. Create these stored procedures/function using SQL Developer

Name			**Type**	**Return Type**
ADD_LOCATION_TO_DB			Stored Procedure	
Description	Adds a new row to the location table			
Parameters	**Name**	**Type**	**Description**	
	ploccode	varchar2	Location Code	
	pminqty	Number	Min qty	
	pmaxqty	Number	Max qty	
Requirements	Add a new row to the location table			
Exceptions	**Type**		**Raise Application Error Details**	
	Duplicate primary key		-20081. Duplicate location ID	
	CHECK_LOCID_LENGTH check failed		-20082. Location Code length invalid	
	CHECK_MINQTY_RANGE check failed		-20083. Minimum Qty out of range	
	CHECK_MAXQTY_RANGE check failed		-20084. Maximum Qty out of range	
	CHECK_MAXQTY_GREATER_MIXQTY check failed		-20086. Minimum Qty larger than Maximum Qty	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
ADD_LOCATION_VIASQLDEV			Stored Procedure	None
Description	Calls ADD_LOCATION_TO_DB			
Parameters	**Name**	**Type**	**Description**	
	ploccode	varchar2	Location Code	
	pminqty	Number	Min qty	
	pmaxqty	Number	Max qty	
Requirements	Display line '----------' using DOPL			
	Display the following **before** deleting row using DOPL "Adding Location LocCode: XXXXX MinQty: 9999 MaxQty: 9999			
	If row inserted successfully display "Location Added OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Please note that the location table is a 'stand alone' table.
The location table currently has no connection to any other table within the database.

Part 3.2.

Copy and paste the stored procedure/function code above into the file named **Ass1_ Code.sql**

Part 3.3.

Execute the following block of code in SQL Developer. (Change the student id on line 2)

```
begin
dbms_output.put_line('Student ID: 1234567');
dbms_output.put_line('==========PART 3 TEST LOCATIONS==========================');
ADD_LOCATION_VIASQLDEV ('AF201',1,2);
ADD_LOCATION_VIASQLDEV ('AF202',-3,4);
ADD_LOCATION_VIASQLDEV ('AF203',5,1);
ADD_LOCATION_VIASQLDEV ('AF204',6,7000);
ADD_LOCATION_VIASQLDEV ('AF20111',8,9);
end;
```

Append the output generated by the above block into a file named **Ass1_Output.TXT**

TASK 2 (parts 4-6)

PART 4. Complex Sale in SQL Developer

Part 4.1. Create these stored procedures/function using SQL Developer

Name			**Type**	**Return Type**
ADD_COMPLEX_SALE_TO_DB			Stored Procedure	None
Description	Adds a complex sale to the database			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pprodid	Number	Product Id	
	pqty	Number	Sale Qty	
	pdate	Varchar2	Sale Date format yyyymmdd	
Requirements	Check if customer status is 'OK'. If not raise an exception. Check if quantity value is valid. If not raise an exception. Check if date value is valid. If not raise an exception. Insert a new row into the Sale table. The saleid value must be obtained from the SALE_SEQ Update both the Customer and Product SalesYTD values Note: The YTD values must be increased by pqty * the unit price Calls UPD_CUST_SALES_YTD_IN_DB and UPD_PROD_SALES_YTD_IN_DB			
Exceptions	**Type**		**Raise Application Error Details**	
	Sale Quantity range 1 - 999		-20091. Sale Quantity outside valid range	
	Invalid customer status (status is not 'OK')		-20092. Customer status is not OK	
	Invalid sale date		-20093. Date not valid	
	No matching customer id found		-20094. Customer ID not found	
	No matching product id found		-20095. Product ID not found	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
ADD_COMPLEX_SALE_VIASQLDEV			Stored Procedure	None
Description	Calls ADD COMPLEX_SALE_TO_DB			
Parameters	**Name**	**Type**	**Description**	
	pcustid	Number	Customer Id	
	pprodid	Number	Product Id	
	pqty	Number	Sale Qty	
	pdate	Varchar2	Sale Date format yyyymmdd	
Requirements	Display line '----------' using DOPL			
	Display the following **before** adding sale row using DOPL "Adding Complex Sale. Cust Id: 999 Prod Id 9999 Date: yyyymmdd Amt: 999 using DOPL Note: The amount in the line above is pqty * product price			
	If row updated successfully display "Added Complex Sale OK" via DOPL			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm **Ensure that if any exception is raised that no data in the database is modified.**	

