/\*

Account Management

1. Allow a regular member to create an account in myLinkedIn by providing email, password, name, city, and country. The unique email will be used as the primary key. You can also assign some distinct identifiers

(e.g., an integer) as primary key. When a member’s account is created, some information related to a member needs to be properly set up, such as creation time and last logintime.

\*/

create or replace procedure LISP\_Create\_Account(

c\_PASSWORD VARCHAR2,

c\_EMAIL VARCHAR2,

c\_FIRST\_NAME VARCHAR2,

c\_LAST\_NAME VARCHAR2,

c\_CITY VARCHAR2,

c\_STATE VARCHAR2,

c\_ZIP\_CODE VARCHAR2,

c\_COUNTRY VARCHAR2,

c\_USER\_TYPE CHAR,

c\_HIRED\_DATE DATE default null, -- not needed for member

c\_DEPARTMENT VARCHAR2 default null, -- not needed for member

c\_SALARY NUMBER default null-- not needed for member

)

as

iemailCount integer;

Begin

if (c\_USER\_TYPE='M' OR c\_USER\_TYPE='A') then

Select Count(\*) into iemailCount from LI\_USER\_MASTER where EMAIL = c\_EMAIL;

if iemailCount > 0 then

dbms\_output.put\_line('New account cannot be created as it Already Exists');

elsif iemailCount = 0 then

insert into LI\_USER\_MASTER

(

USER\_ID ,

PASSWORD ,

EMAIL ,

FIRST\_NAME ,

LAST\_NAME ,

CITY ,

STATE ,

ZIP\_CODE ,

COUNTRY ,

CREATION\_TIME ,

LAST\_LOGINTIME ,

USER\_TYPE

)

values

(

LI\_USER\_MASTER\_SEQ.NextVal,

c\_PASSWORD ,

c\_EMAIL ,

c\_FIRST\_NAME ,

c\_LAST\_NAME ,

c\_CITY ,

c\_STATE ,

c\_ZIP\_CODE ,

c\_COUNTRY ,

SYSTIMESTAMP,

null ,

c\_USER\_TYPE

);

if c\_USER\_TYPE='M' then -- M for Member

insert into LI\_Member (Member\_ID, Account\_Status) values (LI\_USER\_MASTER\_SEQ.CurrVal, 'A');

dbms\_output.put\_line('Member Account Created Successfully');

elsif c\_USER\_TYPE='A' then -- A for Admin

insert into LI\_Administrator (ADMIN\_ID, HIRED\_DATE, DEPARTMENT, SALARY) values (LI\_USER\_MASTER\_SEQ.CurrVal, c\_HIRED\_DATE, c\_DEPARTMENT, c\_SALARY );

dbms\_output.put\_line('Admin Account Created Successfully');

end if;

end if;

else

dbms\_output.put\_line('Invalid Account Type');

end if;

End;

--Execution script:

-- Create Member User  
EXEC LISP\_Create\_Account('msanwal', 'm1@gmail.com','Makesh1', 'Sanwal1', 'Ellicott City', 'MD', '21042', 'US', 'M' );   
   
EXEC LISP\_Create\_Account('phani', 'phanimadha@gmail.com','Phanindra', 'Madha', 'Telgu Town', 'IL', '23451', 'US', 'M' );

EXEC LISP\_Create\_Account('macha', 'kan8@umbc.edu','Ananthachari', 'KV', 'Telgu Town', 'IL', '123456', 'US', 'M' );

EXEC LISP\_Create\_Account('vicky','svivek1@umbc.edu','Vivek', 'sharma', 'B Town', 'NY', '123456', 'US', 'M' );

EXEC LISP\_Create\_Account('Vineet', 'vineeth7@hotmail.com','Choudhary', 'KV', 'Telgu Town', 'TX', '452007', 'US', 'M' );   
  
