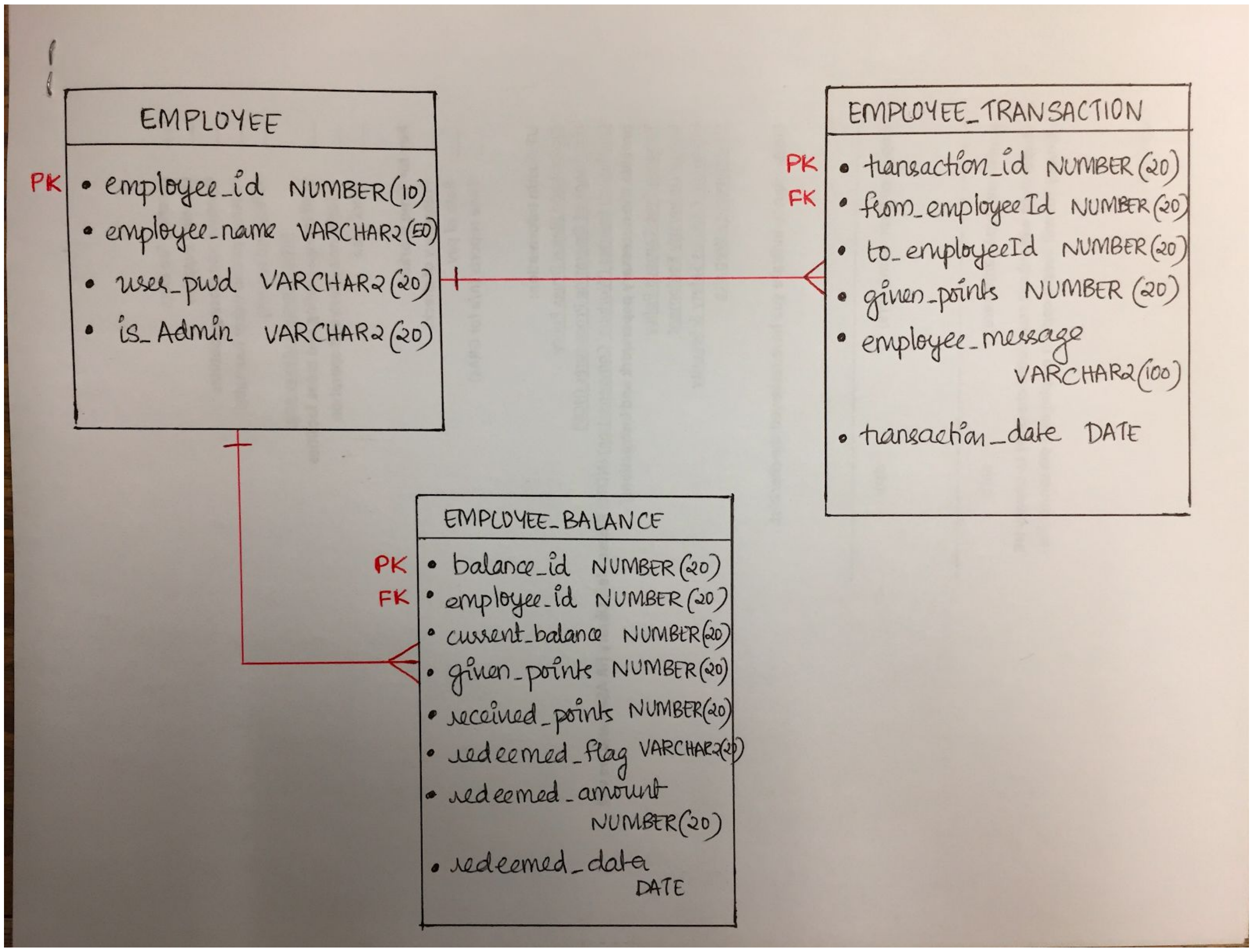
ER Diagram


EMPLOYEE
PK
• employee_id NUMBER(10)
• employee_name VARCHAR2(50)
• user_pwd VARCHAR2(20)
• is_Admin VARCHAR2(20)
EMPLOYEE_TRANSACTION
PK
FK
• transaction_id NUMBER(20)
• from_employeeId NUMBER(20)
• to_employeeId NUMBER(20)
• given_points NUMBER(20)
• employee_message VARCHAR2(100)
• transaction_date DATE
EMPLOYEE_BALANCE
PK
FK
• balance_id NUMBER(20)
• employee_id NUMBER(20)
• current_balance NUMBER(20)
• given_points NUMBER(20)
• received_points NUMBER(20)
• redeemed_flag VARCHAR2(20)
• redeemed_amount NUMBER(20)
• redeemed_date DATE

