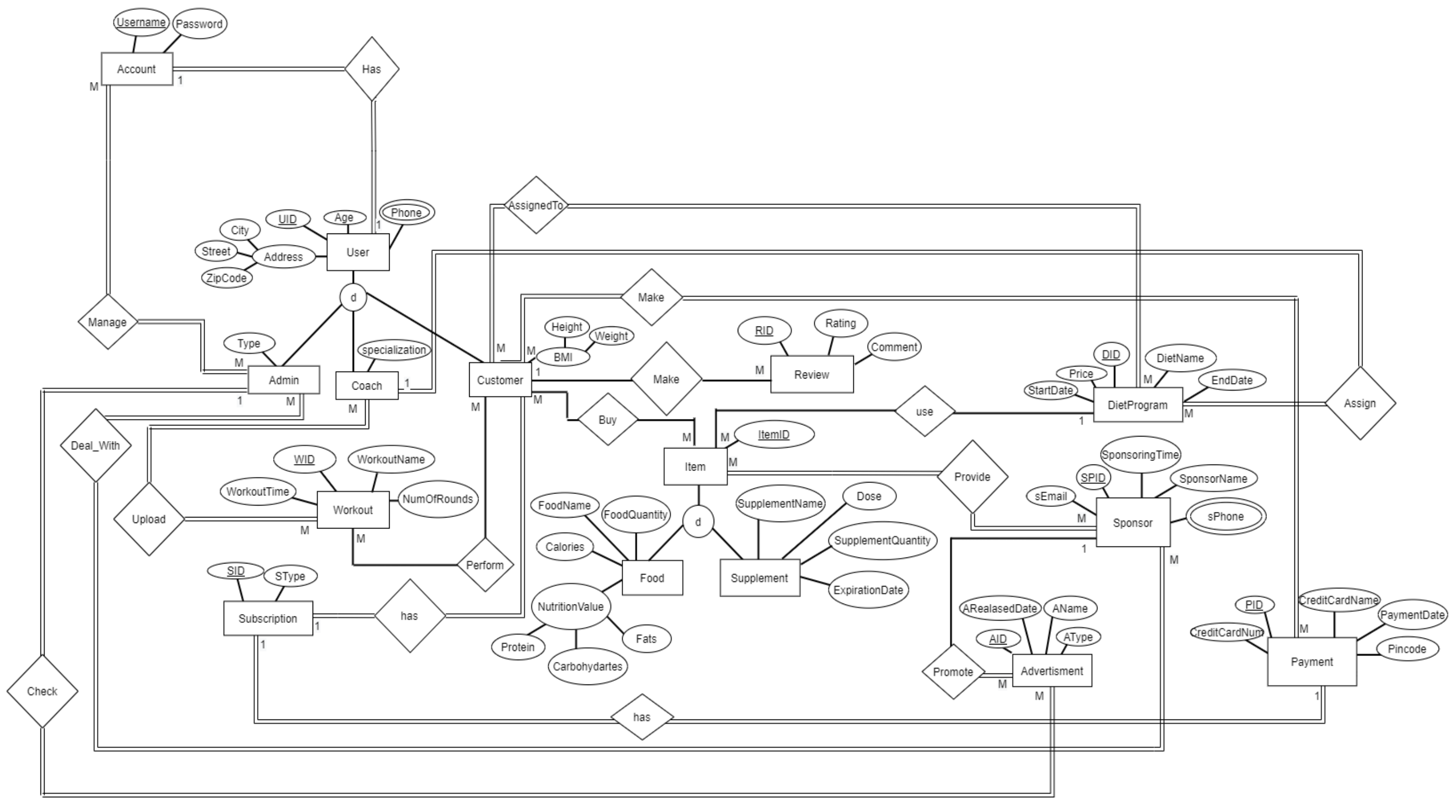
Advanced Database

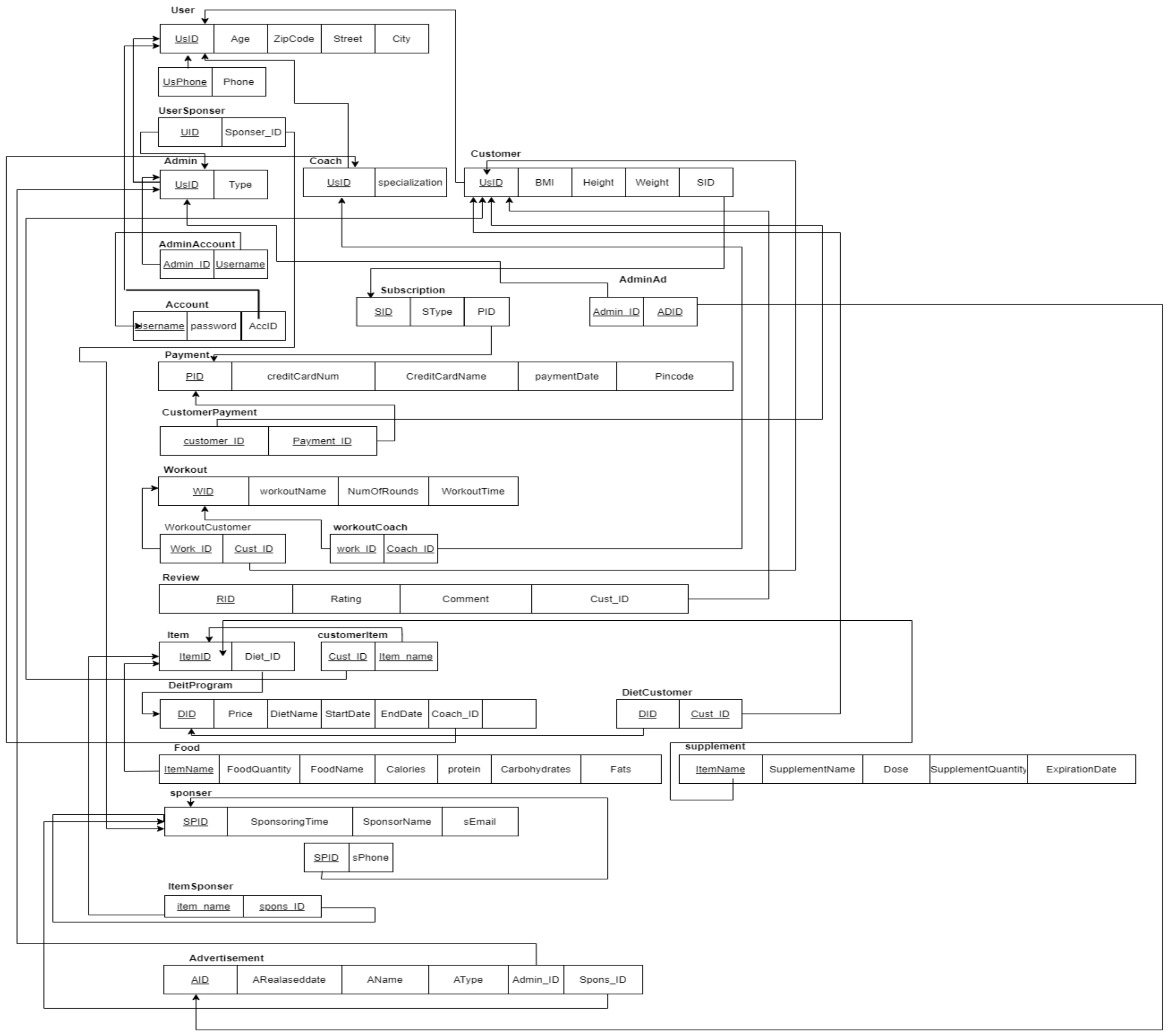
Nutrition System

Name	ID	Email	Contribution
Salma Tarek	184847	Salma184847@bue.edu.eg	All
Katy Milad	179295	Katy179295@bue.edu.eg	All
Youssef Bushra	190346	Youssef190346@bue.edu.eg	All
Abdulrhman Zakaria	186883	Abdulrhman186883@bue.edu.eg	All

ERD

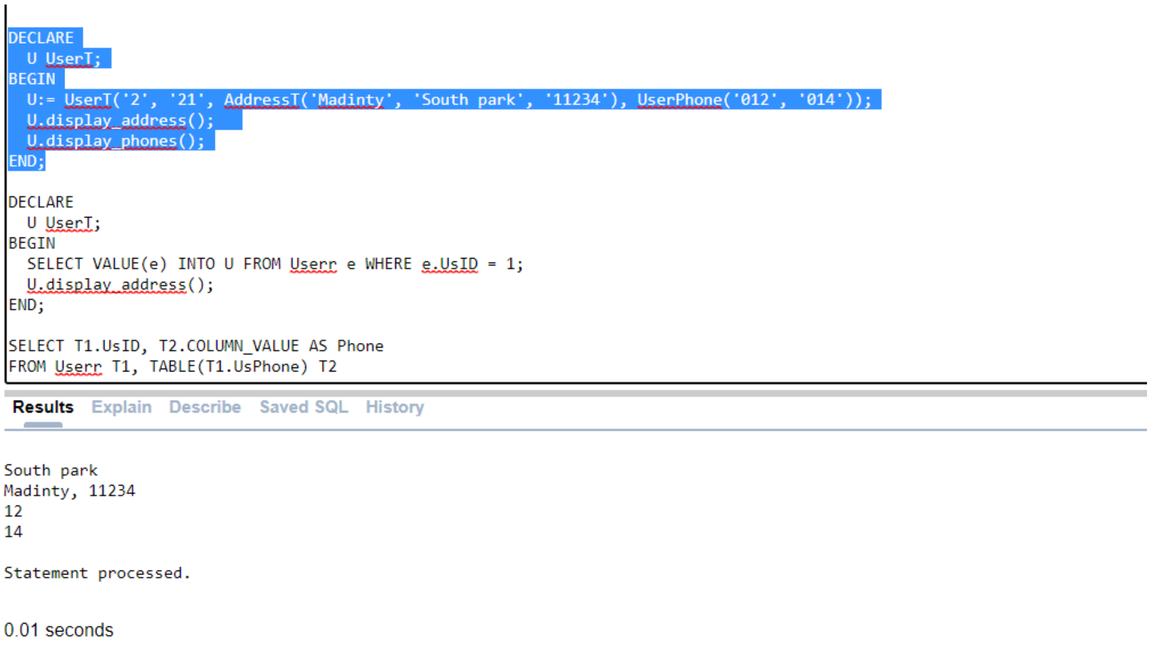

Account
Username
Password
Has
User
UID
Age
Phone
City
Street
Address
ZipCode
d
Manage
Admin
Type
Coach
specialization
Customer
Height
Weight
BMI
AssignedTo
Make
Review
RID
Rating
Comment
DietProgram
DID
Price
StartDate
DietName
EndDate
Assign
use
Buy
Item
ItemID
Provide
Sponsor
SPID
sEmail
SponsoringTime
SponsorName
sPhone
Deal_With
Upload
Workout
WID
WorkoutName
WorkoutTime
NumOfRounds
Perform
Subscription
SID
SType
has
Food
FoodName
FoodQuantity
Calories
NutritionValue
Protein
Carbohydartes
Fats
Supplement
SupplementName
Dose
SupplementQuantity
ExpirationDate
Advertisment
ARealasedDate
AName
AID
AType
Promote
Payment
PID
CreditCardName
PaymentDate
CreditCardNum
Pincode
Check
M
1

Mapping


User
UsID
Age
ZipCode
Street
City
UsPhone
Phone
UserSponser
UID
Sponser_ID
Admin
UsID
Type
Coach
UsID
specialization
Customer
UsID
BMI
Height
Weight
SID
AdminAccount
Admin_ID
Username
Account
Username
password
AccID
Subscription
SID
SType
PID
AdminAd
Admin_ID
ADID
Payment
PID
creditCardNum
CreditCardName
paymentDate
Pincode
CustomerPayment
customer_ID
Payment_ID
Workout
WID
workoutName
NumOfRounds
WorkoutTime
WorkoutCustomer
Work_ID
Cust_ID
workoutCoach
work_ID
Coach_ID
Review
RID
Rating
Comment
Cust_ID
Item
ItemID
Diet_ID
customerItem
Cust_ID
Item_name
DeitProgram
DID
Price
DietName
StartDate
EndDate
Coach_ID
DietCustomer
DID
Cust_ID
Food
ItemName
FoodQuantity
FoodName
Calories
protein
Carbohydrates
Fats
supplement
ItemName
SupplementName
Dose
SupplementQuantity
ExpirationDate
sponser
SPID
SponsoringTime
SponsorName
sEmail
SPID
sPhone
ItemSponser
item_name
spons_ID
Advertisement
AID
ARealaseddate
AName
AType
Admin_ID
Spons_ID

Functionalities description

1) **Display Address:** It displays the information (address) of the users including: Coach, Admin, and Customers.

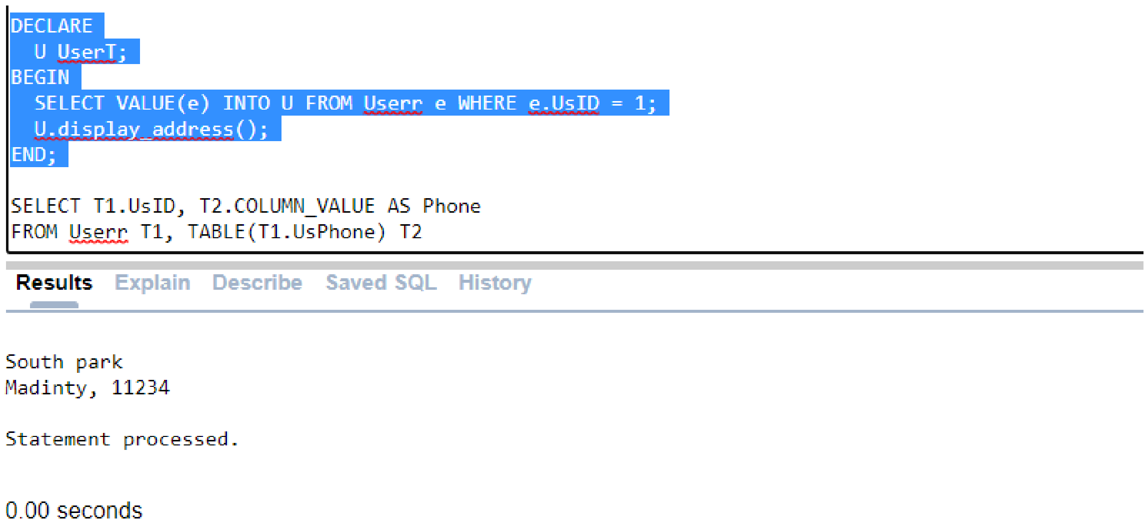


```
DECLARE
  U UserT;
BEGIN
  U:= UserT('2', '21', AddressT('Madinty', 'South park', '11234'), UserPhone('012', '014'));
  U.display_address();
  U.display_phones();
END;

DECLARE
  U UserT;
BEGIN
  SELECT VALUE(e) INTO U FROM Userr e WHERE e.UsID = 1;
  U.display_address();
END;

SELECT T1.UsID, T2.COLUMN_VALUE AS Phone
FROM Userr T1, TABLE(T1.UsPhone) T2
```