-- Create Admin User  
exec LISP\_Create\_Account('admin', 'admin1@gmail.com', 'Makesh','Sanwal','Ellicott City','MD','21042','US','A',sysdate,'IT',175000);

exec LISP\_Create\_Account('admin', 'yt21579@umbc.edu', 'Manisha','Kumari','Newyork','NY','76227','US','A',sysdate,'IT',632000);

exec LISP\_Create\_Account('admin', 'dubey1@umbc.edu', 'Nayan','Dubey','Baltimore','MD','21227','US','A',sysdate,'IT',232000);

exec LISP\_Create\_Account('admin', 'bhu1@umbc.edu', 'Bhushan','Bhilegaonkar','Baltimore','MD','21227','US','A',sysdate,'IT',232000);

-- select \* from LI\_USER\_MASTER;

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

/\*

2. Validate the identity of a regular member or an administrator based on the input email and password. For a

regular member, if the account is inactive, an error message should be displayed. If the validation is successful,

the attribute last logintime for the account should be updated to the current system time.

\*/

/\*

Below procedure is common for both member and and produces result based on input.

\*/

create or replace PROCEDURE LISP\_USER\_LOGIN

(p\_email in varchar2, p\_password in varchar2)

as

c\_count number;

c\_account\_type char(1);

c\_last\_logintime timestamp;

c\_account\_status char(1);

begin

select count(\*)

into c\_count

from LI\_USER\_MASTER

where email = p\_email

and password = p\_password;

if c\_count = 0 then

dbms\_output.put\_line( 'Invalid username or password!');

elsif c\_count = 1 then

-- u/p matches but now we need to check if it is a

select user\_type into c\_account\_type from LI\_USER\_MASTER where email=p\_email;

if c\_account\_type='A' then

dbms\_output.put\_line('Admin Login successful!');

elsif c\_account\_type='M' then

select account\_status into c\_account\_status from LI\_MEMBER, LI\_USER\_MASTER where LI\_MEMBER.MEMBER\_ID=LI\_USER\_MASTER.USER\_ID and LI\_USER\_MASTER.EMAIL=p\_email;

if c\_account\_status='A' then -- Successful login

update LI\_USER\_MASTER SET LAST\_LOGINTIME=SYSTIMESTAMP where LI\_USER\_MASTER.EMAIL=p\_email;

dbms\_output.put\_line('Member Login successful!');

ELSE --inactive user

dbms\_output.put\_line('Member is Inactive!');

end if;

end if;

else

dbms\_output.put\_line('Too many matches, wrong logic!');

end if;

end;

--Execution script:

EXEC LISP\_USER\_LOGIN('user@email.com', 'pass');

EXEC LISP\_USER\_LOGIN('admin@email.com', 'pass');

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

/\*

3. Create the usage report for the member who just logs into the system. The report includes the following:

• The number of days since the member first created his/her account in the system

• The number of 1st-degree connections the member currently has

• The number of 1st-degree connections the member has newly created in the last k days, where k is an

input parameter

• The number of recommendations the member has written for his/her 1st-degree connections

• The number of active recommendations (i.e., have not been expired) the member has received Management of Professional Profiles

/\*

CREATE OR REPLACE PROCEDURE LISP\_MemberUsageReport

(

p\_member\_id integer,

p\_k\_days integer

)

AS

v\_days\_created integer;

v\_1stdeg integer;

v\_1stdegkdays integer;

v\_reccreated integer;

v\_active\_recreceived integer;

BEGIN

if LIFN\_ValidateMember(p\_member\_id) = 0 then

dbms\_output.put\_line('Invalid Member');

else

--The number of days since the member first created his/her account in the system

--date1 - (date2 - 1) \* 24 \* 60 \*60 \* 1000

select sysdate - LIFN\_TimeStampToDate(CREATION\_TIME) into v\_days\_created from LI\_User\_Master where user\_id=p\_member\_id; -- user\_id and member\_id are same so no need to add member\_id as user is already validated

dbms\_output.put\_line('The number of days since "' || LIFN\_MemberFullName(p\_member\_id) ||'" first created his/her account in the system: ' || v\_days\_created);

