**COURSE CCPS** **610**

**DATABASE SYSTEMS (DBMS) II**

**Assignment**

**INSTRUCTOR NAME: ERSAN CAM**

**MARKS ALLOTTED: 100 (%70 back end + %30 Front End GUI)**

**WEIGHTING: 25%**

**General Instructions :**

You can use any Web based front end programming tool which can connect to our database in Lab or your own database

* Clean Coding, Interaction with Database (Connectivity, Login page etc ) , GUI features (Drop down menu , list box, Radio boxes etc if needed) and actual Presentation skills will be evaluated during presentation time.
* Expectation is that everyone in the team shares responsibility. If one person writes some stored procedure and the other one writes function , one or two can work on GUI same as in real life environment.
* **Presentation will be on April 7th.**
* Make power point presentation and do share with instructor in advance. Instructor will open presentation for you to present in front of class or you can plug your own laptop.
* If you want to use your own laptop , it is your responsibility to test connectivity of Projector otherwise do send all files and presentation to instructor and do DEMO from Instructor’s computer.

**Marking guidelines**

**Back-end programming (Up to 70 points)**

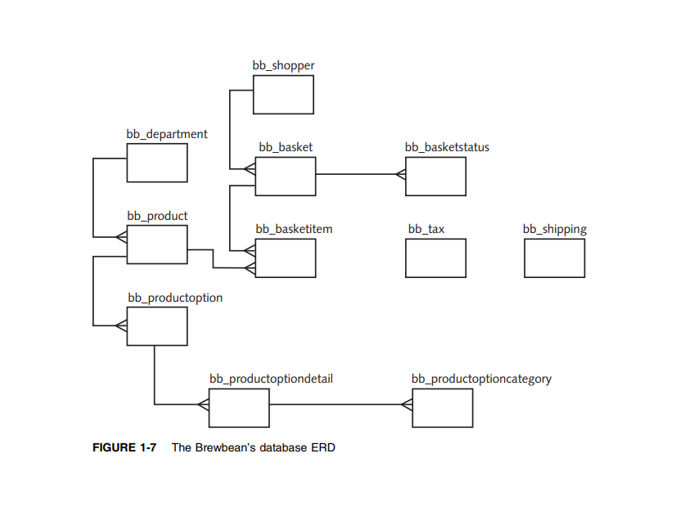
All the procedure ,function, triggers must be cleanly compiled with no errors. Tested on SQL Developer

You will be collecting points by doing successful Demo and test in front of Instructor and audience

**2-Front end GUI effectiveness (0 up to30 pnts)**

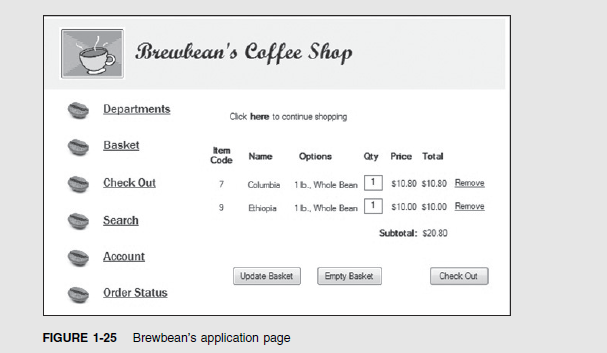
How user friendly is your GUI? Is it just basic & quickly written-basic but functional GUI or efficient and colorful/fancy GUI ?

Presentation skills (Eye contact with audience - Voice control - body gesture etc (Do some research on basic presentation skills )

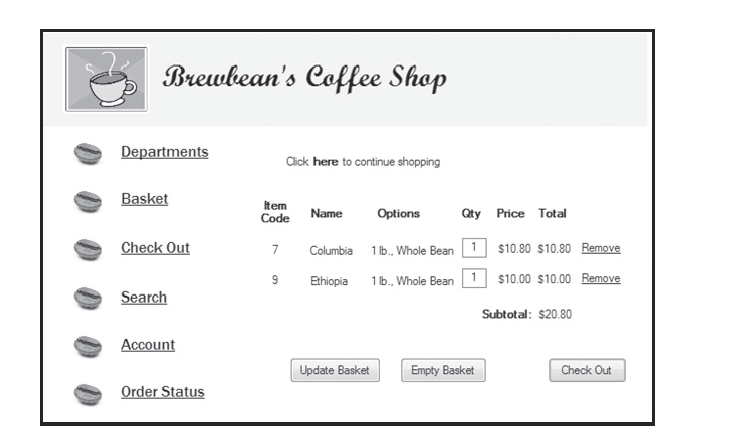


This is a hypothetical real-world business—Brewbean’s, an online coffee goods retailer. First, the business