ADMIN

- APARNA - employee name
- Hello - password

USERS

- NETHRA - hello1
- LEENA - hello2
- AIDITH - hello3
- ANAS - hello4
- HATCHI - hello5

Assumptions made :

1. **Employee can redeem only once in a day.**
2. **No sign up page given.**

--- EMPLOYEE table --

```
# Employee table is created
CREATE TABLE employee(
   employee_id  NUMBER(10) NOT NULL,
   employee_name VARCHAR2(50)  NOT NULL,
   user_pwd VARCHAR2(20));


SELECT * FROM employee ;

# Primary key constraint added
ALTER TABLE employee ADD (
  CONSTRAINT employee_pk PRIMARY KEY (employee_id));

# EmployeeId is generated using sequence
CREATE SEQUENCE employee_sequence
         START WITH 1
         INCREMENT BY 1
         CACHE 100 ;

DROP SEQUENCE employee_sequence;

# Added column using ADD column constraint
ALTER TABLE employee
ADD  is_Admin VARCHAR2(20);
```

-- PASSWORD encryption --

-- Security Package --

```
CREATE OR REPLACE PACKAGE employee_security AS

  FUNCTION get_hash (p_username  IN  VARCHAR2,
                     p_password  IN  VARCHAR2)
   RETURN VARCHAR2;

  PROCEDURE add_user (p_username  IN  VARCHAR2,
                    p_password  IN  VARCHAR2);

  PROCEDURE valid_user (p_username  IN  VARCHAR2,
                        p_password  IN  VARCHAR2);

  FUNCTION valid_user (p_username  IN  VARCHAR2,
                       p_password  IN  VARCHAR2)
    RETURN BOOLEAN;

END;
/
```

-- Body of the package --

```
CREATE OR REPLACE PACKAGE BODY employee_security AS

  FUNCTION get_hash (p_username  IN  VARCHAR2,
                     p_password  IN  VARCHAR2)
    RETURN VARCHAR2 AS
    l_salt VARCHAR2(30) := 'PutYourSaltHere';
  BEGIN
  RETURN DBMS_OBFUSCATION_TOOLKIT.MD5(input_string => UPPER(p_username) ||
l_salt || UPPER(p_password));
   -- RETURN DBMS_CRYPTO.HASH(UTL_RAW.CAST_TO_RAW(UPPER(p_username) ||
l_salt || UPPER(p_password)),DBMS_CRYPTO.HASH_SH1);
  END;

  PROCEDURE add_user (p_username  IN  VARCHAR2,
                      p_password  IN  VARCHAR2) AS
  BEGIN
    INSERT INTO employee (
      employee_id,
      employee_name,
      user_pwd
    )
    VALUES (
      employee_sequence.NEXTVAL,
      UPPER(p_username),
      get_hash(p_username, p_password)
    );

    COMMIT;
  END;
  PROCEDURE valid_user (p_username  IN  VARCHAR2,
                        p_password  IN  VARCHAR2) AS
    v_dummy  VARCHAR2(1);
  BEGIN
    SELECT '1'
    INTO   v_dummy
    FROM   employee
    WHERE  employee_name = UPPER(p_username)
    AND    user_pwd = get_hash(p_username, p_password);
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE_APPLICATION_ERROR(-20000, 'Invalid username/password.');
  END;

  FUNCTION valid_user (p_username  IN  VARCHAR2,
                       p_password  IN  VARCHAR2)
    RETURN BOOLEAN AS
  BEGIN
    valid_user(p_username, p_password);
    RETURN TRUE;
  EXCEPTION
    WHEN OTHERS THEN
      RETURN FALSE;
  END;

END;
/
```

-- Testing - by creating a new user --

```
exec employee_security.add_user('Aparna','hello');
exec employee_security.add_user('Nethra','hello1');
exec employee_security.add_user('Leena','hello2');
exec employee_security.add_user('Aidith','hello3');
exec employee_security.add_user('Anas','hello4');
exec employee_security.add_user('Hatchi','hello5');

UPDATE employee
set is_Admin = 'N'
where employee_id = 5;

SELECT * FROM employee;
```