--The number of 1st-degree connections the member currently has

SELECT count(\*) into v\_1stdeg FROM LI\_MEMBER UM, LI\_CONNECTIONS CON1

where CON1.CONNECTED\_MEMBER\_ID = UM.Member\_id AND CON1.MEMBER\_ID=p\_member\_id; -- and accepted\_date >= (TO\_TIMESTAMP(sysdate,'DD-Mon-YYYY HH24-MI-SS') - p\_k\_days);

dbms\_output.put\_line('The number of 1st-degree connections the member currently has : ' || v\_1stdeg);

LISP\_FIRSTDEGCON(p\_member\_id);

-- The number of 1st-degree connections the member has newly created in the last k days, where k is an input parameter

SELECT count(\*) into v\_1stdegkdays FROM LI\_MEMBER UM, LI\_CONNECTIONS CON1

where CON1.CONNECTED\_MEMBER\_ID = UM.Member\_id AND CON1.MEMBER\_ID=p\_member\_id and accepted\_date >= (TO\_TIMESTAMP(sysdate,'DD-Mon-YYYY HH24-MI-SS') - p\_k\_days);

dbms\_output.put\_line('The number of 1st-degree connections the member has newly created in the last ' || p\_k\_days ||' days: ' || LIFN\_MemberFullName(p\_member\_id) ||'" first created his/her account in the system: ' || v\_1stdegkdays);

--The number of recommendations the member has written for his/her 1st-degree connections

select count(\*) into v\_reccreated from LI\_RECOMMENDATIONS where recommender\_mem\_id=p\_member\_id and active\_status='A';

dbms\_output.put\_line('The number of recommendations the "' || LIFN\_MemberFullName(p\_member\_id) ||'" has written for 1st-degree connections : ' || v\_reccreated);

--The number of active recommendations (i.e., have not been expired) the member has received Management of Professional Profiles

select count(\*) into v\_active\_recreceived from LI\_RECOMMENDATIONS where recommendee\_mem\_id=p\_member\_id and active\_status='A';

dbms\_output.put\_line('The number of active recommendations ' || LIFN\_MemberFullName(p\_member\_id) ||' received : ' || v\_active\_recreceived);

end if;

-- exec LISP\_MemberUsageReport (1003, 1);

END;

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

4. Allow a member to select up to 10 skills to indicate his/her expertise. The skills may be those which already

appear in the database, or new ones that are provided by the member.

\*/

-- First Create a member profile as that is required to create skills

create or replace PROCEDURE LISP\_ADD\_Member\_Profile

(

p\_member\_id integer,

p\_prof\_name varchar,

p\_summary varchar2

)

AS

BEGIN

insert into LI\_Professional\_Profiles (

PROF\_ID,

SUMMARY,

MEMBER\_ID,

PROF\_NAME)

values

(

LI\_Professional\_Profiles\_SEQ.NextVal,

p\_summary,

p\_member\_id,

p\_prof\_name

);

dbms\_output.put\_line('Profile created successfully');

END;

-- Execution Script:

EXEC LISP\_ADD\_Member\_Profile (1001, 'Java Developer', 'This is summary');

EXEC LISP\_ADD\_Member\_Profile (1001, 'Painter', 'This is summary');

EXEC LISP\_ADD\_Member\_Profile (1003, 'Business Analyst', 'Handling the Requirement gathering Process');

EXEC LISP\_ADD\_Member\_Profile (1003, 'Event Manager', 'I do part time event management');

EXEC LISP\_ADD\_Member\_Profile (1002, 'System Engineer', 'This Summary for system Engineer');

EXEC LISP\_ADD\_Member\_Profile (1002, 'Model', 'Supermodel and Actor. Did promotional Videos');

EXEC LISP\_ADD\_Member\_Profile (1002, 'Anchor', 'Part time Anchoring');

EXEC LISP\_ADD\_Member\_Profile (1004, 'Graduate Assistant', 'Assist Department of Computer Science');

EXEC LISP\_ADD\_Member\_Profile (1004, 'Senior Systems Engineer', 'Enhancement of RnD');