operation and database structure are introduced and given. There are some parts in this assignments that your help to solve for current application challenges Brewbean’s needs to address.



These are some Sample Screens of Coffee Shop application. You don’t need to make exact same GUI as it is here. These are just some ideas



**There are some challenges already in the application (Keep this in your mind)**

These tasks include calculating taxes, calculating shipping charges, checking and/or updating product

inventory, and determining whether the shopper already has credit card information stored.

So how do you approach programming the required tasks? Typically,

-a block is written to perform a particular task. For example, for the Brewbean’s shopping basket, a

PL/SQL block might be written to handle calculating shipping costs based on the number

of items in the order and the type of shipping method that’s selected.

-In addition, separate blocks can be developed to calculate applicable taxes and handle inventory issues. Using

this modular approach makes developing an application and maintaining and reusing code

more manageable than having large, more complex blocks of code.

- Keep reusability in mind (Procedures, Functions) as you construct codes or modules of code.

-Look at the shipping calculation as an example. This block could accept input, such as a basket

number, from the shopping basket. The program then might use this input to query the

database for the number of items in the basket. By accepting input values, the same

PL/SQL block is made dynamic, meaning it can calculate the shipping costs for any order

because it queries the database for the total number of items based on the basket number.

**Task1 : Creating a Procedure and GUI screen for Product Description change ability.**

GUI will have two edit box. One for Product ID (Primary Key) and then another edit box for New Description to overwrite old one.

Use these below steps to create a procedure that allows a company employee to make corrections to

a product’s assigned name (Change Product Description). Review the BB\_PRODUCT table and identify the PRODUCT NAME and PRIMARY KEY columns. The procedure needs two IN parameters to identify the product ID and supply the new description. This procedure needs to perform only a DML action, so no

OUT parameters are necessary. (**Optional** : It is best to list Product code name from Drop down list menu and let user choose one product name but find proper Product Code and pass that code as Product Code)

**Hint For Procedure Body:**

**UPDATE bb\_product**

**SET description = p\_descrip**

**WHERE PK= p**

**TIP : Don’t forget to add COMMIT command after each INSERT -UPDATE and DELETE commands**

**Code：**

**CREATE OR REPLACE PROCEDURE changeProductDesc (id IN NUMBER, p\_descrip IN varchar2)**

**is**

**BEGIN**

**UPDATE bb\_product**

**SET Description = p\_descrip**

**WHERE idProduct = id;**

**commit;**

**END;**

**Testing:**

1. Before testing the procedure, verify the current description value for product ID 1 with

**SELECT \* FROM bb\_product;**.

2. Call the procedure with parameter values of **1** for the product ID and **CapressoBar Model**

**#388** for the description.

3. Verify that the update was successful by querying the table with **SELECT \* FROM**

**bb\_product;**.

**Task 2: Create(Enter) new product by using a Procedure with IN Parameters**

Follow these steps to create a procedure that allows a company employee to **add a new**

**product to the database**. This procedure needs only IN parameters.

1. In SQL Developer, create a procedure named **PROD\_ADD\_SP** that adds a row for a new

product in the BB\_PRODUCT table. Keep in mind that the user provides values for the

product name, description, image filename (use same template for file name as ‘roasted.jpg’ or ‘double-roasted.jpg’ ) , price, and active status. Address the input values or parameters in the same order as in the preceding sentence.