Name			**Type**	**Return Type**
GET_ALLSALES_FROM_DB			Stored Function	SYS_REFCURSOR
Description	Get all customer details and return as a SYS_REFCURSOR			
Parameters	**Name**	**Type**	**Description**	
Requirements	Get all complex sale details and return as a SYS_REFCURSOR			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			Type	Return Type
GET_ALLSALES_VIASQLDEV			Stored Procedure	None
Description	Calls GET_ALLSALES_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** listing any rows using DOPL "Listing All Complex Sales Details			
	Display each complex sale using the following format via DOPL Saleid: 9999 Custid: 999 Prodid: 9999 Date 31 DEC 2000 Amount: 9999.99 If no sales exist, then display No rows found.			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Name			Type	Return Type
COUNT_PRODUCT_SALES_FROM_DB			Stored Function	Number
Description	Count and return the number of sales with nn days of current date			
Parameters	**Name**	**Type**	**Description**	
	pdays	number	Count sales made within pdays of today's date	
Requirements	Count and return the number of sales in the SALES table with ***nn days*** of current date			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			Type	Return Type
COUNT_PRODUCT_SALES_VIASQLDEV			Stored Procedure	None
Description	Calls COUNT_PRODUCT_SALES_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
	pdays	number	Count sales made within pdays of today's date	
Requirements	Display line '----------' using DOPL			
	Display the following **before** calculation using DOPL "Counting sales within ***nn days***" using DOPL			
	If successful display in this format "Total number of sales: 999" via DOPL			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Part 4.2.

Add the statement **DELETE FROM SALE;** to the top of the section named "TEST DELETION OF EXISTING DATA"

Add the following statements to TEST PART 4 section of the script

```
ADD_CUSTOMER_VIASQLDEV(10,'Mieko Hayashi');
ADD_CUSTOMER_VIASQLDEV(11,'John Kalia');
ADD_CUSTOMER_VIASQLDEV(12,'Alex Kim');
ADD_PRODUCT_VIASQLDEV(2001,'Chair', 10);
ADD_PRODUCT_VIASQLDEV(2002,'Table', 45);
ADD_PRODUCT_VIASQLDEV(2003,'Lamp', 22);
ADD_COMPLEX_SALE_VIASQLDEV (10,2001,6,'20140301');
ADD_COMPLEX_SALE_VIASQLDEV (10,2002,1,'20140320');
ADD_COMPLEX_SALE_VIASQLDEV (11,2001,1,'20140301');
ADD_COMPLEX_SALE_VIASQLDEV (11,2003,2,'20140215');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,10,'20140131');
COUNT_PRODUCT_SALES_VIASQLDEV( sysdate-to_date('01-Jan-2014'));
COUNT_PRODUCT_SALES_VIASQLDEV( sysdate-to_date('01-Feb-2014'));
GET_ALLSALES_VIASQLDEV;
ADD_COMPLEX_SALE_VIASQLDEV (99,2001,10,'20140131');
ADD_COMPLEX_SALE_VIASQLDEV (12,9999,10,'20140131');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,9999,'20140131');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,10,'99999999');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,10, '20141331');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,10,'20140132');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,10, '20140');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,10,'201401311');
UPD_CUST_STATUS_VIASQLDEV(12,'SUSPEND');
ADD_COMPLEX_SALE_VIASQLDEV (12,2002,10,'20140131');
```

PART 5. Delete Complex Sale in SQL Developer

Part 5.1. Create these stored procedures/function using SQL Developer

Name			**Type**	**Return Type**
DELETE_SALE_FROM_DB			Stored Function	Number
Description	Delete a row from the SALE table			
Parameters	**Name**	**Type**	**Description**	
Requirements	Determine the smallest saleid value in the SALE table. (use Select MIN()…) If the value is NULL raise a No Sale Rows Found exception. Otherwise delete a row from the SALE table with the matching sale id Calls UPD_CUST_SALES_YTD_IN_DB and UPD_PROD_SALES_YTD_IN_DB so that the correct amount is subtracted from SALES_YTD. You must calculate the amount using the PRICE in the SALE table multiplied by the QTY This function must return the SaleID value of the Sale row that was deleted. *(It is a bit unrealistic to delete a row with the smallest saleid. Normally you would ask a user to enter a sale id value. However this is difficult to do when testing with an anonymous block. So we will settle for smallest saleid in this assignment).*			
Exceptions	**Type**		**Raise Application Error Details**	
	No Sale Rows Found		-20101. No Sale Rows Found	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
DELETE_SALE_VIASQLDEV			Stored Procedure	None
Description	Calls DELETE_SALE_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** deleting the sale using DOPL "Deleting Sale with smallest SaleId value" using DOPL			
	If successful display in this format "Deleted Sale OK. SaleID: 9999" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Name			**Type**	**Return Type**
DELETE_ALL_SALES_FROM_DB			Stored Procedure	None
Description	Delete a row from the SALE table			
Parameters	**Name**	**Type**	**Description**	
Requirements	Delete all rows in the SALE table Set the Sales_YTD value to zero for all rows in the Customer and Product tables			
Exceptions	**Type**		**Raise Application Error Details**	
	Other		-20000. Use value of sqlerrm	