Results Explain Describe Saved SQL History

```
South park
Madinty, 11234
12
14

Statement processed.
```

0.01 seconds



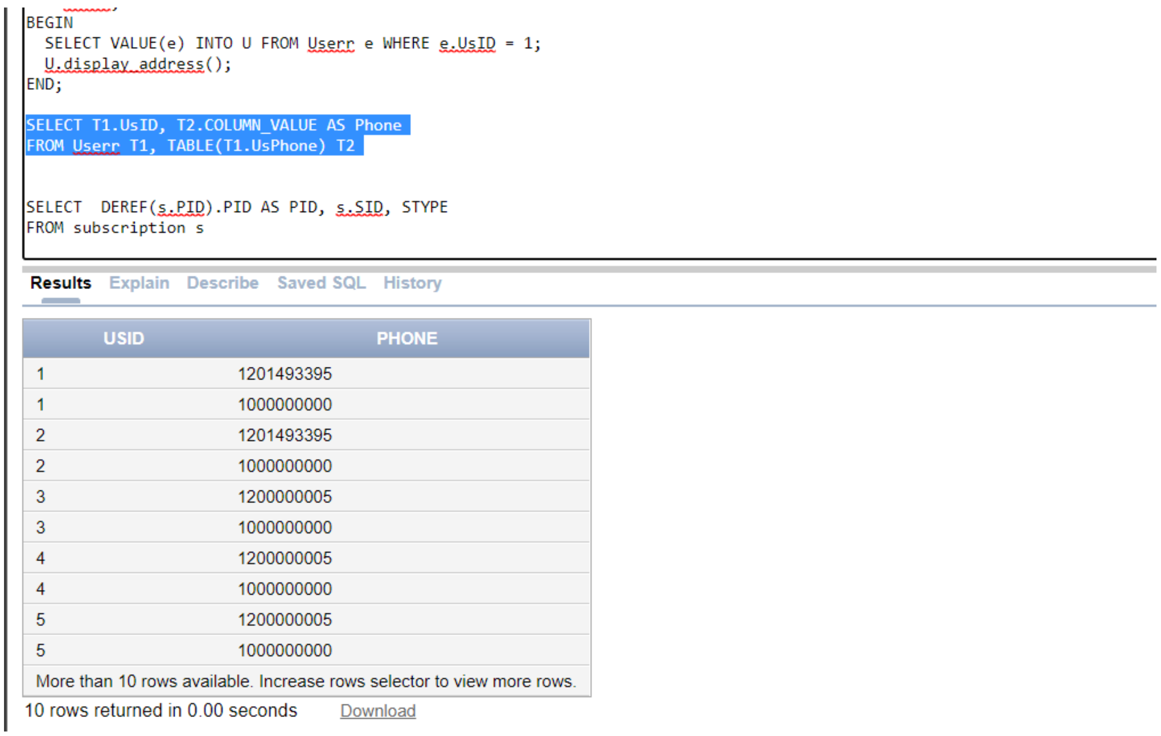
```
DECLARE
  U UserT;
BEGIN
  SELECT VALUE(e) INTO U FROM Userr e WHERE e.UsID = 1;
  U.display_address();
END;

SELECT T1.UsID, T2.COLUMN_VALUE AS Phone
FROM Userr T1, TABLE(T1.UsPhone) T2
```

Results Explain Describe Saved SQL History

```
South park
Madinty, 11234

Statement processed.
```

0.00 seconds

2) **Display Phones:** It displays the information (phone number) of the users including: Coach, Admin, and Customers.



```
BEGIN
  SELECT VALUE(e) INTO U FROM Userr e WHERE e.UsID = 1;
  U.display_address();
END;

SELECT T1.UsID, T2.COLUMN_VALUE AS Phone
FROM Userr T1, TABLE(T1.UsPhone) T2


SELECT  DEREF(s.PID).PID AS PID, s.SID, STYPE
FROM subscription s
```

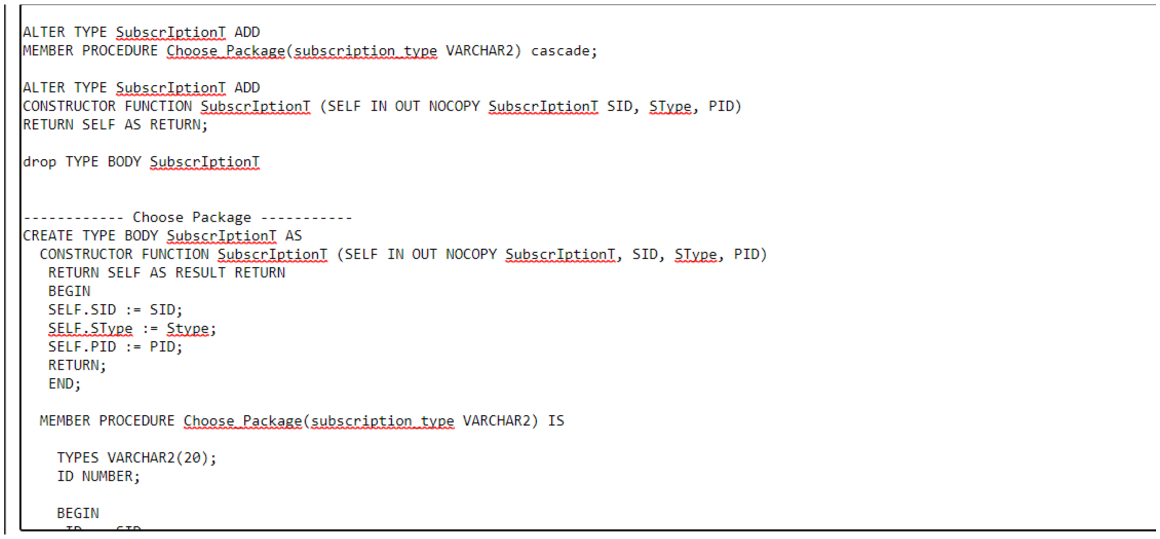
Results Explain Describe Saved SQL History

USID	PHONE
1	1201493395
1	1000000000
2	1201493395
2	1000000000
3	1200000005
3	1000000000
4	1200000005
4	1000000000
5	1200000005
5	1000000000

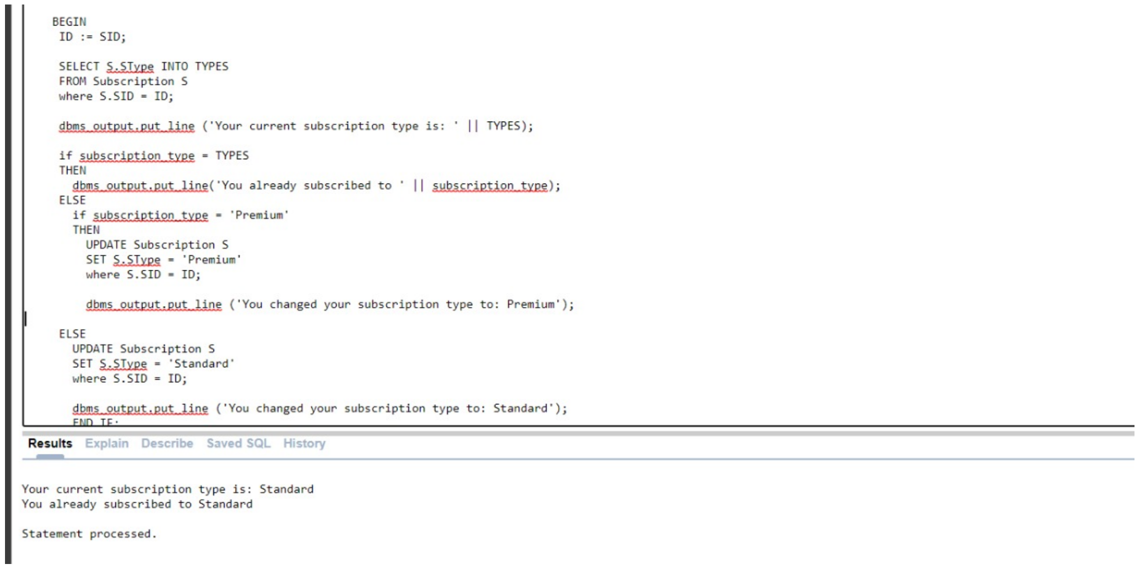
More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.00 seconds Download

3) **Choose Package:** The customer chooses his package type either standard or premium. The package chosen is checked, then it got changed if it's not the same as the previous package type. **(Procedure 1)**



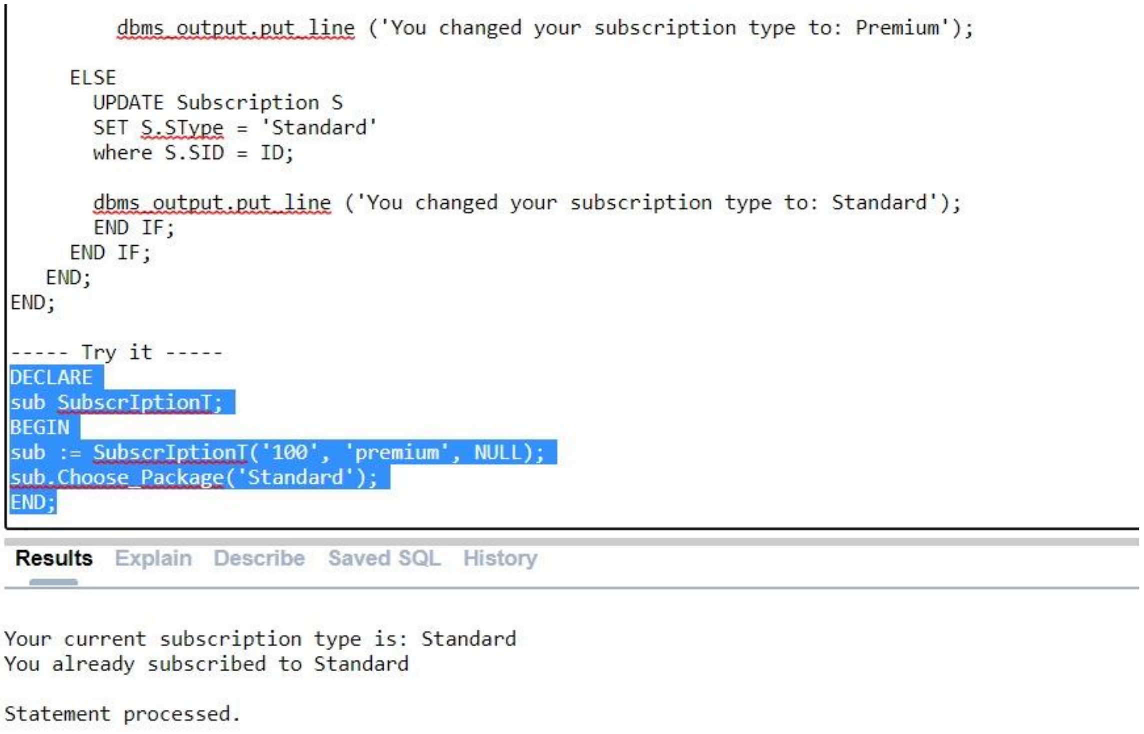
```
ALTER TYPE SubscrIptionT ADD
MEMBER PROCEDURE Choose_Package(subscription_type VARCHAR2) cascade;

ALTER TYPE SubscrIptionT ADD
CONSTRUCTOR FUNCTION SubscrIptionT (SELF IN OUT NOCOPY SubscrIptionT SID, SType, PID)
RETURN SELF AS RETURN;

drop TYPE BODY SubscrIptionT


------------ Choose Package -----------
CREATE TYPE BODY SubscrIptionT AS
  CONSTRUCTOR FUNCTION SubscrIptionT (SELF IN OUT NOCOPY SubscrIptionT, SID, SType, PID)
   RETURN SELF AS RESULT RETURN
   BEGIN
   SELF.SID := SID;
   SELF.SType := Stype;
   SELF.PID := PID;
   RETURN;
   END;

  MEMBER PROCEDURE Choose_Package(subscription_type VARCHAR2) IS

    TYPES VARCHAR2(20);
    ID NUMBER;

    BEGIN
```



```
   BEGIN
    ID := SID;

    SELECT S.SType INTO TYPES
    FROM Subscription S
    where S.SID = ID;

    dbms_output.put_line ('Your current subscription type is: ' || TYPES);

    if subscription_type = TYPES
    THEN
      dbms_output.put_line('You already subscribed to ' || subscription_type);
    ELSE
      if subscription_type = 'Premium'
      THEN
        UPDATE Subscription S
        SET S.SType = 'Premium'
        where S.SID = ID;

        dbms_output.put_line ('You changed your subscription type to: Premium');

    ELSE
      UPDATE Subscription S
      SET S.SType = 'Standard'
      where S.SID = ID;

      dbms_output.put_line ('You changed your subscription type to: Standard');
      END IF;
```

Results Explain Describe Saved SQL History

```
Your current subscription type is: Standard
You already subscribed to Standard

Statement processed.
```

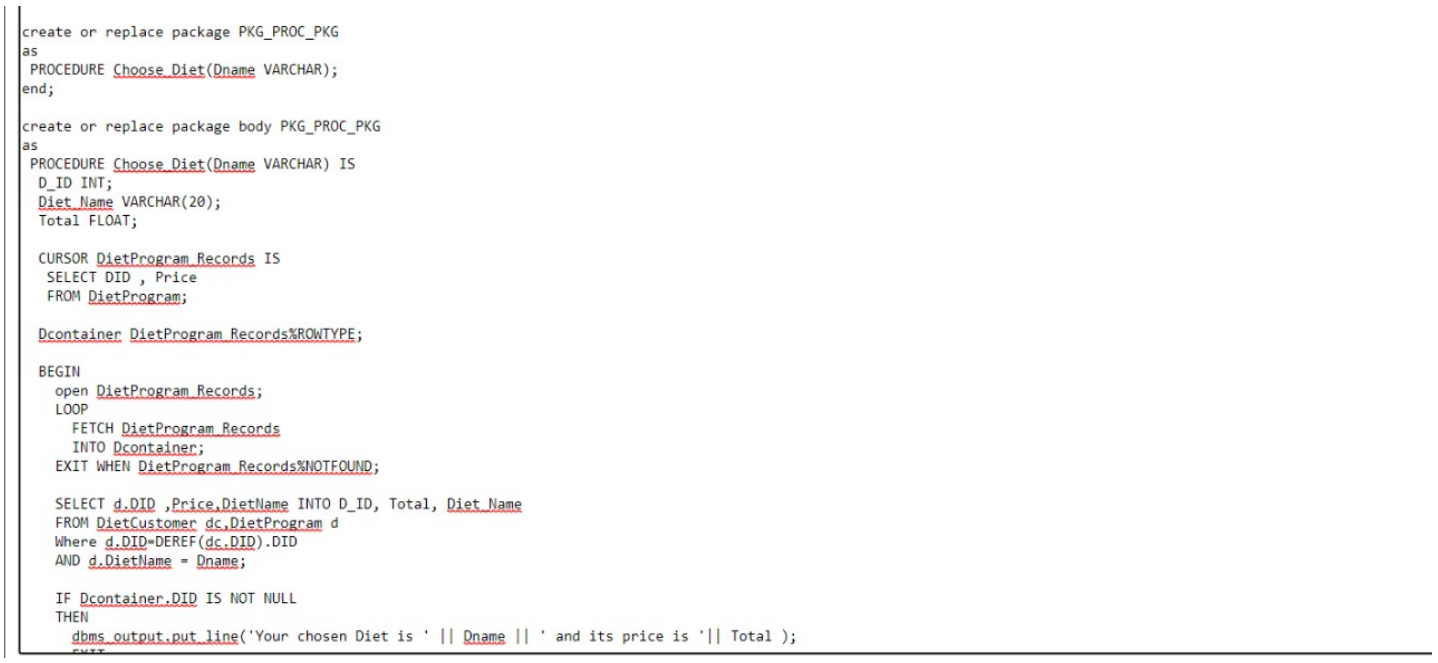


```
        dbms_output.put_line ('You changed your subscription type to: Premium');

    ELSE
      UPDATE Subscription S
      SET S.SType = 'Standard'
      where S.SID = ID;

      dbms_output.put_line ('You changed your subscription type to: Standard');
      END IF;
    END IF;
  END;
END;

----- Try it -----
DECLARE
sub SubscrIptionT;
BEGIN
sub := SubscrIptionT('100', 'premium', NULL);
sub.Choose_Package('Standard');
END;
```