EXEC LISP\_ADD\_Member\_Profile (1004, 'Footballer', 'Play County League Football');

EXEC LISP\_ADD\_Member\_Profile (1005, 'Data scientist', 'Do pattern detection in data');

EXEC LISP\_ADD\_Member\_Profile (1006, 'Researcher', 'finding fast algorithm techniques ');

EXEC LISP\_ADD\_Member\_Profile (1007, 'Doctor', 'Heart Specialist ');

-- Add Skill Procedure

create or replace PROCEDURE LISP\_ADD\_MEMBER\_SKILL

(

p\_skill\_name VARCHAR2,

p\_member\_id INTEGER,

p\_prof\_id INTEGER

) AS

v\_mscount INTEGER; -- store member skill count for skill

v\_scount INTEGER; -- store skill count in the master

v\_skill\_id INTEGER; -- temporary store skill id for later insertion

v\_tscount INTEGER; -- total skill count for 10 skills restriction.

BEGIN

-- check if skill exists for the member

-- Check if alredy 10 skills exists for member

select count(\*) into v\_tscount from LI\_MEMBER\_SKILL ms, LI\_PROFESSIONAL\_PROFILES pp where

ms.PROF\_ID=pp.PROF\_ID and pp.MEMBER\_ID=p\_member\_id and pp.PROF\_ID=p\_prof\_id;

if v\_tscount>=10 then -- we can also use sql%RowCount

dbms\_output.put\_line('Member already has 10 skills in the profile so cannot add more, please remove anotther first to add one!');

else -- we are good to add a skill subject to if it is not already added

select count(\*) into v\_mscount from LI\_Skill\_Master sm, LI\_MEMBER\_SKILL ms, LI\_PROFESSIONAL\_PROFILES pp where

sm.SKILL\_ID=ms.SKILL\_ID and ms.PROF\_ID=pp.PROF\_ID and sm.SKILL\_NAME = p\_skill\_name and pp.PROF\_ID=p\_prof\_id;

if v\_mscount = 0 then --skill does not exist for member so need to add it

BEGIN -- ADD SKILL

-- now we need to check if the skill exists in the Skill Master and if not then create it.

select count(\*) into v\_scount from LI\_Skill\_Master where SKILL\_NAME=p\_skill\_name;

if v\_scount=0 then -- Skill does not exit so first add it

INSERT INTO LI\_Skill\_Master (Skill\_ID, Skill\_Name) values (LI\_Skill\_Master\_SEQ.NextVal, p\_skill\_name);

v\_skill\_id := LI\_SKILL\_MASTER\_SEQ.CurrVal;

elsif v\_scount=1 then

select SKILL\_ID into v\_skill\_id from LI\_Skill\_Master where SKILL\_NAME=p\_skill\_name;

end if;

INSERT INTO LI\_MEMBER\_SKILL (MEMBER\_SKILL\_ID, PROF\_ID,SKILL\_ID) values (LI\_MEMBER\_SKILL\_SEQ.NextVal, p\_prof\_id, v\_skill\_id);

dbms\_output.put\_line('Skill "' || p\_skill\_name || '" added for the Member!');

END; -- END ADD SKILL

elsif v\_mscount >= 1 then

dbms\_output.put\_line('Skill "' || p\_skill\_name || '" already exists for the Member!');

end if;

end if;

END;

-- Use below script to exec the add script

-- EXEC LISP\_ADD\_MEMBER\_SKILL ('C Sharp', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('object Oriented Programming', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('MySQL', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('Web Development', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('Web Services', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('Oracle', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('C', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('C++', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('ASP.NET', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('ADO.NET', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('ASP.NET MVC', 1001, 1000);

EXEC LISP\_ADD\_MEMBER\_SKILL ('drawing', 1001, 1001);

EXEC LISP\_ADD\_MEMBER\_SKILL ('color combination', 1001, 1001);

EXEC LISP\_ADD\_MEMBER\_SKILL ('painting', 1001, 1001);