-- Valid user procedure --

```
EXEC employee_security.valid_user('Aparna','hello');
```

-- Valid user function --

```
SET SERVEROUTPUT ON
BEGIN
   IF employee_security.valid_user('Aparna','hello') THEN
     DBMS_OUTPUT.PUT_LINE('TRUE');
   ELSE
      DBMS_OUTPUT.PUT_LINE('FALSE');
   END IF;
END;
/

BEGIN
   IF employee_security.valid_user('Aparna','hALLA') THEN
     DBMS_OUTPUT.PUT_LINE('TRUE');
   ELSE
      DBMS_OUTPUT.PUT_LINE('FALSE');
   END IF;
END;
/
```

-------------------------------------- EMPLOYEE_TRANSACTION table --------------------------------------

```
SELECT * from employee_transaction;
```

# Employee_transaction table is created

```
CREATE TABLE employee_transaction(
   transaction_id NUMBER(20) NOT NULL,
   from_employeeId NUMBER(20) NOT NULL,
   to_employeeId NUMBER(20),
   given_points NUMBER(20),
   employee_message VARCHAR2(100),
   transaction_date DATE );

DROP TABLE employee_transaction;
```

# Primary key constraint added

```
ALTER TABLE employee_transaction ADD (
   CONSTRAINT transaction_pk PRIMARY KEY (transaction_id));



# Foreign key constraint added
ALTER TABLE employee_transaction ADD (
   CONSTRAINT transaction_fk FOREIGN KEY (from_employeeId) REFERENCES
employee(employee_id));

# Transaction_Id is generated using sequence
CREATE SEQUENCE transaction_sequence
         START WITH 100
         INCREMENT BY 1
         NOCACHE ;

DROP SEQUENCE transaction_sequence ;

# Trigger is being used to automatically add the date of transaction to the table whenever a
transaction happens
CREATE OR REPLACE TRIGGER transction_date_trigger
BEFORE INSERT ON employee_transaction
FOR EACH ROW
BEGIN
  :NEW.transaction_date := SYSDATE;

END;
/


# A stored procedure is created to insert data into the table employee_transaction
CREATE OR REPLACE procedure insert_employee_transaction
( t_from_employeeId in NUMBER,
  t_to_employeeId in NUMBER,
  t_given_points in NUMBER,
  t_employee_message IN VARCHAR2,
  t_transaction_date IN DATE
)
is
 new_balance number;
begin

  SELECT current_balance
  INTO new_balance
  FROM employee_balance
  where employee_id = t_from_employeeId;
  if new_balance >= 1000
  then
  INSERT INTO employee_transaction (
        transaction_id,
        from_employeeId,
        to_employeeId,
        given_points,
        employee_message,
        transaction_date)
        VALUES (
        transaction_sequence.nextval,
        t_from_employeeId,
        t_to_employeeId,
        t_given_points,
        t_employee_message,'');

    end if;

end insert_employee_transaction;
/

exec insert_employee_transaction(2,1,10000,'Well done!','');
```

-- EMPLOYEE_BALANCE table --

```
SELECT * FROM employee_balance;

# Employee_balance table is created
CREATE TABLE employee_balance(
            balance_id NUMBER(20) NOT NULL,
            employee_id NUMBER(20) NOT NULL,
            current_balance NUMBER(20),
            given_points NUMBER(20),
            received_points NUMBER(20),
            redeemed_flag VARCHAR2(20),
            redeemed_amount NUMBER(20),
            redeemed_data DATE);

DROP TABLE employee_balance;
```

```
# Balance_Id is generated using sequence
CREATE SEQUENCE employee_balanace_sequence
        START WITH 1
        INCREMENT BY 1
        NOCACHE;

DROP SEQUENCE employee_balanace_sequence;

 # Primary key constraint added
ALTER TABLE employee_balance ADD (
   CONSTRAINT balance_pk PRIMARY KEY(balance_id));

# Foreign key constraint added
ALTER TABLE employee_balance ADD (
   CONSTRAINT balance_fk FOREIGN KEY (employee_id) REFERENCES
employee(employee_id));
 # Inserting the values to the table .The current_balance column is set to 1000 initially and
reddemed_flag to ‘N’ and all the other data as 0(for testing purpose of redeem functionality
received_points for few emplyee_id is being set to values greater than 10,000).The date is given
empty since we are using a trigger while redeeming the points which will update the date to
sysdate hen the trigger is triggered.

INSERT INTO employee_balance (
balance_id,employee_id,current_balance,given_points,received_points,redeemed_flag,redeem
ed_amount,redeemed_data)
   VALUES(employee_balanace_sequence.NEXTVAL,1,1000,0,10050,'N',0,'');

INSERT INTO employee_balance (
balance_id,employee_id,current_balance,given_points,received_points,redeemed_flag,redeem
ed_amount,redeemed_data)
   VALUES(employee_balanace_sequence.NEXTVAL,2,1000,0,0,'N',0,'');

INSERT INTO employee_balance (
balance_id,employee_id,current_balance,given_points,received_points,redeemed_flag,redeem
ed_amount,redeemed_data)
   VALUES(employee_balanace_sequence.NEXTVAL,3,1000,0,20000,'N',0,'');

INSERT INTO employee_balance (
balance_id,employee_id,current_balance,given_points,received_points,redeemed_flag,redeem
ed_amount,redeemed_data)
   VALUES(employee_balanace_sequence.NEXTVAL,4,1000,0,0,'N',0,'');
```

```
INSERT INTO employee_balance (
balance_id,employee_id,current_balance,given_points,received_points,redeemed_flag,redeem
ed_amount,redeemed_data)
   VALUES(employee_balanace_sequence.NEXTVAL,5,1000,0,30000,'N',0,'');
```