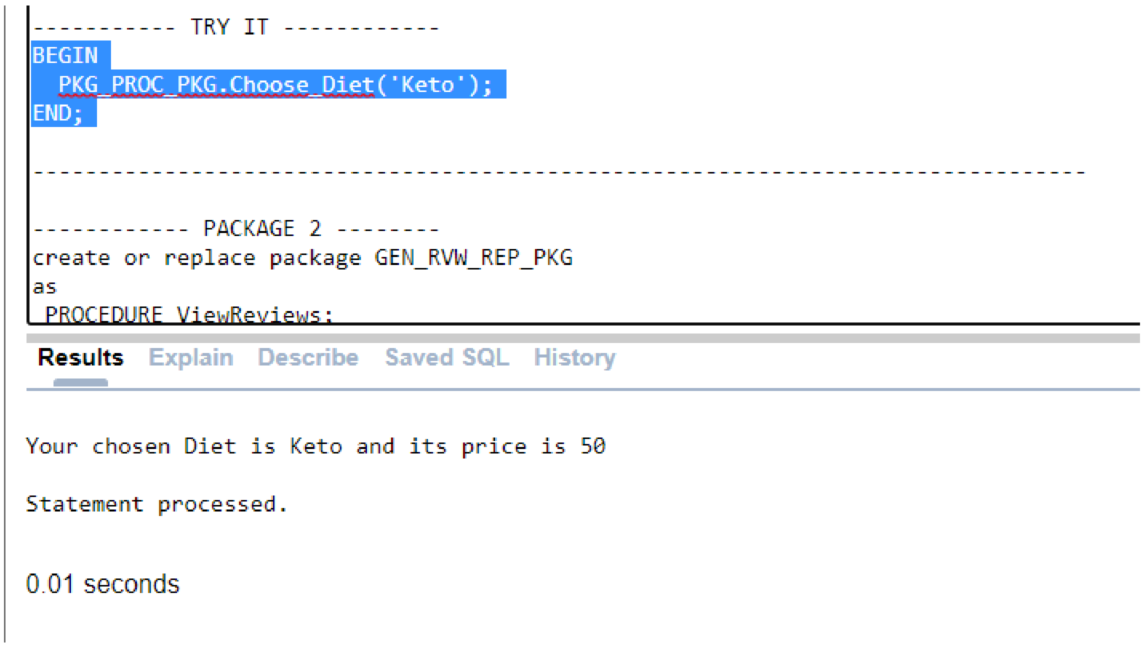
Results Explain Describe Saved SQL History

```
Your current subscription type is: Standard
You already subscribed to Standard

Statement processed.
```

4) **Choose Diet:** The customer chooses his diet by searching for its name. If it's found, the diet name and its price will be displayed. Else the customer will be notified that the searched diet is not found. **(Procedure 2)**



```
create or replace package PKG_PROC_PKG
as
 PROCEDURE Choose_Diet(Dname VARCHAR);
end;

create or replace package body PKG_PROC_PKG
as
 PROCEDURE Choose_Diet(Dname VARCHAR) IS
  D_ID INT;
  Diet_Name VARCHAR(20);
  Total FLOAT;

  CURSOR DietProgram_Records IS
   SELECT DID , Price
   FROM DietProgram;

  Dcontainer DietProgram_Records%ROWTYPE;

  BEGIN
    open DietProgram_Records;
    LOOP
      FETCH DietProgram_Records
      INTO Dcontainer;
    EXIT WHEN DietProgram_Records%NOTFOUND;

    SELECT d.DID ,Price,DietName INTO D_ID, Total, Diet_Name
    FROM DietCustomer dc,DietProgram d
    Where d.DID=DEREF(dc.DID).DID
    AND d.DietName = Dname;

    IF Dcontainer.DID IS NOT NULL
    THEN
      dbms_output.put_line('Your chosen Diet is ' || Dname || ' and its price is '|| Total );
```

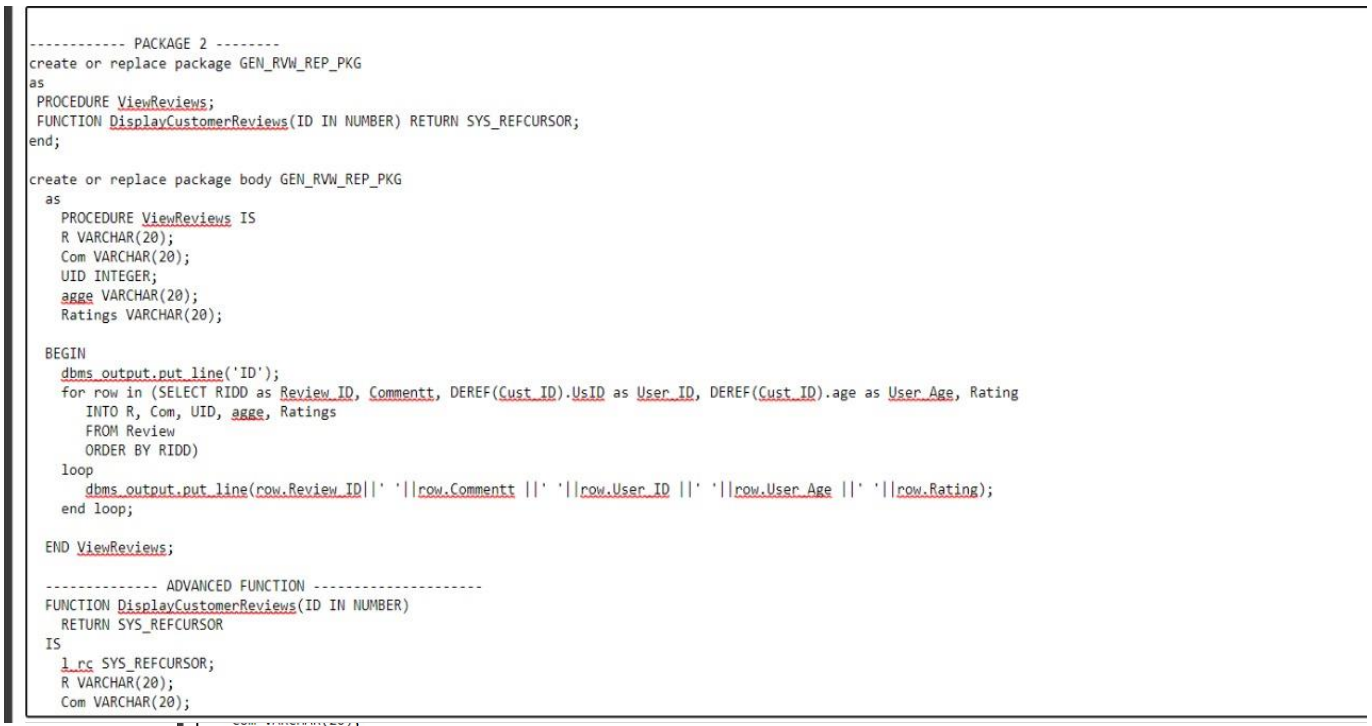


```
----------- TRY IT ------------
BEGIN
  PKG_PROC_PKG.Choose_Diet('Keto');
END;

--------------------------------------------------------------------------------

------------ PACKAGE 2 --------
create or replace package GEN_RVW_REP_PKG
as
 PROCEDURE ViewReviews;
```

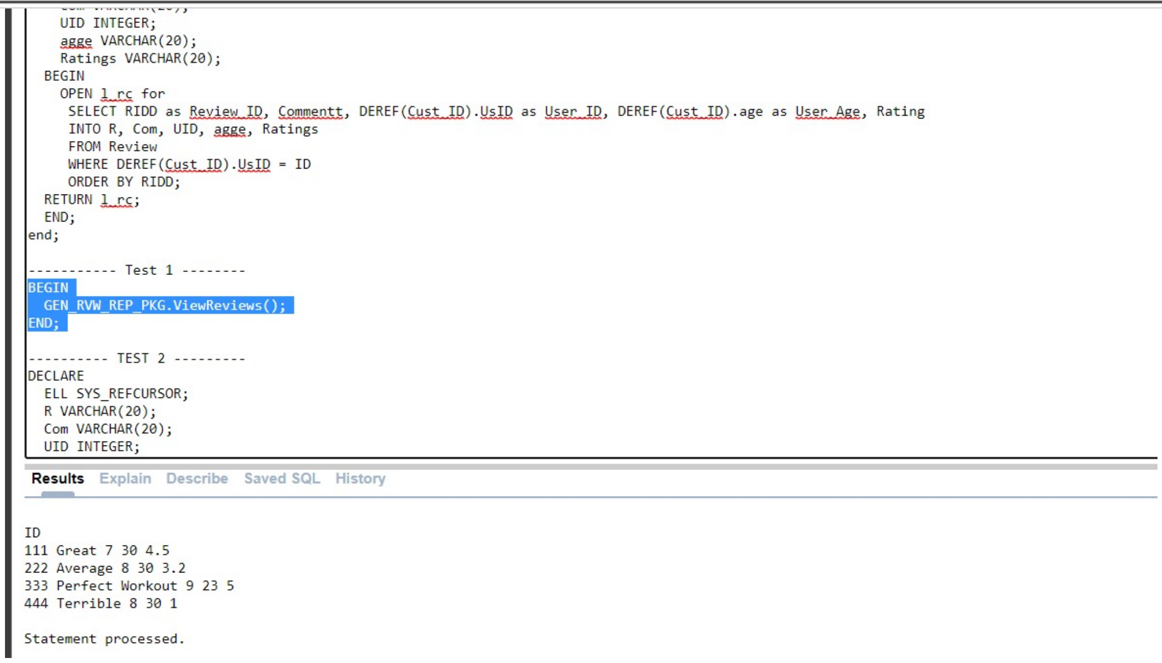
```
Your chosen Diet is Keto and its price is 50

Statement processed.

0.01 seconds
```

Package

5) **View Review:** It views the reviews including the customer id along with his age, comment and rating by searching with the ID number **(Procedure 3)**



```
------------ PACKAGE 2 --------
create or replace package GEN_RVW_REP_PKG
as
 PROCEDURE ViewReviews;
 FUNCTION DisplayCustomerReviews(ID IN NUMBER) RETURN SYS_REFCURSOR;
end;

create or replace package body GEN_RVW_REP_PKG
  as
    PROCEDURE ViewReviews IS
    R VARCHAR(20);
    Com VARCHAR(20);
    UID INTEGER;
    agge VARCHAR(20);
    Ratings VARCHAR(20);

  BEGIN
    dbms_output.put_line('ID');
    for row in (SELECT RIDD as Review_ID, Commentt, DEREF(Cust_ID).UsID as User_ID, DEREF(Cust_ID).age as User_Age, Rating
       INTO R, Com, UID, agge, Ratings
       FROM Review
       ORDER BY RIDD)
    loop
       dbms_output.put_line(row.Review_ID||' '||row.Commentt ||' '||row.User_ID ||' '||row.User_Age ||' '||row.Rating);
    end loop;

  END ViewReviews;

  -------------- ADVANCED FUNCTION ---------------------
  FUNCTION DisplayCustomerReviews(ID IN NUMBER)
    RETURN SYS_REFCURSOR
  IS
    l_rc SYS_REFCURSOR;
    R VARCHAR(20);
    Com VARCHAR(20);
```

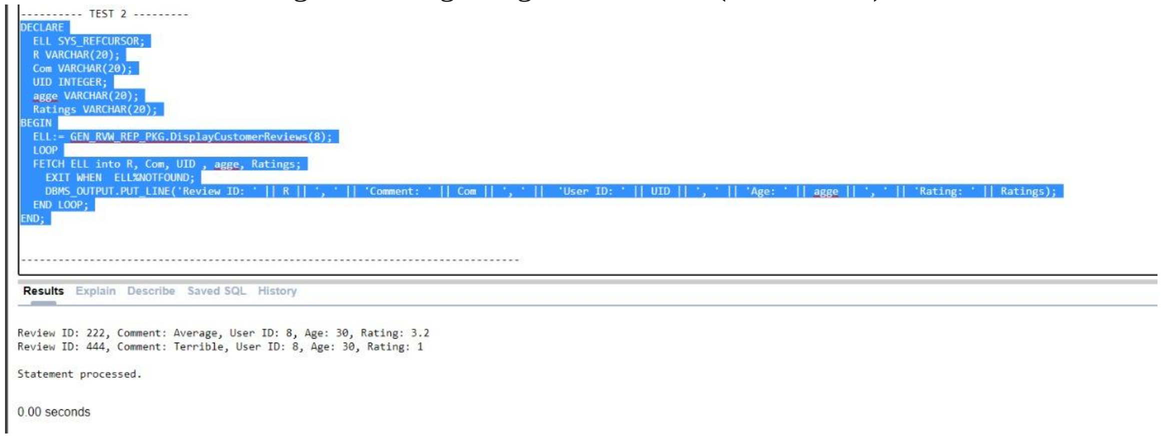


```
    UID INTEGER;
    agge VARCHAR(20);
    Ratings VARCHAR(20);
  BEGIN
    OPEN l_rc for
     SELECT RIDD as Review_ID, Commentt, DEREF(Cust_ID).UsID as User_ID, DEREF(Cust_ID).age as User_Age, Rating
     INTO R, Com, UID, agge, Ratings
     FROM Review
     WHERE DEREF(Cust_ID).UsID = ID
     ORDER BY RIDD;
  RETURN l_rc;
  END;
end;

----------- Test 1 --------
BEGIN
  GEN_RVW_REP_PKG.ViewReviews();
END;

---------- TEST 2 ---------
DECLARE
  ELL SYS_REFCURSOR;
  R VARCHAR(20);
  Com VARCHAR(20);
  UID INTEGER;
```

Results Explain Describe Saved SQL History

```
ID
111 Great 7 30 4.5
222 Average 8 30 3.2
333 Perfect Workout 9 23 5
444 Terrible 8 30 1

Statement processed.
```