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

/\*

5. Allow a member to create a new entry of Work Experience into his/her profile.

\*/

create or replace PROCEDURE LISP\_ADD\_WORKEXPERIENCE

(

p\_member\_id INTEGER,

p\_prof\_id INTEGER,

p\_org VARCHAR2,

p\_job\_position VARCHAR2,

p\_start\_date DATE,

p\_end\_date DATE,

P\_description VARCHAR2

)

AS

v\_pwcount INTEGER; -- store member skill count for skill

v\_wcount INTEGER; -- store skill count in the master

v\_work\_id INTEGER; -- temporary store skill id for later insertion

v\_tscount INTEGER; -- total skill count for 10 skills restriction.

BEGIN

-- check if work exists for the member

select count(\*) into v\_pwcount from LI\_WORK\_EXPERIENCE we, LI\_PROFILE\_WORK\_EXPERIENCE pwe where

we.WORK\_ID=pwe.WORK\_ID and pwe.PRO\_ID=p\_prof\_id and we.MEMBER\_ID = p\_member\_id and we.ORGANIZATION=p\_org and we.START\_DATE=p\_start\_date;

if v\_pwcount = 0 then --work does not exist for member so need to add it

BEGIN -- ADD work

-- now we need to check if the work exists in the work experince and if not then create it.

select count(\*) into v\_wcount from LI\_WORK\_EXPERIENCE we where we.MEMBER\_ID = p\_member\_id and we.ORGANIZATION=p\_org and we.START\_DATE=p\_start\_date ;

if v\_wcount=0 then -- Work does not exit so first add it

INSERT INTO LI\_WORK\_EXPERIENCE (Work\_ID, MEMBER\_ID, ORGANIZATION, JOB\_POSITION, START\_DATE,END\_DATE, DESCRIPTION ) values

(LI\_WORK\_EXPERIENCE\_SEQ.NextVal, p\_member\_id, p\_org, p\_job\_position, p\_start\_date, p\_end\_date, p\_description);

v\_work\_id := LI\_WORK\_EXPERIENCE\_SEQ.CurrVal;

elsif v\_wcount=1 then

select Work\_ID into v\_work\_id from LI\_WORK\_EXPERIENCE we where we.MEMBER\_ID = p\_member\_id and we.ORGANIZATION=p\_org and we.START\_DATE= p\_start\_date;

end if;

INSERT INTO LI\_PROFILE\_WORK\_EXPERIENCE (PRO\_WORK\_EXPERIENCE\_ID, PRO\_ID, WORK\_ID) values (LI\_PROFILE\_WORK\_EXPERIENCE\_SEQ.NextVal, p\_prof\_id, v\_work\_id);

dbms\_output.put\_line('Work Experience "' || P\_description || '" added for the Member profile!');

END; -- END ADD work

elsif v\_pwcount > 1 then

dbms\_output.put\_line('Work experience for "' || p\_org || '" already exists for the Member profile!');

end if;

END;

-- Execution script

insert into li\_work\_experience values (li\_work\_experience\_seq.nextval,1007,'IBM','Director',date'2010-01-01',date'2011-02-02','C Developer');

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

/\*

6. Allow a member to remove one group from his/her profile.

\*/

-- Remove group procedure will be followed by add group procedure

create or replace Procedure LISP\_RemoveGroupFromProfile

(p\_group\_id Integer, p\_memeber\_id Integer)

as

v\_count integer;

begin

select count (\*) into v\_count from LI\_Group\_Members where group\_id = p\_group\_id and member\_id = p\_memeber\_id;

if v\_count=0 then

dbms\_output.put\_line('Group does not belong to member so no action taken, please change the values and try again');

else

delete from LI\_Group\_Members where group\_id = p\_group\_id and member\_id = p\_memeber\_id;

dbms\_output.put\_line('Group removed successfully from member profile');

end if;

-- Test Script

-- Exec LISP\_RemoveGroupFromProfile(1001, 1003)

end;