# A stored procedure is created for the transaction update purpose in the employee_balance table as soon as a transaction happens in the employee_transanction table. According to each transaction the given_points,recieved_points and current_balance of the employee is updated.

```
CREATE OR REPLACE  PROCEDURE given_and_received_proc(
                                      from_employee_id_in IN NUMBER,
                                      to_employee_id_in IN NUMBER,
                                      bal_in IN NUMBER)
IS
  from_given_points number(20);
  to_received_points number(20);
  from_balance_amount number(20);
  to_balance_amount number(20);
BEGIN

   select current_balance
   into from_balance_amount
   from employee_balance
   where employee_id = from_employee_id_in and redeemed_flag = 'N';

   select current_balance
   into to_balance_amount
   from employee_balance
   where employee_id = to_employee_id_in and redeemed_flag = 'N';


   select given_points
   into from_given_points
   from employee_balance
   where employee_id = from_employee_id_in and redeemed_flag = 'N';

   select received_points
   into to_received_points
   from employee_balance
   where employee_id = to_employee_id_in and redeemed_flag = 'N';

   if from_balance_amount < bal_in then
```

```
        dbms_output.put_line('The amount entered is more than the amount balance');
    else
        update employee_balance
        set given_points = from_given_points + bal_in
        where employee_id = from_employee_id_in and redeemed_flag = 'N';


        update employee_balance
        set current_balance = from_balance_amount - bal_in
        where employee_id = from_employee_id_in and redeemed_flag = 'N';

    end if;

    if to_balance_amount < bal_in then
        dbms_output.put_line('The amount entered is more than the amount balance');
    else

        update employee_balance
        set received_points = to_received_points + bal_in
        where employee_id = to_employee_id_in and redeemed_flag = 'N';


        update employee_balance
        set current_balance = to_balance_amount + bal_in
        where employee_id = to_employee_id_in and redeemed_flag = 'N';
    end if;


  dbms_output.put_line('Money has been withdrawn successfully');
END;
/
```

# To execute both the procedures. The procedures are executed sequentially.

```
EXEC insert_employee_transaction(1,2,1000,'Well done!','')
EXEC given_and_received_proc(1,2,1000);
```

------------------------------- REEDEM_POINTS procedure ---

# A stored procedure is created which is used to redeem points. The points are redeemed only if the employees have received_points >=10000. When the employee redeems ,the received_points is subtracted with 10000 and the redeemed_flag is set to ‘Y’ and also the redeemed_amount to 100 with redeemed_date set to sysdate. At the same time a new entry is