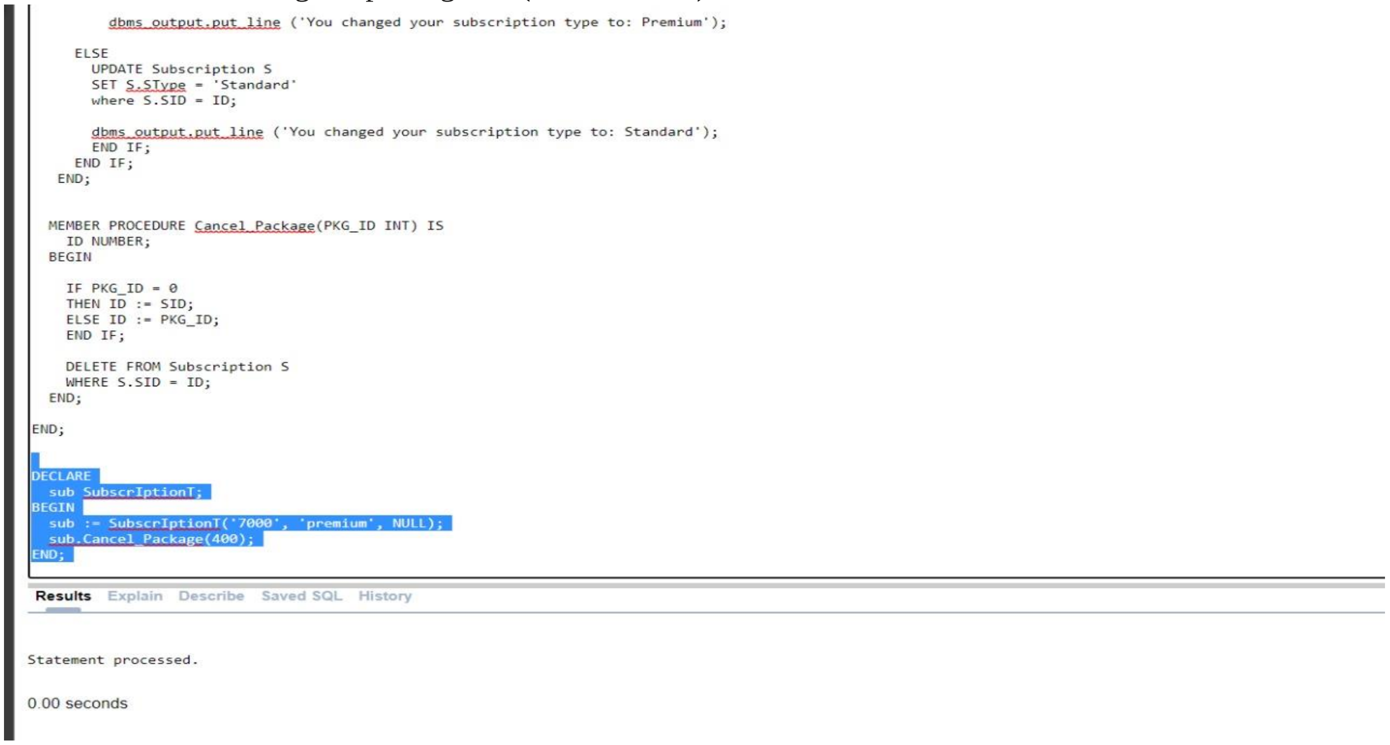
6)Display CustomerReviews : It seraches for all the reviews that were done by a certain customer through searching using Customer ID **(Function 1)**



```
---------- TEST 2 ---------
DECLARE
  ELL SYS_REFCURSOR;
  R VARCHAR(20);
  Com VARCHAR(20);
  UID INTEGER;
  agge VARCHAR(20);
  Ratings VARCHAR(20);
BEGIN
  ELL:= GEN_RVW_REP_PKG.DisplayCustomerReviews(8);
  LOOP
  FETCH ELL into R, Com, UID , agge, Ratings;
    EXIT WHEN  ELL%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Review ID: ' || R || ', ' || 'Comment: ' || Com || ', ' ||  'User ID: ' || UID || ', ' || 'Age: ' || agge || ', ' || 'Rating: ' || Ratings);
  END LOOP;
END;

--------------------------------------------------------------------------------
```

```
Review ID: 222, Comment: Average, User ID: 8, Age: 30, Rating: 3.2
Review ID: 444, Comment: Terrible, User ID: 8, Age: 30, Rating: 1

Statement processed.

0.00 seconds
```

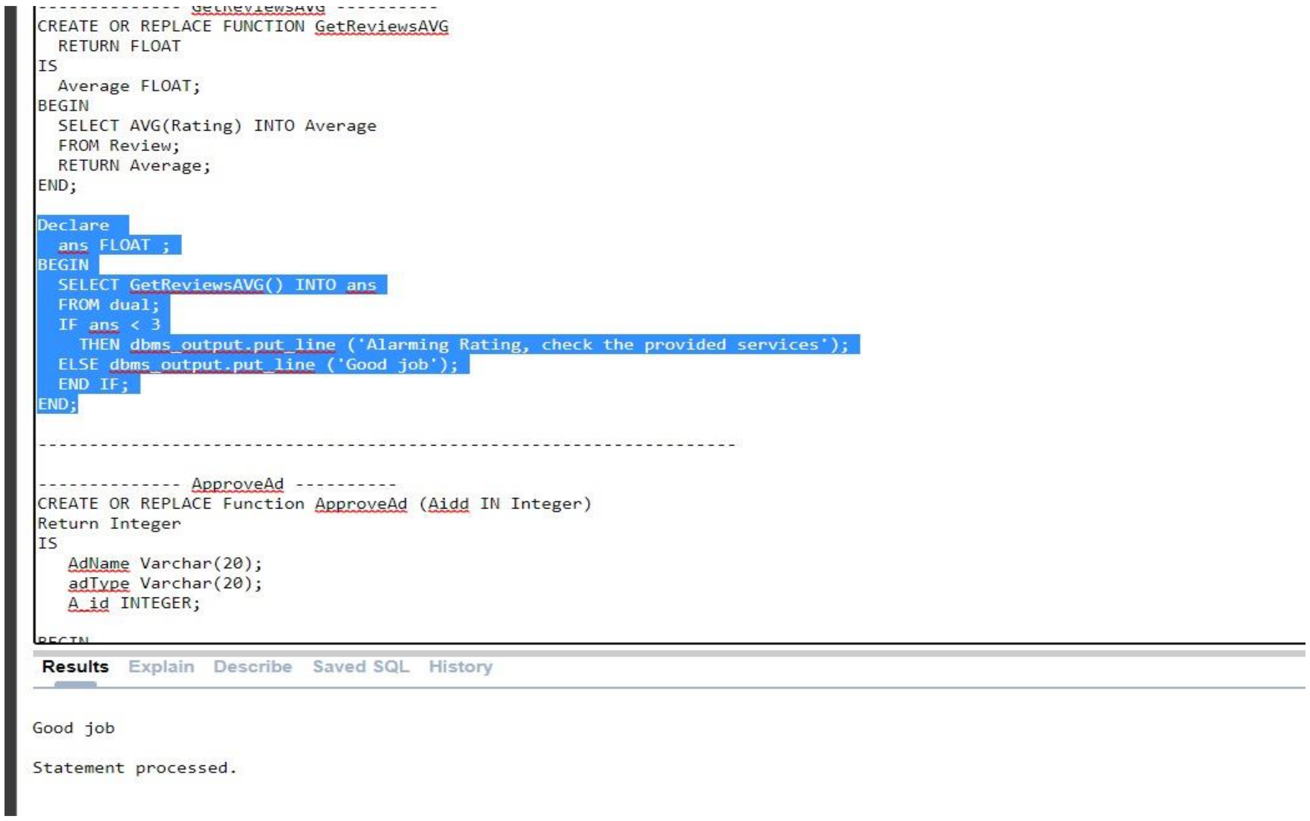
7) **Cancel Package:** The subscription package is cancelled by the customer using the package ID **(Procedure 4)**



```
        dbms_output.put_line ('You changed your subscription type to: Premium');

    ELSE
      UPDATE Subscription S
      SET S.SType = 'Standard'
      where S.SID = ID;

      dbms_output.put_line ('You changed your subscription type to: Standard');
      END IF;
    END IF;
  END;


  MEMBER PROCEDURE Cancel_Package(PKG_ID INT) IS
    ID NUMBER;
  BEGIN

    IF PKG_ID = 0
    THEN ID := SID;
    ELSE ID := PKG_ID;
    END IF;

    DELETE FROM Subscription S
    WHERE S.SID = ID;
  END;

END;


DECLARE
  sub SubscrIptionT;
BEGIN
  sub := SubscrIptionT('7000', 'premium', NULL);
  sub.Cancel_Package(400);
END;
```

```
Statement processed.

0.00 seconds
```

8) **GetReviewsAVG:** It gets the average ratings of the reviews of the customers **(Function2)**

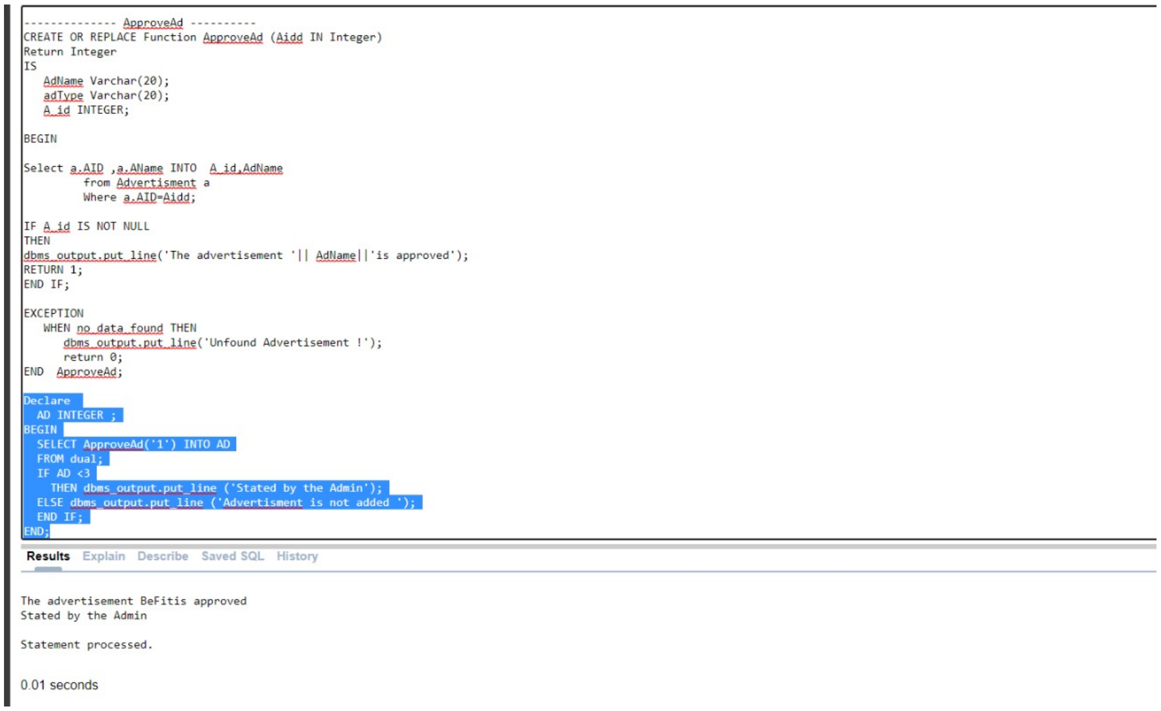


```
CREATE OR REPLACE FUNCTION GetReviewsAVG
  RETURN FLOAT
IS
  Average FLOAT;
BEGIN
  SELECT AVG(Rating) INTO Average
  FROM Review;
  RETURN Average;
END;

Declare
  ans FLOAT ;
BEGIN
  SELECT GetReviewsAVG() INTO ans
  FROM dual;
  IF ans < 3
    THEN dbms_output.put_line ('Alarming Rating, check the provided services');
  ELSE dbms_output.put_line ('Good job');
  END IF;
END;

-------------------------------------------------------------------


-------------- ApproveAd ----------
CREATE OR REPLACE Function ApproveAd (Aidd IN Integer)
Return Integer
IS
   AdName Varchar(20);
   adType Varchar(20);
   A_id INTEGER;
```

```
Good job

Statement processed.
```

9) **ApproveAd: The** advertisement is approved by the Admin if its ID is found in the system, else it needs to get checked again before it got added to the system.

(Function 3)

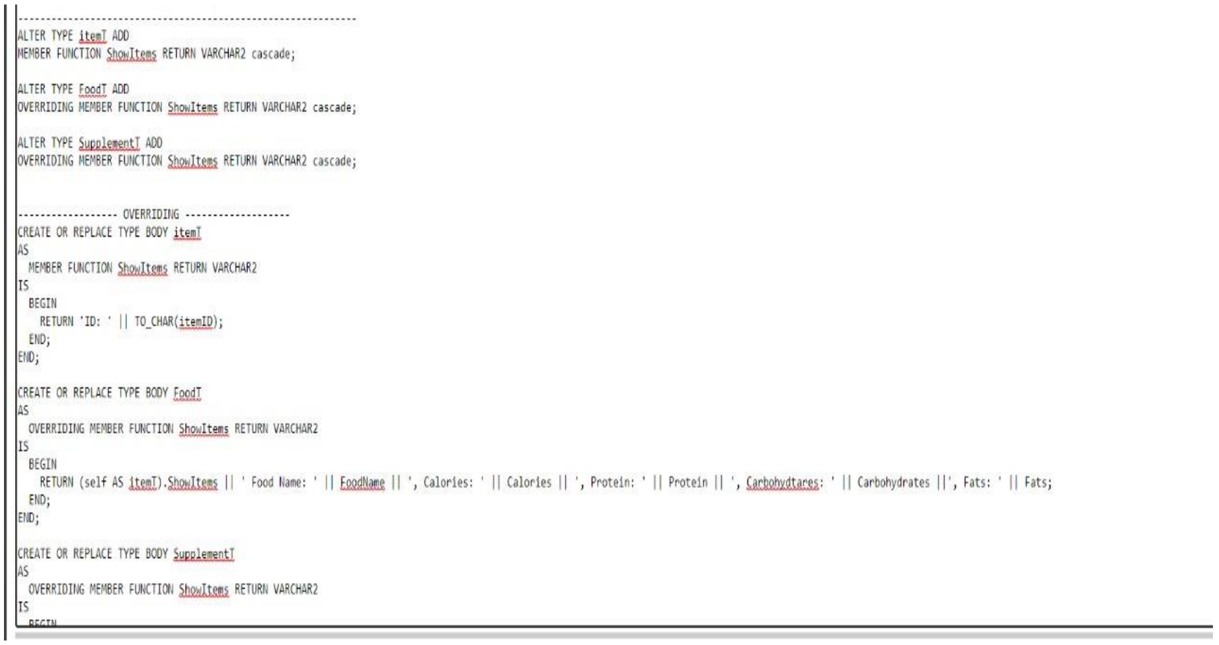


```
-------------- ApproveAd ----------
CREATE OR REPLACE Function ApproveAd (Aidd IN Integer)
Return Integer
IS
   AdName Varchar(20);
   adType Varchar(20);
   A_id INTEGER;

BEGIN

Select a.AID ,a.AName INTO  A_id,AdName
         from Advertisment a
         Where a.AID=Aidd;

IF A_id IS NOT NULL
THEN
dbms_output.put_line('The advertisement '|| AdName||'is approved');
RETURN 1;
END IF;

EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('Unfound Advertisement !');
      return 0;
END  ApproveAd;

Declare
  AD INTEGER ;
BEGIN
  SELECT ApproveAd('1') INTO AD
  FROM dual;
  IF AD <3
    THEN dbms_output.put_line ('Stated by the Admin');
  ELSE dbms_output.put_line ('Advertisment is not added ');
  END IF;
END;
```