Name			**Type**	**Return Type**
DELETE_ALL_SALES_VIASQLDEV			Stored Procedure	None
Description	Calls DELETE_ALL_SALES_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
Requirements	Display line '----------' using DOPL			
	Display the following **before** deleting the sale using DOPL "Deleting all Sales data in Sale, Customer, and Product tables" using DOPL			
	If successful display in this format "Deletion OK" via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Part 5.2.

Remove the statement **DELETE FROM SALE;** from the section named "TEST DELETION OF EXISTING DATA"
Replace it with the statement DELETE_ALL_SALES_VIASQLDEV;

Add the following statements to TEST PART 5 section of the script

```
ADD_CUSTOMER_VIASQLDEV(10,'Mieko Hayashi');
ADD_CUSTOMER_VIASQLDEV(11,'John Kalia');
ADD_CUSTOMER_VIASQLDEV(12,'Alex Kim');
ADD_PRODUCT_VIASQLDEV(2001,'Chair', 10);
ADD_PRODUCT_VIASQLDEV(2002,'Table', 45);
ADD_PRODUCT_VIASQLDEV(2003,'Lamp', 22);
ADD_COMPLEX_SALE_VIASQLDEV (10,2001,6,'20140301');
ADD_COMPLEX_SALE_VIASQLDEV (10,2002,1,'20140320');
ADD_COMPLEX_SALE_VIASQLDEV (11,2001,1,'20140301');
ADD_COMPLEX_SALE_VIASQLDEV (11,2003,2,'20140215');
ADD_COMPLEX_SALE_VIASQLDEV (12,2001,10,'20140131');
COUNT_PRODUCT_SALES_VIASQLDEV(sysdate-to_date('01-Feb-2000'));
GET_ALLSALES_VIASQLDEV;

DELETE_SALE_VIASQLDEV;
GET_ALLSALES_VIASQLDEV;
DELETE_SALE_VIASQLDEV;
GET_ALLSALES_VIASQLDEV;
DELETE_ALL_SALES_VIASQLDEV;
GET_ALLSALES_VIASQLDEV;
```

PART 6. Custom-made exceptions for attempted deletion of child rows

Part 6.1. Create the Delete Customer and Delete Product procedures

Name			Type	Return Type
DELETE_CUSTOMER			Stored Procedure	
Description	Delete a row from the Customer table			
Parameters	**Name**	**Type**	**Description**	
	pCustid	number	Customer Id	
Requirements	Delete a customer with a matching customer id If ComplexSales exist for the customer, Oracle would normally generate a 'Child Record Found' error (error code -2292). Instead, Create a custom made exception to handle this error & raise the exception below			
Exceptions	**Type**		**Raise Application Error Details**	
	No matching customer id found		-20201. Customer ID not found	
	Customer has child complexsales rows		-20202. Customer cannot be deleted as sales exist	
	Other		-20000. Use value of sqlerrm	

Name			Type	Return Type
DELETE_CUSTOMER_VIASQLDEV			Stored Procedure	None
Description	Calls DELETE_CUSTOMER			
Parameters	**Name**	**Type**	**Description**	
	pCustid	number	Customer Id	
Requirements	Display line '----------' using DOPL			
	Display the following **before** deleting the sale using DOPL "Deleting Customer. Cust Id: 9999 using DOPL			
	If successful display in this format "Deleted Customer OK." via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Name			Type	Return Type
DELETE_PROD_FROM_DB			Stored Procedure	
Description	Delete a row from the Product table			
Parameters	**Name**	**Type**	**Description**	
	pProdid	number	Product Id	
Requirements	Delete a product with a matching Product id If ComplexSales exist for the customer, Oracle would normally generate a 'Child Record Found' error (error code -2292). Instead, Create a custom made exception to handle this error & raise the exception below			
Exceptions	**Type**		**Raise Application Error Details**	
	No matching Product id found		-20301. Product ID not found	
	Product has child complexsales rows		-20302. Product cannot be deleted as sales exist	
	Other		-20000. Use value of sqlerrm	

Name			Type	Return Type
DELETE_PROD_VIASQLDEV			Stored Procedure	None
Description	Calls DELETE_PROD_FROM_DB			
Parameters	**Name**	**Type**	**Description**	
	pProdid	number	Product Id	
Requirements	Display line '----------' using DOPL			
	Display the following **before** deleting the sale using DOPL "Deleting Product. Product Id: 9999 using DOPL			
	If successful display in this format "Deleted Product OK." via DOPL & Commit			
Exceptions	**Type**		**Action**	
	Other		use DOPL to show value of sqlerrm	

Part 6.2.