being created in the employee_balance table with the updated recieved_points and current_balance and the redeemed_flag set to ‘N’ and redeemed_date to null.

```
CREATE OR REPLACE PROCEDURE redeem_points( from_employee_id_in IN NUMBER)
IS
  from_received_points number(20);
  new_from_received_points number(20);
  new_current_balance number(20);
  new_given_points number(20);
  new_redeemed_flag varchar2(20);
  dollar_amount number(20);
  new_new_redeemed_flag varchar2(20);
  new_redeemed_date date;

BEGIN
    select received_points
    into from_received_points
    from employee_balance
    where employee_id = from_employee_id_in and redeemed_flag = 'N';

    select redeemed_amount
    into dollar_amount
    from employee_balance
    where employee_id = from_employee_id_in and redeemed_flag = 'N';


     select current_balance
    into new_current_balance
    from employee_balance
    where employee_id = from_employee_id_in and redeemed_flag = 'N';


    select given_points
    into new_given_points
    from employee_balance
    where employee_id = from_employee_id_in and redeemed_flag = 'N';

    select redeemed_flag
    into new_redeemed_flag
    from employee_balance
    where employee_id = from_employee_id_in and redeemed_flag = 'N';

    if  from_received_points >= 10000 and new_redeemed_flag = 'N' then
```

```
        dbms_output.put_line('Money has been withdrawn successfully');

        update employee_balance
        set redeemed_flag = 'Y'
        where employee_id = from_employee_id_in and redeemed_flag = 'N';

        update employee_balance
        set redeemed_amount = 100
        where employee_id = from_employee_id_in and redeemed_flag = 'Y' and redeemed_data
is null;

        update employee_balance
        set redeemed_data = SYSDATE
        where employee_id = from_employee_id_in and redeemed_flag = 'Y' and redeemed_data
is null;

        update employee_balance
        set received_points = from_received_points - 10000
        where employee_id = from_employee_id_in and redeemed_flag = 'Y' and redeemed_data
= (SELECT MAX(REDEEMED_DATA) FROM EMPLOYEE_BALANCE WHERE  employee_id =
from_employee_id_in AND REDEEMED_FLAG='Y');

        select received_points
        into new_from_received_points
        from employee_balance
        where employee_id = from_employee_id_in and redeemed_flag = 'Y' and redeemed_data
= (SELECT MAX(REDEEMED_DATA) FROM EMPLOYEE_BALANCE WHERE  employee_id =
from_employee_id_in AND REDEEMED_FLAG='Y');

        update employee_balance
        set received_points = 10000
        where employee_id = from_employee_id_in and redeemed_flag = 'Y' and redeemed_data
= (SELECT MAX(REDEEMED_DATA) FROM EMPLOYEE_BALANCE WHERE  employee_id =
from_employee_id_in AND REDEEMED_FLAG='Y');



        insert_employee_balance(from_employee_id_in,
                        new_from_received_points ,
                        new_current_balance,
                        new_given_points,
                        new_new_redeemed_flag ,
                        dollar_amount ,
                            new_redeemed_date );




    else
        dbms_output.put_line('The amount is not sufficient to redeem');
    end if;
END;
/


CREATE OR REPLACE procedure insert_employee_balance (
    t_from_employeeId in NUMBER,
    t_received_points in NUMBER,
    t_current_balance in NUMBER,
    t_given_points in NUMBER,
    t_redeemed_flag in VARCHAR2,
    t_dollar_amount in NUMBER,
    t_redeemed_date in DATE)
is
begin

  INSERT INTO employee_balance (
        balance_id,
        employee_id,
        current_balance,
        given_points,
        received_points,
        redeemed_flag,
        redeemed_amount,
        redeemed_data)
        VALUES (
        employee_balanace_sequence.NEXTVAL,
        t_from_employeeId,
        t_current_balance,
        t_given_points,
        t_received_points,
        'N',
        t_dollar_amount,
        '');

end insert_employee_balance;
```

```
/

# To execute the redeem_points procedure with the employee_id
EXEC redeem_points(3);

-------------------------- TO UPDATE CURRENT BALANCE EACH MONTH ---------------------------

CREATE PROCEDURE UPDATE_EMPLOYEE_BALANCE AS
BEGIN
UPDATE EMPLOYEE_BALANCE
SET current_balance = 1000
WHERE EXTRACT(DAY FROM SYSDATE)=01;
COMMIT;
END UPDATE_EMPLOYEE_BALANCE;
/


------------------------------- SCHEDULE PROCEDURE TO RUN EVERYDAY -------------------

BEGIN
  dbms_scheduler.create_job (
   job_name => 'UPDATE_EMPLOYEE_BALANCE_JOB',
   job_type => 'STORED_PROCEDURE',
   job_action => 'UPDATE_EMPLOYEE_BALANCE',
   start_date => '03-NOV-2018 01:00:00 A.M.',
         end_date => '01-DEC-2018 01:00:010 A.M.',
   enabled => true,
   repeat_interval => 'FREQ=DAILY'
  );
END;
/
EXEC DBMS_SCHEDULER.ENABLE('UPDATE_EMPLOYEE_BALANCE_JOB');

-------------------------------------------------- ADMIN RESET --------------------------------------------------

# A stored procedure is provided for the admin to do hard reset of the balances back to 1000.
CREATE OR REPLACE PROCEDURE admin_reset
IS
BEGIN
UPDATE EMPLOYEE_BALANCE
SET current_balance = 1000;
END;
/
```

```
EXEC admin_reset;
SELECT * FROM employee_balance;
```