Code:

CREATE OR REPLACE PROCEDURE PROD\_ADD\_SP (p\_id IN number, p\_name IN varchar2,p\_descrip IN varchar2, p\_image IN varchar2, p\_price IN number, p\_status IN number)

is

BEGIN

insert into bb\_product (idProduct, ProductName, Description,ProductImage, Price, Active)

values

(p\_id, p\_name, p\_descrip,p\_image, p\_price, p\_status );

commit;

END;

2. Call the procedure with these parameter values:

**('Roasted Blend', 'Well-balanced mix of roasted beans, a medium body', 'roasted.jpg',9.50,1)**.

execute PROD\_ADD\_SP(11,'Roasted Blend', 'Well-balanced mix of roasted beans, a medium body', 'roasted.jpg',9.50,1)

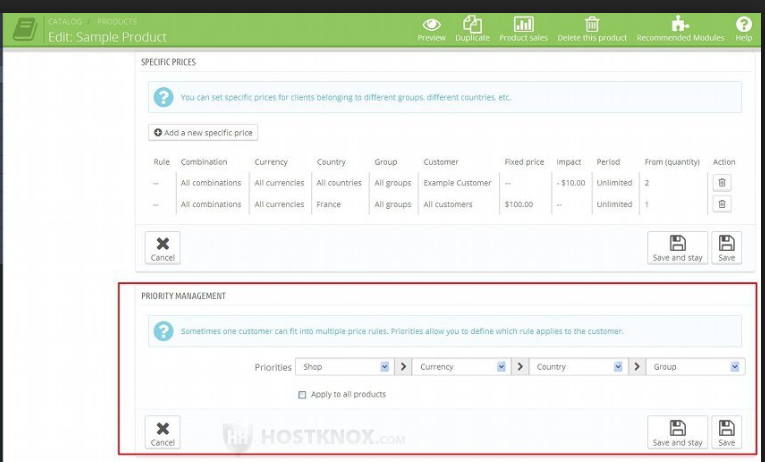
3. Check whether the Insert/Update was successful by querying the BB\_PRODUCT table.

-This will be a page where you allow new product information to be entered into Edit box by end user and then there will be INSERT or ADD Product button to save them by calling this Stored Procedure.

-And Once your Product add tasks is completed GUI will bring the full product list with newly added one at the below part of your screen

-Also there should be only one Product Search capability in the same Page. On the same page,(maybe in different part of your page) open a edit box and accept product name and search based on that product name and bring to list on screen.

**This below one is a sample screen (You don’t have to make same exact one)**



**Task 3: Calculating the Tax on an Order**

Follow these steps to create a procedure for calculating the tax on an order. The BB\_TAX table

contains states that require submitting taxes for Internet sales. If the state isn’t listed in the

table**, no tax should be assessed on the order**. The **shopper’s state** and **basket subtotal** are the

inputs to the procedure, and the tax amount should be returned.

Make two edit box in your GUI form to accept **Shopper’s state** and **basket subtotal**

1. In SQL Developer, create a procedure named **TAX\_COST\_SP**. Remember that the state

and subtotal values **are inputs to the procedure**, which should **return the tax amount**.

Review the BB\_TAX table, which contains the tax rate for each applicable state.

Code:

For sql developer(

create or replace procedure TAX\_COST\_SP

(

 par4State in bb\_tax.state%type,

 par\_subtot in number

)

is

taxrate bb\_tax.taxrate%type;

v\_tax number;

BEGIN

  SELECT taxrate \* par\_subtot

    INTO v\_tax

    FROM bb\_tax

    WHERE state = par4State;

    DBMS\_OUTPUT.PUT\_LINE('$' || v\_tax);

END TAX\_COST\_SP;)

**only for php output**(create or replace procedure TAX\_COST\_SP

(

 par4State in bb\_tax.state%type,

 par\_subtot in number,

 tax\_result out number

)

is

taxrate bb\_tax.taxrate%type;

v\_tax number;

BEGIN

  SELECT taxrate \* par\_subtot

    INTO v\_tax

    FROM bb\_tax

    WHERE state = par4State;

    tax\_result := v\_tax;

END TAX\_COST\_SP;

)

**Testing :**

2. Call the procedure with the values **VA** for the state and **$100** for the subtotal. Display the

tax amount the procedure returns. (It should be $4.50.)

execute TAX\_COST\_SP('VA', 100);

**Task 4: Updating Order Status**

Create a procedure named **STATUS\_SHIP\_SP** that allows an employee in the Shipping

Department to update an **order status** to add **shipping information.** The BB\_BASKETSTATUS

table lists events for each order so that a shopper can see the **current status**, **date**, and

**comments as each stage** of the order process is finished. The IDSTAGE column of the

BB\_BASKETSTATUS table identifies each stage; the value 3 in this column indicates that an

order has been shipped.