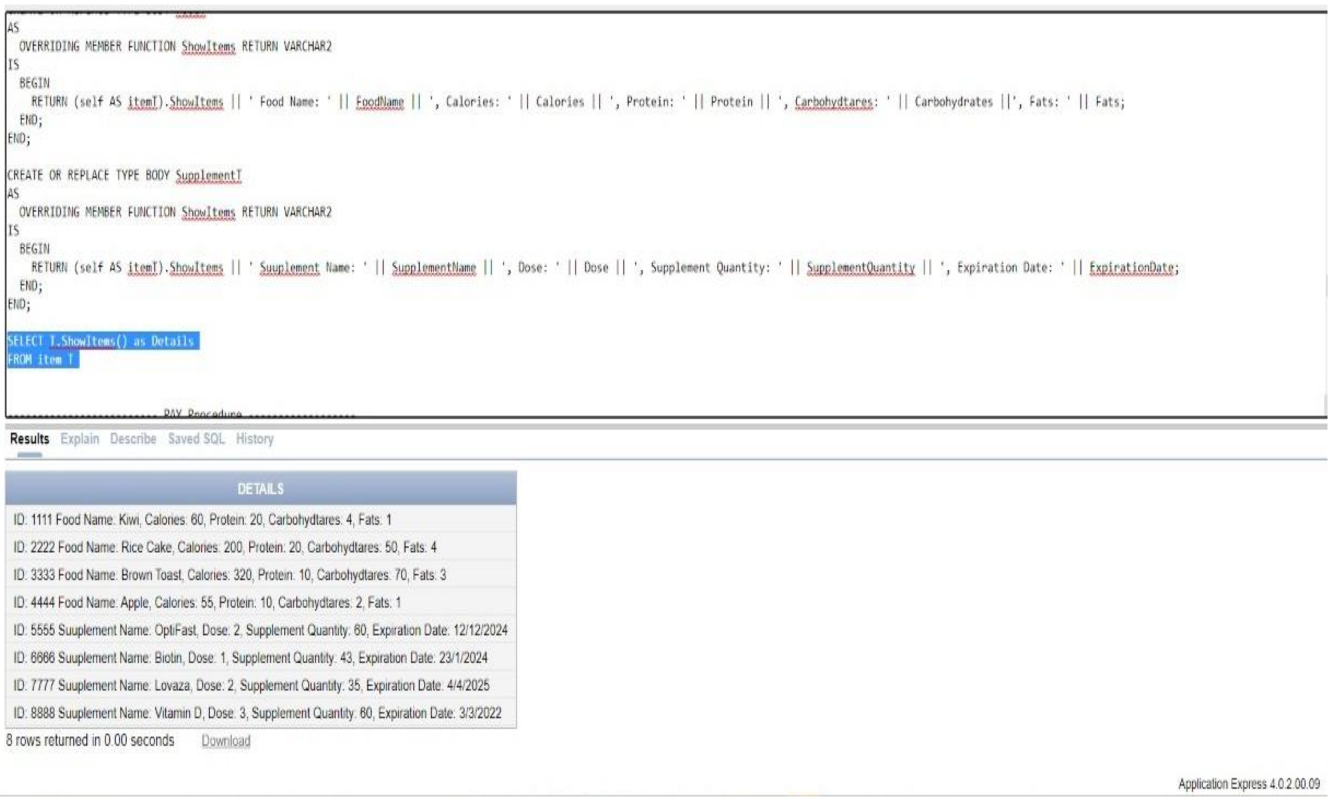
Results Explain Describe Saved SQL History

```
The advertisement BeFitis approved
Stated by the Admin

Statement processed.

0.01 seconds
```

10) **ShowItem:** This function shows item whether if it's food or supplement. The details are shown Food by the food name, calories number, protein portion, carbs portion and fats portion per each food. The details are shown in the supplement by the supplement name , dose, quantity and expiration date per each supplement **(Function 4)**



```
--------------------------------------------------------------
ALTER TYPE itemT ADD
MEMBER FUNCTION ShowItems RETURN VARCHAR2 cascade;

ALTER TYPE FoodT ADD
OVERRIDING MEMBER FUNCTION ShowItems RETURN VARCHAR2 cascade;

ALTER TYPE SupplementT ADD
OVERRIDING MEMBER FUNCTION ShowItems RETURN VARCHAR2 cascade;


----------------- OVERRIDING ------------------
CREATE OR REPLACE TYPE BODY itemT
AS
  MEMBER FUNCTION ShowItems RETURN VARCHAR2
IS
  BEGIN
    RETURN 'ID: ' || TO_CHAR(itemID);
  END;
END;

CREATE OR REPLACE TYPE BODY FoodT
AS
  OVERRIDING MEMBER FUNCTION ShowItems RETURN VARCHAR2
IS
  BEGIN
    RETURN (self AS itemT).ShowItems || ' Food Name: ' || FoodName || ', Calories: ' || Calories || ', Protein: ' || Protein || ', Carbohydtares: ' || Carbohydrates ||', Fats: ' || Fats;
  END;
END;

CREATE OR REPLACE TYPE BODY SupplementT
AS
  OVERRIDING MEMBER FUNCTION ShowItems RETURN VARCHAR2
IS
  BEGIN
```

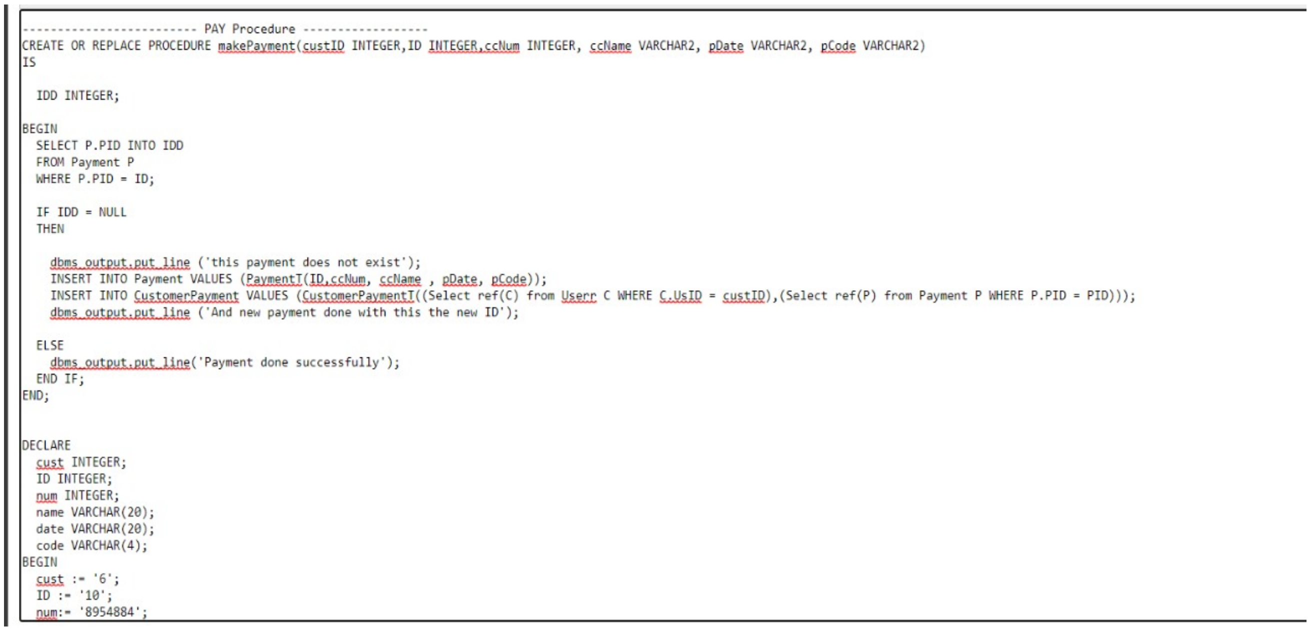


```
AS
  OVERRIDING MEMBER FUNCTION ShowItems RETURN VARCHAR2
IS
  BEGIN
    RETURN (self AS itemT).ShowItems || ' Food Name: ' || FoodName || ', Calories: ' || Calories || ', Protein: ' || Protein || ', Carbohydtares: ' || Carbohydrates ||', Fats: ' || Fats;
  END;
END;

CREATE OR REPLACE TYPE BODY SupplementT
AS
  OVERRIDING MEMBER FUNCTION ShowItems RETURN VARCHAR2
IS
  BEGIN
    RETURN (self AS itemT).ShowItems || ' Suuplement Name: ' || SupplementName || ', Dose: ' || Dose || ', Supplement Quantity: ' || SupplementQuantity || ', Expiration Date: ' || ExpirationDate;
  END;
END;

SELECT T.ShowItems() as Details
FROM item T

------------------------ PAY Procedure -----------------
```

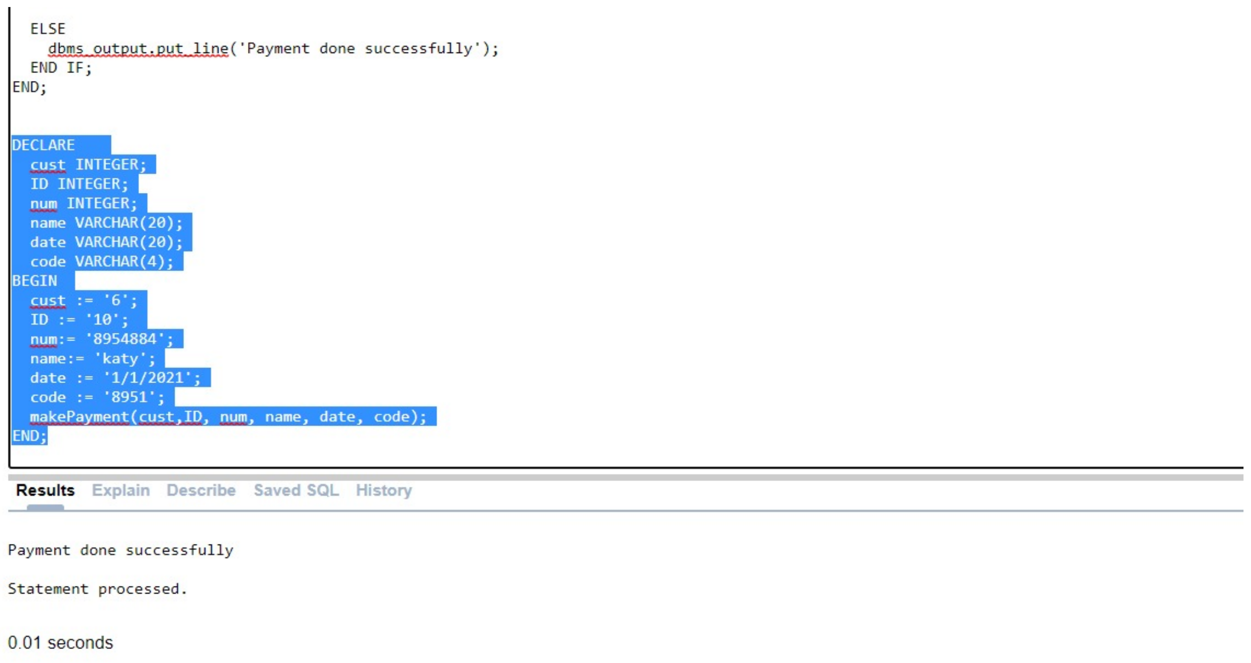
DETAILS
ID: 1111 Food Name: Kiwi, Calories: 60, Protein: 20, Carbohydtares: 4, Fats: 1
ID: 2222 Food Name: Rice Cake, Calories: 200, Protein: 20, Carbohydtares: 50, Fats: 4
ID: 3333 Food Name: Brown Toast, Calories: 320, Protein: 10, Carbohydtares: 70, Fats: 3
ID: 4444 Food Name: Apple, Calories: 55, Protein: 10, Carbohydtares: 2, Fats: 1
ID: 5555 Suuplement Name: OptiFast, Dose: 2, Supplement Quantity: 60, Expiration Date: 12/12/2024
ID: 6666 Suuplement Name: Biotin, Dose: 1, Supplement Quantity: 43, Expiration Date: 23/1/2024
ID: 7777 Suuplement Name: Lovaza, Dose: 2, Supplement Quantity: 35, Expiration Date: 4/4/2025
ID: 8888 Suuplement Name: Vitamin D, Dose: 3, Supplement Quantity: 60, Expiration Date: 3/3/2022

8 rows returned in 0.00 seconds

11) **MakePayment:** The customer makes payment by entering the payment id for each customer, and it checks if the payment existed or not. If the customer didn't make a payment before to the system then he will be added to the system **(Procedure 5)**



```
------------------------ PAY Procedure -----------------
CREATE OR REPLACE PROCEDURE makePayment(custID INTEGER,ID INTEGER,ccNum INTEGER, ccName VARCHAR2, pDate VARCHAR2, pCode VARCHAR2)
IS

  IDD INTEGER;

BEGIN
  SELECT P.PID INTO IDD
  FROM Payment P
  WHERE P.PID = ID;

  IF IDD = NULL
  THEN

    dbms_output.put_line ('this payment does not exist');
    INSERT INTO Payment VALUES (PaymentT(ID,ccNum, ccName , pDate, pCode));
    INSERT INTO CustomerPayment VALUES (CustomerPaymentT((Select ref(C) from Userr C WHERE C.UsID = custID),(Select ref(P) from Payment P WHERE P.PID = PID)));
    dbms_output.put_line ('And new payment done with this the new ID');

  ELSE
    dbms_output.put_line('Payment done successfully');
  END IF;
END;


DECLARE
  cust INTEGER;
  ID INTEGER;
  num INTEGER;
  name VARCHAR(20);
  date VARCHAR(20);
  code VARCHAR(4);
BEGIN
  cust := '6';
  ID := '10';
  num:= '8954884';
```