REPORTS(using views)

1 . One that shows the aggregate usage of points on a monthly basis – both rewards given out and rewards cashed in, as well as broken down by user, ranked in order of most points received to least

```
CREATE  VIEW Report_1
AS

SELECT
    NVL(A.EMPLOYEE_ID,B.EMPLOYEE_ID) AS EMPLOYEE,
    COALESCE(A.REWARD_GIVEN,0) AS REWARD_GIVEN,
    COALESCE(B.REWARD_RECEIVED,0) AS REWARD_RECEIVED,
    NVL(B.REWARD_RECEIVED_MONTH,A.REWARD_GIVEN_MONTH) AS MONTH
    FROM
    (
    SELECT
    FROM_EMPLOYEEID AS EMPLOYEE_ID,
   SUM(GIVEN_POINTS) AS REWARD_GIVEN,
   EXTRACT(MONTH FROM TRANSACTION_DATE) AS REWARD_GIVEN_MONTH
   FROM EMPLOYEE_TRANSACTION
   GROUP BY FROM_EMPLOYEEID,EXTRACT(MONTH FROM TRANSACTION_DATE)
   )
   A
   FULL OUTER JOIN
   (
   SELECT
   TO_EMPLOYEEID AS EMPLOYEE_ID,
   SUM(GIVEN_POINTS) AS REWARD_RECEIVED,
   EXTRACT(MONTH FROM TRANSACTION_DATE) AS REWARD_RECEIVED_MONTH
   FROM EMPLOYEE_TRANSACTION
   GROUP BY TO_EMPLOYEEID,EXTRACT(MONTH FROM TRANSACTION_DATE)
   )
   B
   ON A.EMPLOYEE_ID=B.EMPLOYEE_ID AND
A.REWARD_GIVEN_MONTH=B.REWARD_RECEIVED_MONTH
   ORDER BY  REWARD_RECEIVED DESC;
```

```
SELECT  * FROM Report_1;
```

2. One that shows who isn't giving out all of their points for the most recent month only (including those that haven't used any)

```
CREATE  VIEW Report_2
AS

SELECT EMPLOYEE_ID,'NO CREDIT POINTS TRANSFERED' AS STATUS_MESSAGE
FROM EMPLOYEE_BALANCE
WHERE  current_balance =1000 AND REDEEMED_FLAG='N'

UNION

SELECT EMPLOYEE_ID,'NO REDEMPTION FOR THE LATEST MONTH' AS
STATUS_MESSAGE
FROM EMPLOYEE_BALANCE
WHERE EMPLOYEE_ID NOT IN (SELECT DISTINCT EMPLOYEE_ID FROM
EMPLOYEE_BALANCE WHERE  REDEEMED_FLAG='Y' AND EXTRACT(MONTH FROM
REDEEMED_DATA)=EXTRACT(MONTH FROM SYSDATE)) and redeemed_flag = 'N';


select * from Report_2;
```

3. One that shows all redemptions, by month by user, for the previous two months

```
CREATE VIEW Report_3
AS

SELECT EMPLOYEE_ID,SUM( RECEIVED_POINTS) AS
TOTAL_REDEMPTION,EXTRACT(MONTH FROM  REDEEMED_DATA) as MONTH
   FROM EMPLOYEE_BALANCE
   WHERE REDEEMED_FLAG = 'Y' AND REDEEMED_DATA BETWEEN
(LAST_DAY(ADD_MONTHS(SYSDATE,-3)))+1 AND
LAST_DAY(ADD_MONTHS(SYSDATE,-1))
   GROUP BY EMPLOYEE_ID,EXTRACT(MONTH FROM  REDEEMED_DATA);
```

```
SELECT * FROM Report_3;
```