-- Add Group Script

create or replace Procedure LISP\_AddGroup

(p\_group\_name varchar, p\_memeber\_id Integer)

as

v\_count integer;

begin

select count(\*) into v\_count from LI\_Groups where GROUP\_NAME=p\_group\_name;

if v\_count > 0 then

dbms\_output.put\_line('The Group "' || p\_group\_name || '" already exists, please change name and try again!');

elsif v\_count=0 then

SAVEPOINT sp\_spcommit;

BEGIN

insert into LI\_Groups (GROUP\_ID, GROUP\_NAME, CREATEDBY\_MEMBER\_ID, CREATED\_DATE) values (LI\_Groups\_SEQ.NextVal, p\_group\_name, p\_memeber\_id, systimestamp);

insert into LI\_GROUP\_MEMBERS (GROUP\_ID, MEMBER\_ID, JOIN\_DATE) VALUES (LI\_Groups\_SEQ.CURRVAL, p\_memeber\_id, SYSDATE);

dbms\_output.put\_line('The Group "' || p\_group\_name || '" created successfully! The group also added to the Member profile');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK TO sp\_spcommit;

-- And of course we raise again,

-- since we don't want to hide the error.

-- Not raising here is an errorm, but we are displaying a message.

dbms\_output.put\_line('There has been error in saving the group please try again');

--RAISE;

END;

else

dbms\_output.put\_line('The Group "' || p\_group\_name || '" already exists!');

end if;

-- Test Script

-- EXEC LISP\_AddGroup ('LinkedIN Group', 1003);

end;

-- Execution Script:

EXEC LISP\_AddGroup ('Java Developer Group', 1003);

EXEC LISP\_AddGroup ('Oracle Developer Group', 1003);

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

/\*

7. Allow a member to retrieve the list of his/her top-3 skills and the corresponding skill endorsements that he/she has received from others. The skills need to be sorted in descending order based on the number of

endorsements. Only the top-3 skills are needed to be retrieved.

\*/

CREATE OR REPLACE PROCEDURE LISP\_MemberTop3Skills

(

p\_member\_id integer

)

AS

cursor c1 is SELECT \* FROM (select sm.SKILL\_NAME Skill\_Name, count(\*) ecount from LI\_Professional\_Profiles pp, LI\_Member\_skill ms, LI\_ENDORSEMENTS en, LI\_SKILL\_MASTER sm

where pp.PROF\_ID = ms.PROF\_ID and ms.SKILL\_ID = sm.SKILL\_ID and ms.MEMBER\_SKILL\_ID = en.MEMBER\_SKILL\_ID and pp.MEMBER\_ID=p\_member\_id

group by sm.SKILL\_NAME

order by count(\*) desc

)

WHERE ROWNUM <= 3;

v\_count integer;

BEGIN

v\_count := 0;

if LIFN\_ValidateMember(p\_member\_id) = 0 then

dbms\_output.put\_line('Invalid Member');

else

dbms\_output.put\_line(LIFN\_MemberFullName(p\_member\_id) || '''s top-3 skills and the corresponding skill endorsements numbers: ');

for c in c1 loop

exit when c1%notfound;

v\_count := 1;

dbms\_output.put\_line('SKILL Name: ' || c.Skill\_Name || ', No. of Endorsements: ' || c.ecount);

end loop;

if v\_count = 0 then

dbms\_output.put\_line( 'No skill endorsements found for the member!');

end if;

end if;

END;

-- EXEC LISP\_MemberTop3Skills (1003);

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

/\*8. Allow a member to retrieve his/her complete professional profile, including Summary, Work Experience, Education, Skills, Groups, and Contact Information. Please organize the entries in Work Experience and Education using reverse chronological order.