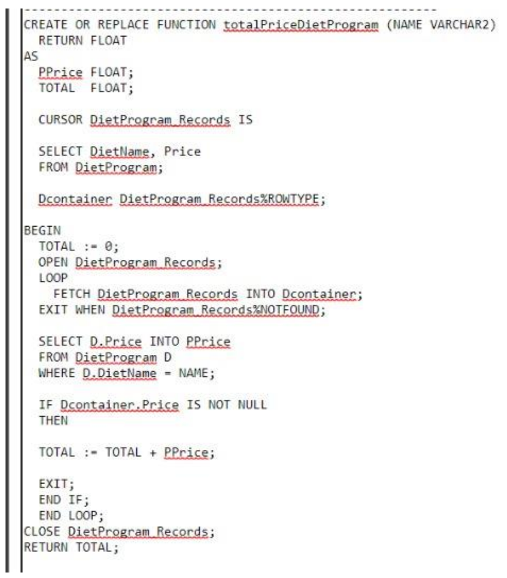
```
  ELSE
    dbms_output.put_line('Payment done successfully');
  END IF;
END;


DECLARE
  cust INTEGER;
  ID INTEGER;
  num INTEGER;
  name VARCHAR(20);
  date VARCHAR(20);
  code VARCHAR(4);
BEGIN
  cust := '6';
  ID := '10';
  num:= '8954884';
  name:= 'katy';
  date := '1/1/2021';
  code := '8951';
  makePayment(cust,ID, num, name, date, code);
END;
```

Results   Explain   Describe   Saved SQL   History

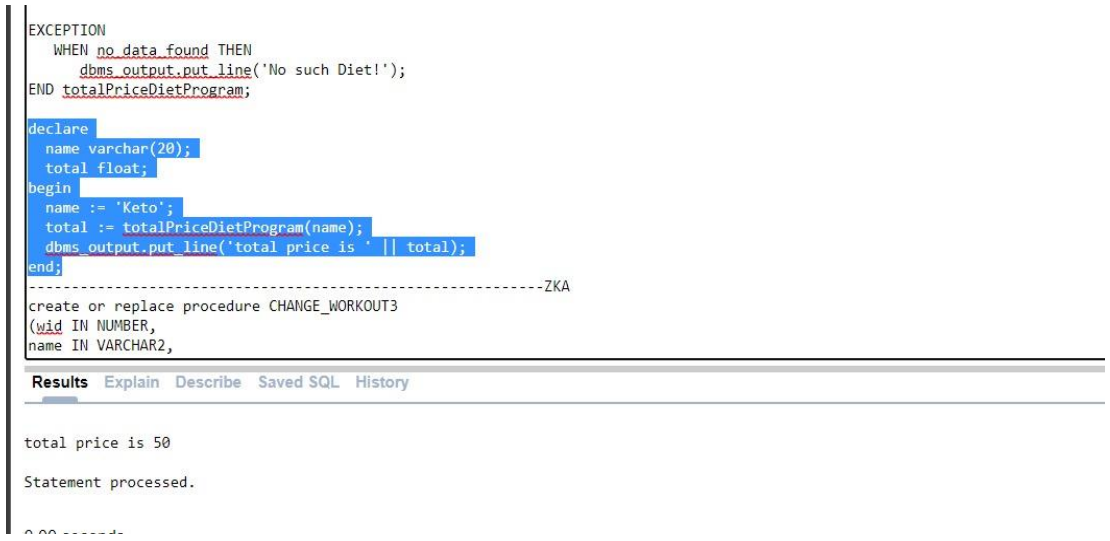
```
Payment done successfully

Statement processed.
```

0.01 seconds

12) TotalPriceDietProgram : It calculates the diet program's total by searching its name **(Function 5)**



```
--------------------------------------------------------
CREATE OR REPLACE FUNCTION totalPriceDietProgram (NAME VARCHAR2)
  RETURN FLOAT
AS
  PPrice FLOAT;
  TOTAL  FLOAT;

  CURSOR DietProgram_Records IS

  SELECT DietName, Price
  FROM DietProgram;

  Dcontainer DietProgram_Records%ROWTYPE;

BEGIN
  TOTAL := 0;
  OPEN DietProgram_Records;
  LOOP
    FETCH DietProgram_Records INTO Dcontainer;
  EXIT WHEN DietProgram_Records%NOTFOUND;

  SELECT D.Price INTO PPrice
  FROM DietProgram D
  WHERE D.DietName = NAME;

  IF Dcontainer.Price IS NOT NULL
  THEN

  TOTAL := TOTAL + PPrice;

  EXIT;
  END IF;
  END LOOP;
CLOSE DietProgram_Records;
RETURN TOTAL;
```

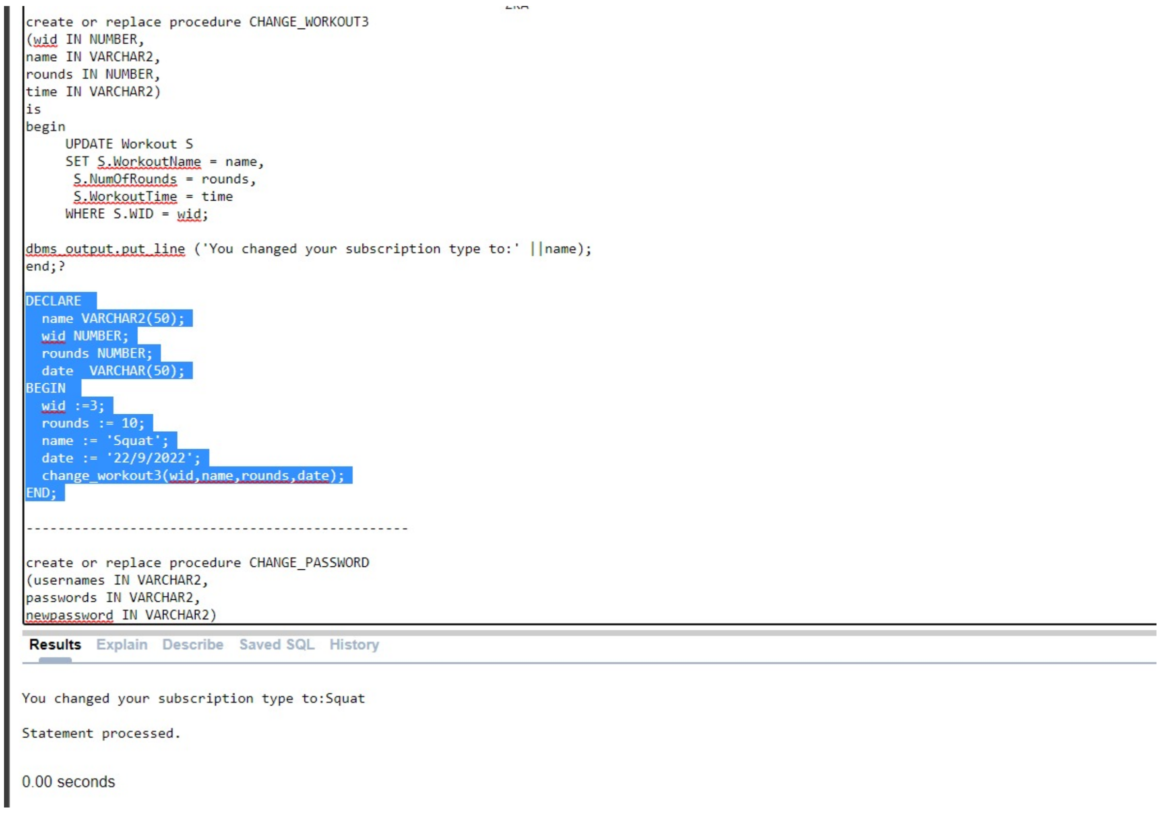


```
EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('No such Diet!');
END totalPriceDietProgram;

declare
  name varchar(20);
  total float;
begin
  name := 'Keto';
  total := totalPriceDietProgram(name);
  dbms_output.put_line('total price is ' || total);
end;
-----------------------------------------------------------ZKA
create or replace procedure CHANGE_WORKOUT3
(wid IN NUMBER,
name IN VARCHAR2,
```

Results Explain Describe Saved SQL History

```
total price is 50

Statement processed.
```

13) ChangeWorkout: The workout is changed by the coach in which he updates the workout information including name , number of rounds and workout time **(Procedure 6)**

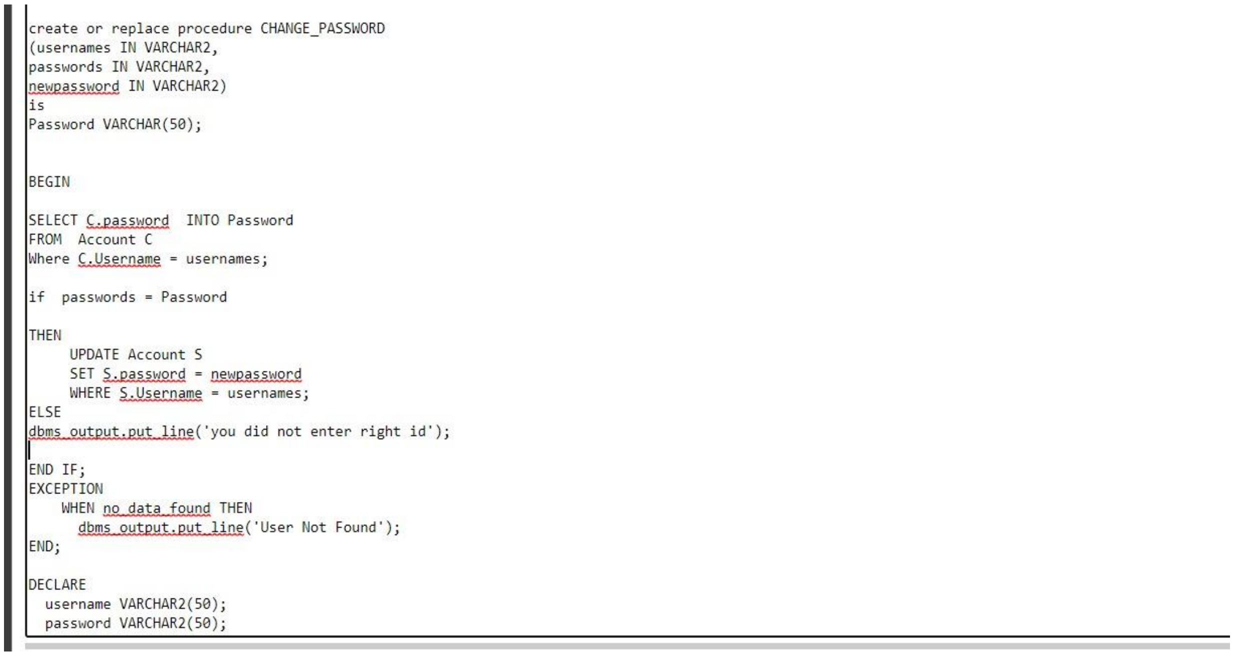


```
create or replace procedure CHANGE_WORKOUT3
(wid IN NUMBER,
name IN VARCHAR2,
rounds IN NUMBER,
time IN VARCHAR2)
is
begin
     UPDATE Workout S
     SET S.WorkoutName = name,
      S.NumOfRounds = rounds,
      S.WorkoutTime = time
     WHERE S.WID = wid;

dbms_output.put_line ('You changed your subscription type to:' ||name);
end;?

DECLARE
  name VARCHAR2(50);
  wid NUMBER;
  rounds NUMBER;
  date  VARCHAR(50);
BEGIN
  wid :=3;
  rounds := 10;
  name := 'Squat';
  date := '22/9/2022';
  change_workout3(wid,name,rounds,date);
END;

-----------------------------------------------

create or replace procedure CHANGE_PASSWORD
(usernames IN VARCHAR2,
passwords IN VARCHAR2,
newpassword IN VARCHAR2)
```

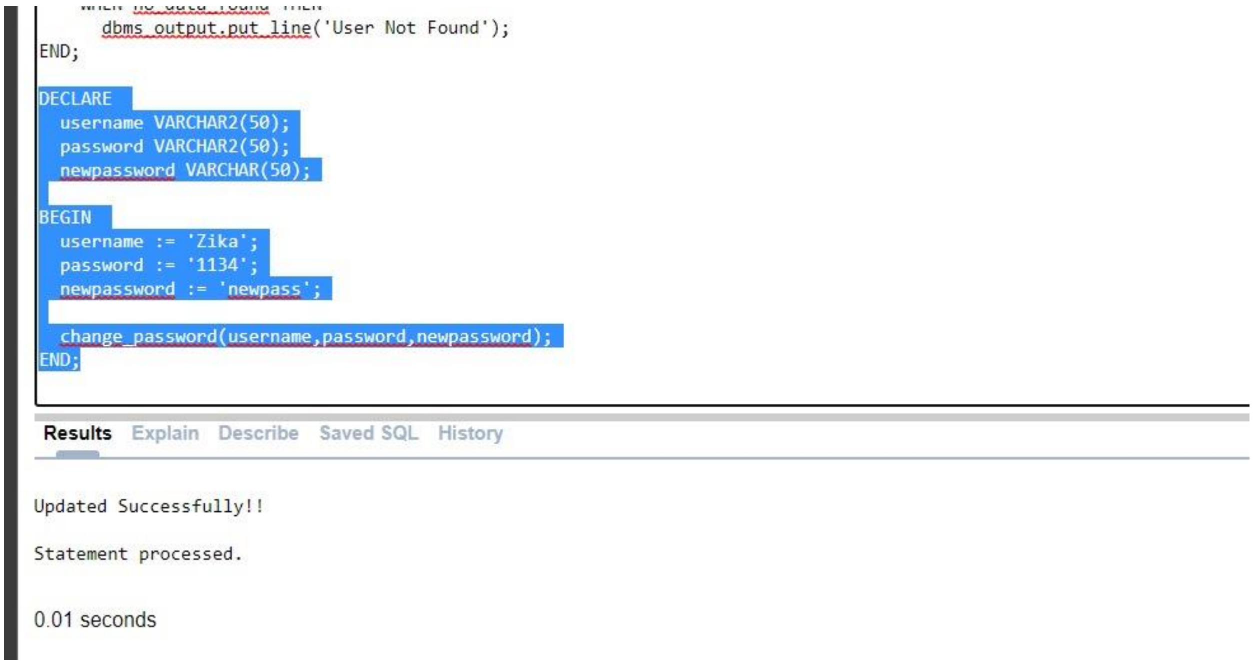
Results Explain Describe Saved SQL History

```
You changed your subscription type to:Squat

Statement processed.

0.00 seconds
```

14) **Change Password:** The user whether admin, customer or coach can change their password by inserting their current username and old password to enter their new password. If the username or old password are not correct then the password will not get changed. **(Procedure 7)**