The procedure should allow adding a row with an IDSTAGE of 3, date shipped, tracking

number, and shipper. The BB\_STATUS\_SEQ sequence is used to provide an incremental unique auto increment value for the primary key column.

**HINT**: In your Procedure main part (BEGIN \_END)

BEGIN

INSERT INTO bb\_basketstatus (idstatus, idbasket, idstage, dtstage,

shipper, shippingnum)

VALUES (**bb\_status\_seq.NEXTVAL**, p\_basketid, 3, p\_date, p\_shipper,

p\_shipnum);

COMMIT;

END;

**bb\_status\_seq.NEXTVAL *is auto number generator bb\_status\_seq.NEXTVAL is created in the database already. Every time you call this sequence nextval it will generate automatic/an unique /non repeated number for Primary key for idstatus***

Code：

CREATE OR REPLACE PROCEDURE STATUS\_SHIP\_SP(p\_idBasket IN number,

p\_dtStage IN varchar2,

p\_shipper IN varchar2,

p\_ShippingNum IN varchar2)

is

BEGIN

INSERT INTO bb\_BasketStatus(idStatus, idBasket, idStage, dtStage, shipper, ShippingNum)

VALUES (bb\_status\_seq.NEXTVAL, p\_idBasket, 3, p\_dtStage, p\_shipper, p\_ShippingNum);

COMMIT;

END;

Test the procedure with the following information:

Basket # = 3

Date shipped = 20-FEB-12

Shipper = UPS

Tracking # = ZW2384YXK4957

Accept entry on your GUI for 4 different edit box and collect them and use them to call stored procedure in your code

**Execute** **status\_ship\_sp**(3,'20-FEB-12','UPS','ZW2384YXK4957')

**Task 5: Adding Items to a Basket**

As a shopper selects products on the Brewbean’s site, a procedure is needed to add a newly

selected item to the current shopper’s basket. Create a procedure named **BASKET\_ADD\_SP** that

accepts a product ID, basket ID, price, quantity, size code option (1 or 2), and form code option

(3 or 4) and uses this information to add a new item to the BB\_BASKETITEM table. The table’s

PRIMARY KEY column is generated by BB\_IDBASKETITEM\_SEQ.

**Testing:**

Run and test the procedure with the following values:

• Basket ID—14

• Product ID—8

• Price—10.80

• Quantity—1

• Size code—2

• Form code—4

**Hint**:

INSERT INTO **BASKET\_ADD\_SP VALUES (**BB\_IDBASKETITEM\_SEQ.NEXTVAL, x,x,x,x **);**

COMMIT;

Code:

CREATE OR REPLACE PROCEDURE BASKET\_ADD\_SP(p\_idBasket IN number,

p\_idProduct IN number,

p\_Price in number,

p\_Quantity in number,

p\_option1 IN number,

p\_option2 IN number)

is

BEGIN

INSERT INTO bb\_basketItem(idBasketItem, idBasket, idProduct, Price, Quantity, option1, option2)

VALUES (BB\_IDBASKETITEM\_SEQ.NEXTVAL, p\_idBasket, p\_idProduct, p\_Price, p\_Quantity, p\_option1, p\_option2);

COMMIT;

END;

.

Test: Execute BASKET\_ADD\_SP(14,8,10.8,1,2,4)

**Task 6: Identifying Sale Products**

When a product is placed on sale, Brewbean’s records the sale’s start and end dates in

columns of the BB\_PRODUCT table. A **function** is needed to provide sales information when a

shopper selects an item. If a product is on sale, the function should return the value ON SALE!.

However, if it isn’t on sale, the function should return the value Great Deal!. These values are

used on the product display page.

Create a function named **CK\_SALE\_SF** that accepts a **date** and

**product ID** as parameters, checks whether the date falls within the product’s sale period, and returns

the corresponding string value. Bring the result onto GUI screen

Test the function with the product ID 6 and two dates: 10-JUN-12 and 19-JUN-12. Verify your results by reviewing the product sales information.