\*/

--get work experience details

create or replace procedure LISP\_GetMemberProf\_Profile (p\_prof\_id integer) as

cursor c1 is select lwe.organization,lwe.job\_position,lwe.start\_date,lwe.end\_date,lwe.description

from li\_work\_experience lwe,li\_professional\_profiles lpp,li\_member lm

where lpp.member\_id=lm.member\_id and lm.member\_id=lwe.member\_id and p\_prof\_id=lpp.prof\_id

order by start\_date desc;

--get educational details

cursor c2 is select led.university,led.major,led.degree,led.start\_date,led.end\_date,led.grade

from li\_education\_details led,li\_professional\_profiles lpp,li\_member lm

where lpp.member\_id=lm.member\_id and lm.member\_id=led.member\_id and p\_prof\_id=lpp.prof\_id

order by start\_date desc;

--get skills of member

cursor c3 is select skill\_name from li\_skill\_master,li\_member\_skill,li\_professional\_profiles

where li\_skill\_master.skill\_id=li\_member\_skill.skill\_id and li\_member\_skill.prof\_id=li\_professional\_profiles.prof\_id

and p\_prof\_id=li\_professional\_profiles.prof\_id;

--get groups information of member

cursor c4 is select lg.group\_name from li\_professional\_profiles lpp,LI\_MEMBER lm,li\_group\_members lgm,li\_groups lg

where p\_prof\_id=lpp.prof\_id and lpp.member\_id=lm.member\_id and lm.member\_id=lgm.member\_id and lgm.group\_id=lg.group\_id;

--get user information

cursor c5 is select lw.web\_url,lw.web\_type,lce.email,lmcd.cell\_phone,lmcd.home\_phone,lmcd.facebook\_url,lmcd.twitter\_handle

from li\_professional\_profiles lpp,li\_member lm,li\_member\_contact\_details lmcd,li\_websites lw,li\_contact\_email lce

where lpp.prof\_id=p\_prof\_id and lpp.member\_id=lm.member\_id and lm.member\_id=lmcd.member\_id and lmcd.contact\_id=lw.contact\_id and lmcd.contact\_id=lce.contact\_id;

v\_profid LI\_PROFESSIONAL\_PROFILES.PROF\_ID%type;

sumv li\_professional\_profiles.summary%type;

v\_o LI\_WORK\_EXPERIENCE.ORGANIZATION%type;

v\_jp LI\_WORK\_EXPERIENCE.JOB\_POSITION%type;

v\_sd LI\_WORK\_EXPERIENCE.start\_date%type;

v\_ed LI\_WORK\_EXPERIENCE.end\_date%type;

v\_d LI\_WORK\_EXPERIENCE.description%type;

v\_uni LI\_EDUCATION\_DETAILS.UNIVERSITY%type;

v\_maj LI\_EDUCATION\_DETAILS.MAJOR%type;

v\_deg LI\_EDUCATION\_DETAILS.DEGREE%type;

v\_sde LI\_EDUCATION\_DETAILS.START\_DATE%type;

v\_ede LI\_EDUCATION\_DETAILS.END\_DATE%type;

v\_g LI\_EDUCATION\_DETAILS.GRADE%type;

v\_sns LI\_SKILL\_MASTER.SKILL\_NAME%type;

v\_gng LI\_GROUPS.GROUP\_NAME%type;

v\_wuc li\_websites.web\_url%type;

v\_wtc li\_websites.web\_type%type;

v\_ec li\_contact\_email.email%type;

v\_cpc li\_member\_contact\_details.cell\_phone%type;

v\_hpc li\_member\_contact\_details.home\_phone%type;

v\_fuc li\_member\_contact\_details.facebook\_url%type;

v\_thc li\_member\_contact\_details.twitter\_handle%type;

begin

select prof\_id into v\_profid from LI\_PROFESSIONAL\_PROFILES where p\_prof\_id=prof\_id;

if (p\_prof\_id=v\_profid) then

dbms\_output.put\_line('User details are given below:');

select summary into sumv from li\_professional\_profiles lpp where

lpp.prof\_id=p\_prof\_id;

dbms\_output.put\_line('Summary: '||sumv);

open c1;

dbms\_output.put\_line('Work Experience:');

loop

fetch c1 into v\_o,v\_jp,v\_sd,v\_ed,v\_d;

exit when c1%notfound;

dbms\_output.put\_line('Organization:'||v\_o||', Position:'||v\_jp||', Start Date:'||v\_sd||', End Date'||v\_ed||', Description:'||v\_d);

end loop;

close c1;

open c2;

dbms\_output.put\_line('Education:');

loop

fetch c2 into v\_uni,v\_maj,v\_deg,v\_sde,v\_ede,v\_g;

exit when c2%notfound;

dbms\_output.put\_line('University:'||v\_uni||', Major:'||v\_maj||', Degree:'||v\_deg||', Start Date:'||v\_sde||', End Date:'||v\_ede||', Grade:'||v\_g);

end loop;

close c2;

open c3;