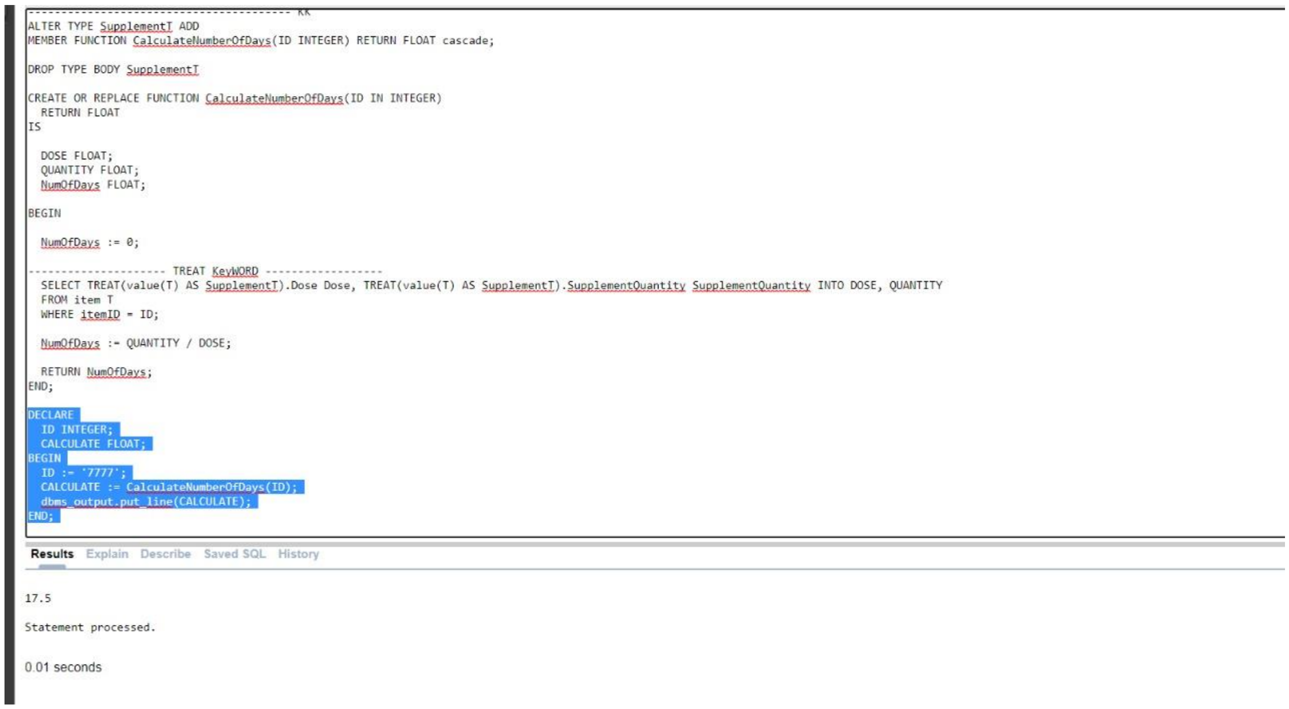
```
create or replace procedure CHANGE_PASSWORD
(usernames IN VARCHAR2,
passwords IN VARCHAR2,
newpassword IN VARCHAR2)
is
Password VARCHAR(50);


BEGIN

SELECT C.password  INTO Password
FROM  Account C
Where C.Username = usernames;

if  passwords = Password

THEN
     UPDATE Account S
     SET S.password = newpassword
     WHERE S.Username = usernames;
ELSE
dbms_output.put_line('you did not enter right id');

END IF;
EXCEPTION
    WHEN no_data_found THEN
      dbms_output.put_line('User Not Found');
END;

DECLARE
  username VARCHAR2(50);
  password VARCHAR2(50);
```



```
      dbms_output.put_line('User Not Found');
END;

DECLARE
  username VARCHAR2(50);
  password VARCHAR2(50);
  newpassword VARCHAR(50);

BEGIN
  username := 'Zika';
  password := '1134';
  newpassword := 'newpass';

  change_password(username,password,newpassword);
END;
```

```
Updated Successfully!!

Statement processed.

0.01 seconds
```

15) **CalculateNumberOfDays :** The quantity and dose of the supplement are calculated to inform the customer with the number of days to take the supplement. **(Function 6)**

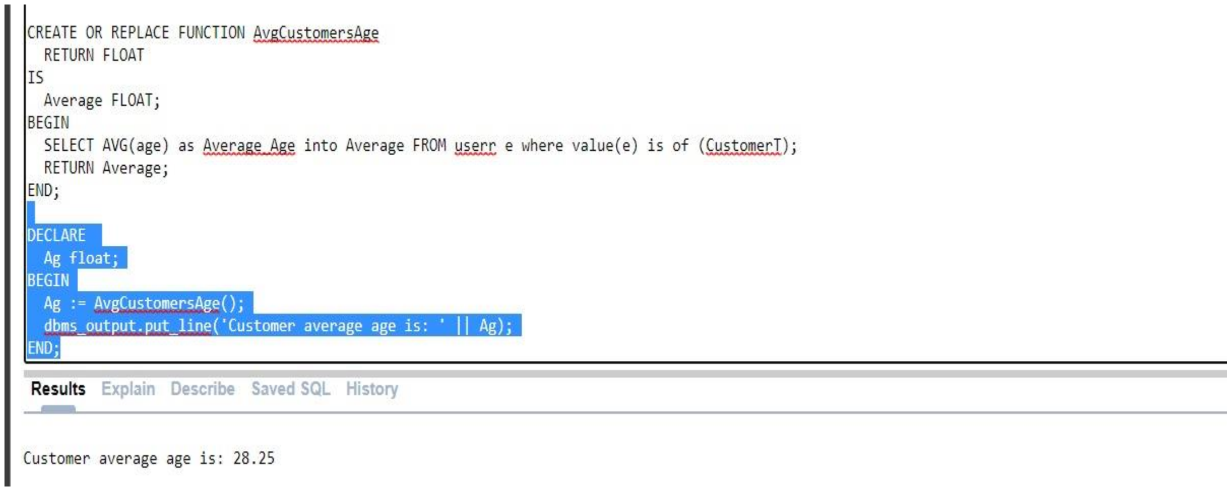


```
------------------------------------------ KK
ALTER TYPE SupplementT ADD
MEMBER FUNCTION CalculateNumberOfDays(ID INTEGER) RETURN FLOAT cascade;

DROP TYPE BODY SupplementT

CREATE OR REPLACE FUNCTION CalculateNumberOfDays(ID IN INTEGER)
  RETURN FLOAT
IS

  DOSE FLOAT;
  QUANTITY FLOAT;
  NumOfDays FLOAT;

BEGIN

  NumOfDays := 0;

-------------------- TREAT KeyWORD -----------------
  SELECT TREAT(value(T) AS SupplementT).Dose Dose, TREAT(value(T) AS SupplementT).SupplementQuantity SupplementQuantity INTO DOSE, QUANTITY
  FROM item T
  WHERE itemID = ID;

  NumOfDays := QUANTITY / DOSE;

  RETURN NumOfDays;
END;

DECLARE
  ID INTEGER;
  CALCULATE FLOAT;
BEGIN
  ID := '7777';
  CALCULATE := CalculateNumberOfDays(ID);
  dbms_output.put_line(CALCULATE);
END;
```

Results Explain Describe Saved SQL History

```
17.5

Statement processed.

0.01 seconds
```

16) **AvgCustomerAge:** The average age of the customers is calculated for the sake of segmenting the target audience of the nutrition system. **(Function 7)**



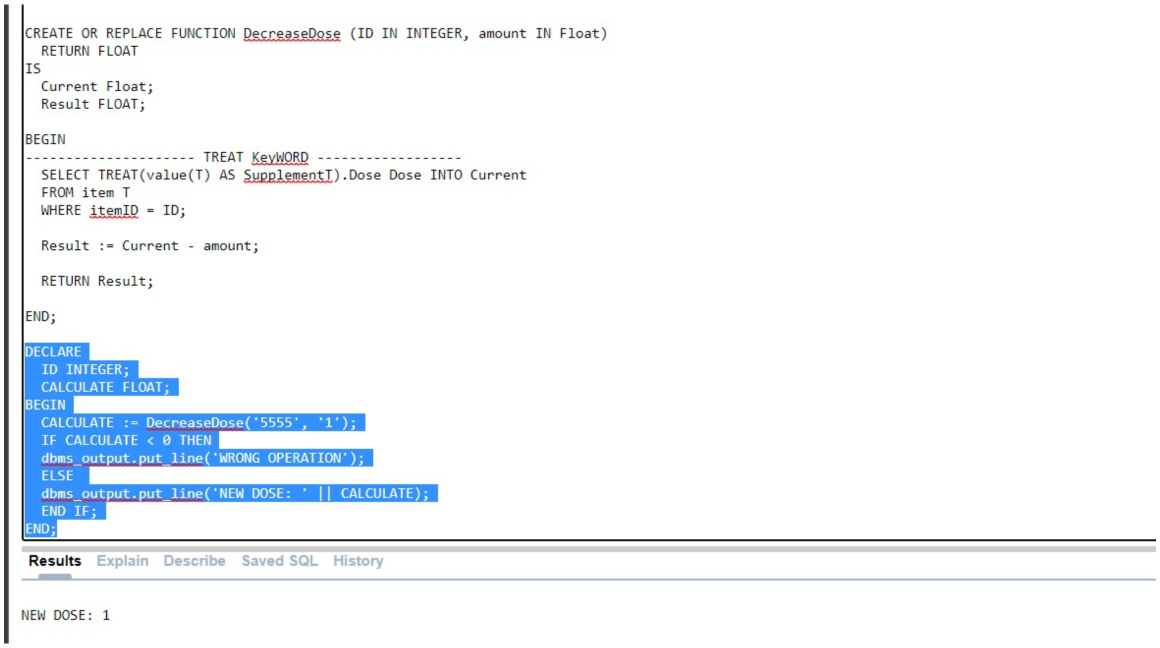
```
CREATE OR REPLACE FUNCTION AvgCustomersAge
  RETURN FLOAT
IS
  Average FLOAT;
BEGIN
  SELECT AVG(age) as Average_Age into Average FROM userr e where value(e) is of (CustomerT);
  RETURN Average;
END;

DECLARE
  Ag float;
BEGIN
  Ag := AvgCustomersAge();
  dbms_output.put_line('Customer average age is: ' || Ag);
END;
```

Results Explain Describe Saved SQL History

```
Customer average age is: 28.25
```

17) DecreaseDose: The supplement dose amount is decreased based on the market demand. In which the customer requests the supplement id and the needed amount.

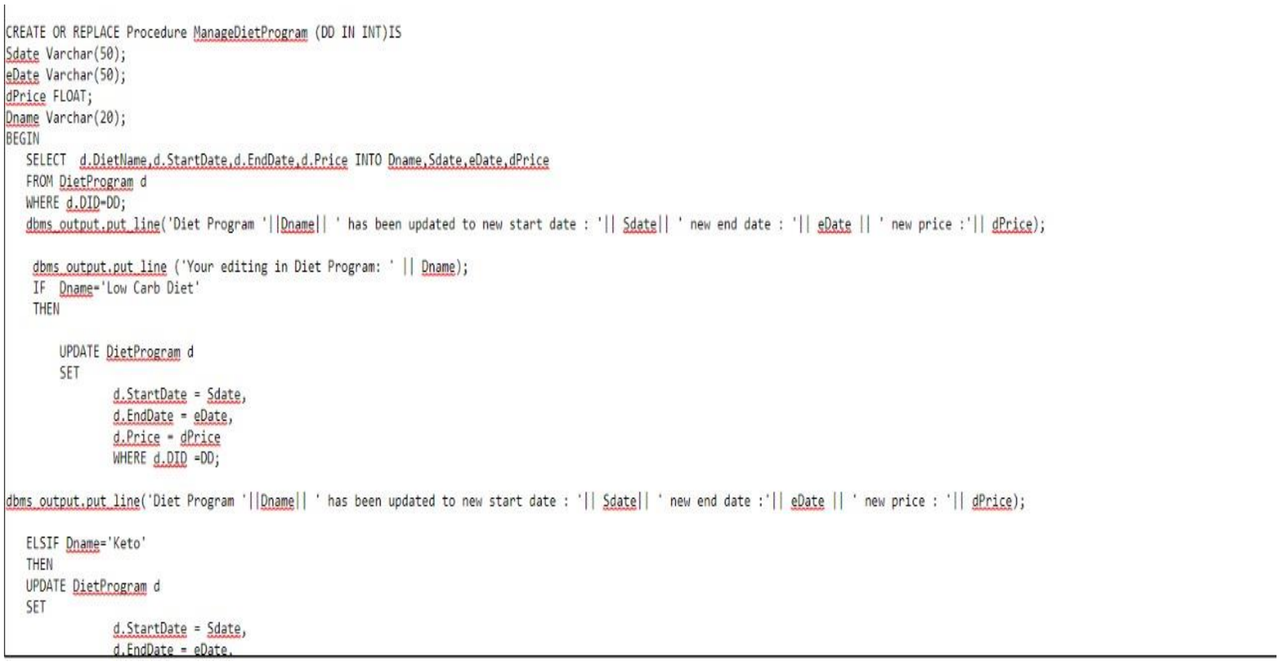
(Function 8)



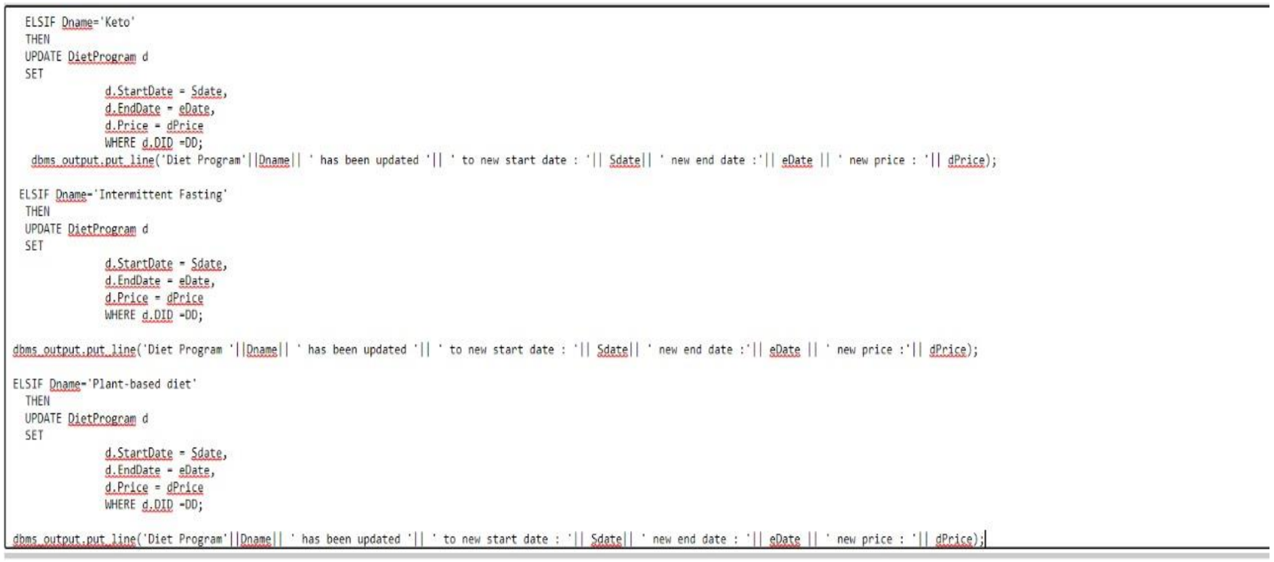
```
CREATE OR REPLACE FUNCTION DecreaseDose (ID IN INTEGER, amount IN Float)
  RETURN FLOAT
IS
  Current Float;
  Result FLOAT;

BEGIN
--------------------- TREAT KeyWORD ------------------
  SELECT TREAT(value(T) AS SupplementT).Dose Dose INTO Current
  FROM item T
  WHERE itemID = ID;

  Result := Current - amount;

  RETURN Result;

END;

DECLARE
  ID INTEGER;
  CALCULATE FLOAT;
BEGIN
  CALCULATE := DecreaseDose('5555', '1');
  IF CALCULATE < 0 THEN
  dbms_output.put_line('WRONG OPERATION');
  ELSE
  dbms_output.put_line('NEW DOSE: ' || CALCULATE);
  END IF;
END;
```

```
NEW DOSE: 1
```