Code：（for sql developer）

CREATE or replace FUNCTION CK\_SALE\_SF

(f\_idproduct in bb\_product.idproduct%type,

f\_date in date)

return varchar is

v\_salestart bb\_product.salestart%type;

v\_saleend bb\_product.saleend%type;

v\_result varchar2(20);

salestart bb\_product.salestart%type;

saleend bb\_product.saleend%type;

begin

select salestart, saleend into v\_salestart, v\_saleend from bb\_product

where idproduct = f\_idproduct;

if v\_salestart <= f\_date and v\_saleend >= f\_date then

v\_result:= 'On Sale';

else v\_result:= 'Good Deal';

end if;

return v\_result;

end CK\_SALE\_SF;

SELECT CK\_SALE\_SF (6,'17-JUN-12' ) FROM dual;

Code: (for php output)

CREATE or replace FUNCTION CK\_SALE\_SF\_test

(f\_date VARCHAR,

f\_idproduct in bb\_product.idproduct%type

)

return varchar is

v\_salestart VARCHAR2(20);

v\_saleend VARCHAR2(20);

v\_result VARCHAR2(20);

salestart DATE;

saleend DATE;

begin

select TO\_CHAR(salestart,'DD-MON-YY'), TO\_CHAR(saleend,'DD-MON-YY') into v\_salestart, v\_saleend from bb\_product

where idproduct = f\_idproduct;

if f\_date >= v\_salestart and f\_date <= v\_saleend then

v\_result:= 'On Sale';

else v\_result:= 'Good Deal';

end if;

return v\_result;

end CK\_SALE\_SF\_test;

SELECT CK\_SALE\_SF\_test ('19-JUN-12',6) FROM dual;

===================================================================

Reports Menu

Create separate Report menu where you may have these sub reports

**Report 1: Report to show whether all items in her/his basket are in Stock or not? Using an Explicit Cursor**

In the Brewbean’s application, a customer can ask to check whether all items in his or her

**basket** are in stock. In this task, you create a procedure that uses an explicit cursor to

retrieve all items in the basket and determine whether all items are in stock by comparing the

item quantity with the product stock amount.

If all items are in stock, display the message **“All items in stock!”** **onscreen**.

If not, display the message **“All items NOT in stock!”** **onscreen**.

The basket number will be entered by End User on the screen and then used as INPUT parameter for procedure (Either edit box or make drop down menu item where you bring all basket numbers from database – Up to you) .

**HINT:** Use below code . Either place this code into Stored Procedure or use it direct PL/SQL in your Java, .ASP ort .NET GUI front end if you can. Up to you.

DECLARE

CURSOR cur\_basket IS

SELECT bi.idBasket, bi.quantity, p.stock

FROM bb\_basketitem bi INNER JOIN bb\_product p

USING (idProduct)

WHERE bi.idBasket = 6;

TYPE type\_basket IS RECORD (

basket bb\_basketitem.idBasket%TYPE,

qty bb\_basketitem.quantity%TYPE,

stock bb\_product.stock%TYPE);

rec\_basket type\_basket;

lv\_flag\_txt CHAR(1) := 'Y';

BEGIN

FOR rec\_basket IN cur\_basket

LOOP

IF rec\_basket.stock < rec\_basket.qty THEN lv\_flag\_txt := 'N'; END IF;

END LOOP;

IF lv\_flag\_txt = 'Y' THEN DBMS\_OUTPUT.PUT\_LINE('All items in stock!'); END IF;

IF lv\_flag\_txt = 'N' THEN DBMS\_OUTPUT.PUT\_LINE('All items NOT in stock!'); END IF;

END;

For sql developer(

CREATE OR REPLACE PROCEDURE DISPLAY\_STOCK (p\_id\_basket IN bb\_basketitem.idbasket%TYPE)is

CURSOR cur\_basket (c\_idbasket bb\_basketitem.idBasket%TYPE)IS

SELECT bi.idBasket, bi.quantity, p.stock

FROM bb\_basketitem bi INNER JOIN bb\_product p

USING (idProduct)