Add the following statements to TEST PART 6 section of the script

```
ADD_CUSTOMER_VIASQLDEV(17,'Stephen Ward');
ADD_CUSTOMER_VIASQLDEV(18,'Lisa Church');
ADD_CUSTOMER_VIASQLDEV(19,'Joel Pairman');
ADD_PRODUCT_VIASQLDEV(2005,'Desk', 195);
ADD_PRODUCT_VIASQLDEV(2006,'Footrest', 20);
ADD_PRODUCT_VIASQLDEV(2007,'Bookcase', 85);
ADD_COMPLEX_SALE_VIASQLDEV (17,2005,1,'20140302');
ADD_COMPLEX_SALE_VIASQLDEV (17,2006,1,'20140303');
ADD_COMPLEX_SALE_VIASQLDEV (19,2005,1,'20140304');
DELETE_CUSTOMER_VIASQLDEV (17);
DELETE_CUSTOMER_VIASQLDEV(18);
DELETE_CUSTOMER_VIASQLDEV(19);
DELETE_PROD_VIASQLDEV (2005);
DELETE_PROD_VIASQLDEV(2006);
DELETE_PROD_VIASQLDEV(2007);
```

TASK 3 (parts 7-9)

PART 7. Visual Basic STORED PROCEDURES / FUNCTIONS and SQL DEVELOPER testing

Part 7.1. Create these stored procedures/function using SQL Developer

Create a VB or C# host application that calls stored procedures and functions from part 1.

Details:
Create a button (or menu item if you want to create a menu) for various requirements.

The stored procedures / functions to be called by your host application are:

- ADD_CUST_TO_DB
- DELETE_ALL_CUSTOMERS_FROM_DB
- ADD_PROD_TO_DB
- DELETE_ALL_PRODUCTS_FROM_DB
- GET_CUST_STRING_FROM_DB
- UPD_CUST_SALESYTD_IN_DB
- GET_PROD_STRING_FROM_DB
- UPD_PROD_SALESYTD_IN_DB
- UPD_CUST_STATUS_IN_DB
- ADD_SIMPLE_SALE_TO_DB
- SUM_CUST_SALESYTD

Note: NEVER call an Oracle SP or SF that has that contains the text _VIASQLDEV.

Obtaining user data:
You may use any method you like to obtain data from the user interactively.

- The most simple method is to use an InputBox statement for each piece of data required.
- Alternatively you can use text boxes on a Form.
- There are no additional or bonus marks for using extravagant designs, so I suggest that you keep it simple.

Displaying output
As each requirement (above) is successfully completed (such as adding a new customer), your code must display an appropriate message.
The messages must be the same as those found in the "_VIASQLDEV" Stored Procedures from Tasks 1 and 2 above.
E.g.
When a new customer is successfully added, the message "Customer Added OK" is displayed.
If unsuccessful then the exception message must be displayed.

Suggestion: Use a Listbox or a Label for all display items

Transactions
For each requirement (above) that could modify the database, it **must** be performed within a **transaction**. Every **transaction** must be either **explicitly** committed or rolled-back.

Your Oracle stored procedures and functions must **never** perform an explicit commit or rollback.

PART 8. Visual Basic / Cursors / Packages

Part 8.1. Create these stored procedures/function using SQL Developer

Modify the VB or C# application so that there are two additional buttons or menu options to retrieve all customer and product data.

This will require you to create **packages** in your Oracle database that data retrieved by GET_ALLPROD_FROM_DB & GET_ALLCUST can be processed and displayed in VB/C#.

PART 9. Visual Basic – remaining functionality

Part 9.1. Create these stored procedures/function using SQL Developer

Modify the VB or C# application so that there are additional buttons or menu options to retrieve carry out all other functionality that you have created in parts 4, 5 & 6.

Note: NEVER call an Oracle SP or SF that contains the text _VIASQLDEV.