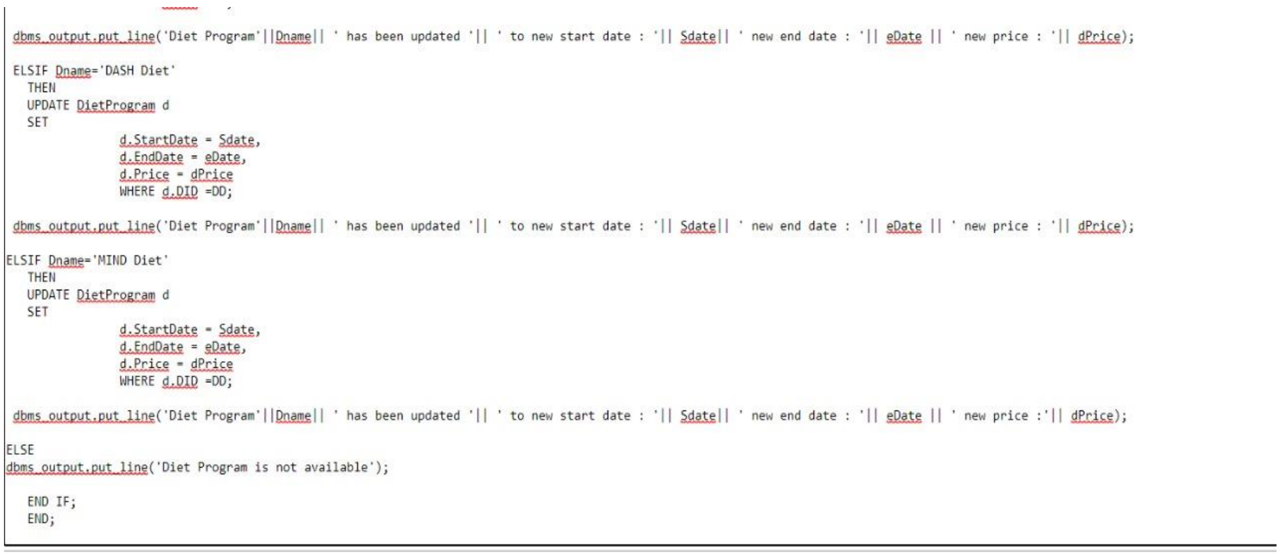
18) ManageDietProgram: The diet program is updated by entering its id then the name will be displayed and the new start date, end date and price will be changed. If the diet program name is not found then a notification will appear that the diet program is not available. **(Procedure 8)**



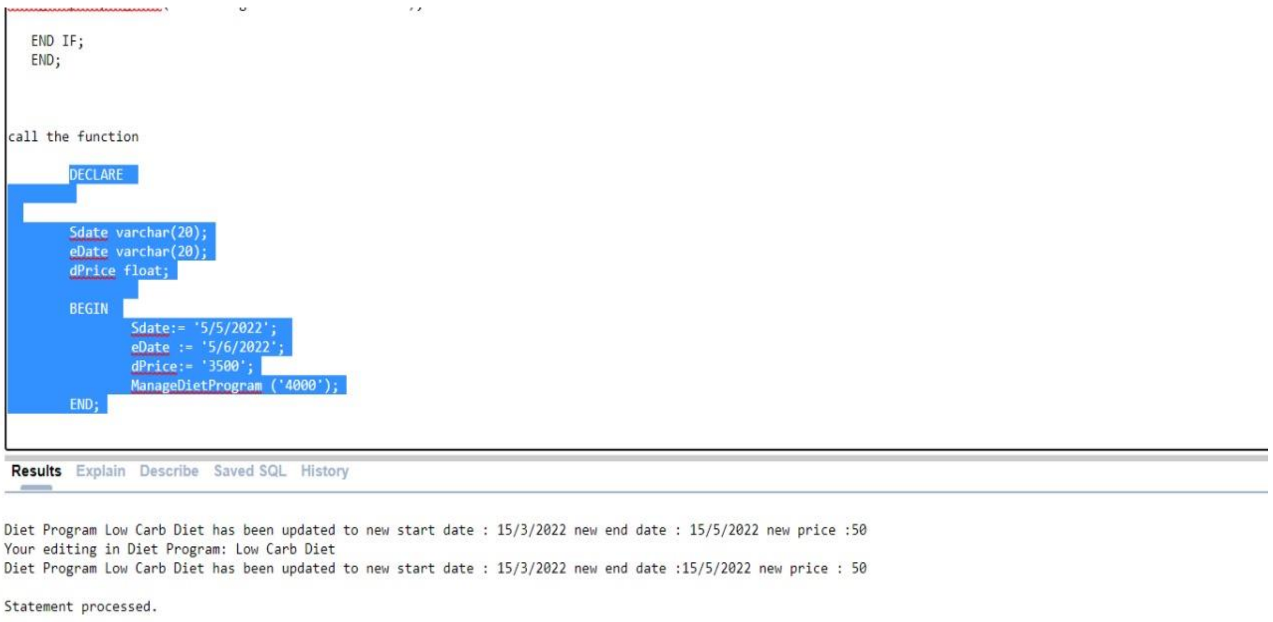
```
CREATE OR REPLACE Procedure ManageDietProgram (DD IN INT)IS
Sdate Varchar(50);
eDate Varchar(50);
dPrice FLOAT;
Dname Varchar(20);
BEGIN
   SELECT  d.DietName,d.StartDate,d.EndDate,d.Price INTO Dname,Sdate,eDate,dPrice
   FROM DietProgram d
   WHERE d.DID=DD;
   dbms_output.put_line('Diet Program '||Dname|| ' has been updated to new start date : '|| Sdate|| ' new end date : '|| eDate || ' new price :'|| dPrice);

    dbms_output.put_line ('Your editing in Diet Program: ' || Dname);
    IF  Dname='Low Carb Diet'
    THEN

        UPDATE DietProgram d
        SET
                d.StartDate = Sdate,
                d.EndDate = eDate,
                d.Price = dPrice
                WHERE d.DID =DD;

dbms_output.put_line('Diet Program '||Dname|| ' has been updated to new start date : '|| Sdate|| ' new end date :'|| eDate || ' new price : '|| dPrice);

   ELSIF Dname='Keto'
   THEN
   UPDATE DietProgram d
   SET
                d.StartDate = Sdate,
                d.EndDate = eDate,
```



```
  ELSIF Dname='Keto'
  THEN
  UPDATE DietProgram d
  SET
             d.StartDate = Sdate,
             d.EndDate = eDate,
             d.Price = dPrice
             WHERE d.DID =DD;
   dbms_output.put_line('Diet Program'||Dname|| ' has been updated '|| ' to new start date : '|| Sdate|| ' new end date :'|| eDate || ' new price : '|| dPrice);

 ELSIF Dname='Intermittent Fasting'
  THEN
  UPDATE DietProgram d
  SET
             d.StartDate = Sdate,
             d.EndDate = eDate,
             d.Price = dPrice
             WHERE d.DID =DD;

dbms_output.put_line('Diet Program '||Dname|| ' has been updated '|| ' to new start date : '|| Sdate|| ' new end date :'|| eDate || ' new price :'|| dPrice);

ELSIF Dname='Plant-based diet'
  THEN
  UPDATE DietProgram d
  SET
             d.StartDate = Sdate,
             d.EndDate = eDate,
             d.Price = dPrice
             WHERE d.DID =DD;

dbms_output.put_line('Diet Program'||Dname|| ' has been updated '|| ' to new start date : '|| Sdate|| ' new end date : '|| eDate || ' new price : '|| dPrice);
```



```
dbms_output.put_line('Diet Program'||Dname|| ' has been updated '|| ' to new start date : '|| Sdate|| ' new end date : '|| eDate || ' new price : '|| dPrice);

ELSIF Dname='DASH Diet'
  THEN
  UPDATE DietProgram d
  SET
             d.StartDate = Sdate,
             d.EndDate = eDate,
             d.Price = dPrice
             WHERE d.DID =DD;

dbms_output.put_line('Diet Program'||Dname|| ' has been updated '|| ' to new start date : '|| Sdate|| ' new end date : '|| eDate || ' new price : '|| dPrice);

ELSIF Dname='MIND Diet'
  THEN
  UPDATE DietProgram d
  SET
             d.StartDate = Sdate,
             d.EndDate = eDate,
             d.Price = dPrice
             WHERE d.DID =DD;

dbms_output.put_line('Diet Program'||Dname|| ' has been updated '|| ' to new start date : '|| Sdate|| ' new end date : '|| eDate || ' new price :'|| dPrice);

ELSE
dbms_output.put_line('Diet Program is not available');

  END IF;
  END;
```

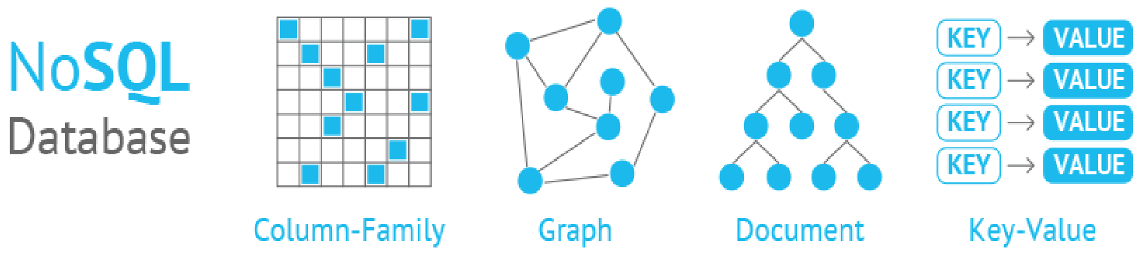

```
  END IF;
  END;


call the function

        DECLARE


        Sdate varchar(20);
        eDate varchar(20);
        dPrice float;

        BEGIN
                Sdate:= '5/5/2022';
                eDate := '5/6/2022';
                dPrice:= '3500';
                ManageDietProgram ('4000');
        END;
```

Results Explain Describe Saved SQL History

```
Diet Program Low Carb Diet has been updated to new start date : 15/3/2022 new end date : 15/5/2022 new price :50
Your editing in Diet Program: Low Carb Diet
Diet Program Low Carb Diet has been updated to new start date : 15/3/2022 new end date :15/5/2022 new price : 50

Statement processed.
```

Research

NoSQL database is a technology for storing information in a form of JSON documents, unlike the relational schema used in our nutrition system database that combines both objects and relational schema that stores the information in columns and rows by applying the object oriented concepts. NoSQL has different types which are document databases, key-value stores, wide column databases and graph databases. Document databases are mainly used in storing documents as information; however the amount of information doesn't depend on the capacity of JSON documents only. It may use XML document or any other type of documents. Key Value stores group of data with their records and for facilitating the retrieval process, each is uniquely identified with the primary keys. Key values supports additional feature in NoSQL, as it combines the benefits of relational database and the benefits of NoSQL. Wide Column databases are used to organize data into columns that can flexibly spread through several servers, by referencing these data with using multi-dimensional mapping where each has column, row and a timestamp. It's known as column family databases, as the whole column will be loaded and searched rapidly, where data is stored in a form of columns. Graph databases are used to describe the relationship which lies between stored data points, where it has the start node, end node, type and direction. It helps in storing and navigating the relationships, as the edge describes the parent child relationships. Considering that the relationships number and kind nodes have no limitations. NoSQL database uses unstructured storage and it is developed for faster, simpler, wider data, frequently application changes, and it is much easier for programming developers. While the major cause of using it, is for distributed data where data storage will be divided across several systems.



NoSQL database’s benefits are focusing on flexibility, scalability and high performance. Flexibility in which the data is stored in free form without using the rigid schema, causing rapid development in applications. Structured, semi-structured, and non-structured data in a single data store could be found in NoSQL database. Scalability is by depending on horizontally and vertically scaling. Sharding, which is a process used to scale up horizontally, respond to multiple machines which will be added to handle multiple servers. Horizontal scaling handles large amounts of data efficiently. While vertically scaling evolve datasets by becoming larger and powerful, unfortunately it can be unsustainable when more storage is needed. High performance is used when the data volume increases, NoSQL database will respond rapidly and deliver data reliably, in addition to the highly interactive user experience.

NoSQL database has drawback as well, as it depends on de-normalized data that supports the types of applications that uses documents and doesn’t have relationships with referencing models. Therefore, it supports some applications which uses less tables. However; some businesses application depend on normalizing data storage to prevent anomalies and data duplication. NoSQL database supports complex queries, in addition, it doesn’t use joins, nested queries, sub queries and WHERE clause, which makes the relational database to be a better option to process queries. Therefore, companies use hybrid approach that combined both NoSQL and relational database, in order to increase the flexibility and ensure the consistency of transactions’ performance.

Cassandra is an open source. It’s used for distributed or decentralized storage. It manages a huge amount of structured data across the world. It provides the non-failure architecture with highly available service. Cassandra is scalable, consistent and fault tolerant. It’s a column oriented database. Amazon’s Dynamo is the basic of Cassandra’s distribution design and Google’s Bigtable’s data model. It adds more powerful column family data model. Cassandra is used by the big companies such as Facebook, Twitter, Cisco, eBay, Netflix and more. Cassandra’s features are elastic scalability, always on architecture, fast linear scale performance, flexible data storage, easy data storage, easy data distribution, transaction support and fast writes. Elastic scalability accommodate more customers and more data by allowing more hardware components to participate. Cassandra doesn’t afford failure. It accepts all data formats which are structured, semi-structured, and unstructured, and it accepts changes in the data structure according to the need. Cassandra replicates data across several data centers. It can run efficiently on cheap hardware by storing hundreds of terabytes of data without sacrificing the read efficiency.

The difference between Cassandra and Oracle PL/SQL that is written in our project is that Oracle PL/SQL is an originally relational database management system (RDBMS). While Cassandra is Wide Column Store. Cassandra’s subset might be 1 million rows, which makes it capable to handle large scale of data. While our Oracle PL/SQL works on small scale of data hence it is a query language and it can be accessed on major platforms like Windows, UNIX, Linux, and macOS. Unlike Cassandra which is considered as a database tool that has an open source which can be divided across several machines, under NO SQL database. Although Cassandra is distributed, it’s available to users as a unified whole. While our project is in a fixed place that cannot be automatically clustered across machines except that has operational Oracle. Our schema is good for complex queries as it doesn’t take time in processing. However, Cassandra is not recommended for complex queries. PL/SQL will be slowed because a search is being performed before the write, but in Cassandra there is a higher performance during writing as it uses append model.

References

- https://www.couchbase.com/resources/why-nosql
- https://www.oracle.com/database/nosql/what-is-nosql/
- https://www.tutorialspoint.com/cassandra/cassandra_introduction.htm