WHERE bi.idBasket = c\_idbasket;

TYPE type\_basket IS RECORD (

basket bb\_basketitem.idBasket%TYPE,

qty bb\_basketitem.quantity%TYPE,

stock bb\_product.stock%TYPE);

rec\_basket type\_basket;

lv\_flag\_txt CHAR(1) := 'Y';

BEGIN

OPEN cur\_basket(p\_id\_basket);

LOOP

FETCH cur\_basket INTO rec\_basket;

EXIT WHEN cur\_basket%NOTFOUND;

IF rec\_basket.stock < rec\_basket.qty

THEN lv\_flag\_txt := 'N';

END IF;

END LOOP;

CLOSE cur\_basket;

IF lv\_flag\_txt = 'Y'

THEN DBMS\_OUTPUT.PUT\_LINE('All items in stock!');

END IF;

IF lv\_flag\_txt = 'N'

THEN DBMS\_OUTPUT.PUT\_LINE('All items NOT in stock!');

END IF;

END;

/)

For php output(

CREATE OR REPLACE PROCEDURE DISPLAY\_STOCK (p\_id\_basket IN bb\_basketitem.idbasket%TYPE, output out varchar2)is

CURSOR cur\_basket (c\_idbasket bb\_basketitem.idBasket%TYPE)IS

SELECT bi.idBasket, bi.quantity, p.stock

FROM bb\_basketitem bi INNER JOIN bb\_product p

USING (idProduct)

WHERE bi.idBasket = c\_idbasket;

TYPE type\_basket IS RECORD (

basket bb\_basketitem.idBasket%TYPE,

qty bb\_basketitem.quantity%TYPE,

stock bb\_product.stock%TYPE);

rec\_basket type\_basket;

lv\_flag\_txt CHAR(1) := 'Y';

BEGIN

OPEN cur\_basket(p\_id\_basket);

LOOP

FETCH cur\_basket INTO rec\_basket;

EXIT WHEN cur\_basket%NOTFOUND;

IF rec\_basket.stock < rec\_basket.qty

THEN lv\_flag\_txt := 'N';

END IF;

END LOOP;

CLOSE cur\_basket;

IF lv\_flag\_txt = 'Y'

THEN output:= 'All items in stock!';

END IF;

IF lv\_flag\_txt = 'N'

THEN output:= 'All items NOT in stock!';

END IF;

END;

/

)

**Report 2: Calculating a Shopper’s Total Spending**

Many of the reports generated from the system calculate the total dollars in a shopper’s

purchases. Follow these steps to create a function named **TOT\_PURCH\_SF** that accepts a

shopper ID as input and returns the total dollars the shopper has spent with Brewbean’s.

Use the function in a SELECT statement that shows the shopper ID and total purchases for every shopper in the database.

1. Develop and run a CREATE FUNCTION statement to create the **TOT\_PURCH\_SF** function.

The function code needs a formal parameter for the shopper ID and to sum the TOTAL

column of the BB\_BASKET table.

CREATE OR REPLACE FUNCTION TOT\_PURCH\_SF

(shopper\_id IN BB\_BASKET.IDSHOPPER%TYPE)

RETURN NUMBER IS V\_TOTAL BB\_BASKET.TOTAL%TYPE;

BEGIN

SELECT SUM(BB\_BASKET.TOTAL)AS TOTAL\_PURCHASE INTO V\_TOTAL FROM BB\_BASKET

WHERE IDSHOPPER = shopper\_id;

RETURN V\_TOTAL;

END TOT\_PURCH\_SF;

2. Develop a SELECT statement, using the BB\_SHOPPER table, to produce a list of each

shopper in the database and his or her total purchases.

HINT : Optionally you can have one edit box to accept end user’s entry for Shopper ID as input. If user enter any other value than valid Shopper ID (Such as date or character then give error , if end user enter valid Shopper ID then list Total Spending for that specific Shopper ID, if edit box is left Empty (Without anything) and press LIST button then list all of the Shopper ID and their total spending beside each of them.

HINT: For this , you can use PL/SQL procedure, or PL/SQL Anonyms block or even just use Front End GUI capability (SQL) to bring this onto Screen. Up to you