--dbms\_output.put\_line('Skill:');

loop

fetch c3 into v\_sns;

exit when c3%notfound;

dbms\_output.put\_line('Skill:'||v\_sns);

end loop;

close c3;

open c4;

loop

fetch c4 into v\_gng;

exit when c4%notfound;

dbms\_output.put\_line('Groups:'||v\_gng);

end loop;

close c4;

open c5;

dbms\_output.put\_line('Contact Details:');

loop

fetch c5 into v\_wuc,v\_wtc,v\_ec,v\_cpc,v\_hpc,v\_fuc,v\_thc;

exit when c5%notfound;

dbms\_output.put\_line('website:'||v\_wuc||', web\_type:'||v\_wtc||', Email:'||v\_ec||', cell\_phone:'||v\_cpc||', home\_phone:'||v\_hpc||', facebook\_url:'||v\_fuc||', twitter\_handle:'||v\_thc);

end loop;

close c5;

end if;

exception

when no\_data\_found then dbms\_output.put\_line('Invalid ID!');

end;

-- Execution Script:

-- EXEC LISP\_GetMemberProf\_Profile (1002);

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

Management of Social Relationships

/\*

9. Allow a member to create a new connection to a specific individual in myLinkedIn.

\*/

create or replace procedure LISP\_Add\_Connection

(

p\_member\_id in integer,

p\_connected\_member\_id in integer)

is

v\_count integer;

v\_count1 integer;

begin

select count(\*) into v\_count1 from LI\_member where member\_id = p\_member\_id;

if v\_count1=0 then

dbms\_output.put\_line('Invalid Member');

else

select count(\*) into v\_count from LI\_connections where member\_id = p\_member\_id and connected\_member\_id = p\_connected\_member\_id;

-- Connection exists validation

if(v\_count=0) then

insert into LI\_connections values(LI\_connections\_SEQ.nextval, p\_member\_id, P\_connected\_member\_id, SYSDATE, 'A', SYSDATE);

insert into LI\_connections values(LI\_connections\_SEQ.nextval, P\_connected\_member\_id, p\_member\_id, SYSDATE, 'A', SYSDATE);

dbms\_output.put\_line('Member connection is established');

else

dbms\_output.put\_line('No action is taken as connection is already established');

end if;

end if;

end;

-- Execution Script:

EXEC LISP\_Add\_Connection (1001, 1002);

EXEC LISP\_Add\_Connection (1001, 1003);

EXEC LISP\_Add\_Connection (1001, 1004);

EXEC LISP\_Add\_Connection (1001, 1005);

EXEC LISP\_Add\_Connection (1001, 1006);

EXEC LISP\_Add\_Connection (1001, 1007);

EXEC LISP\_Add\_Connection (1002, 1006);

EXEC LISP\_Add\_Connection (1002, 1007);

/\*

10. Allow a member to write a recommendation for one of his/her 1st-degree connections. If the individual is not

a 1st-degree connection, an error message should be displayed.

\*/

create or replace procedure LISP\_Add\_Recommendation(

p\_member\_id in integer,

p\_connected\_member\_id in integer,

p\_comment varchar2

)

is

v\_count integer;

begin

select count(\*) into v\_count from LI\_connections where member\_id =

p\_member\_id and connected\_member\_id = p\_connected\_member\_id;

if(v\_count=0) then

dbms\_output.put\_line('No action is taken as connection is not established!');

else

insert into LI\_recommendations

values(LI\_RECOMMENDATIONS\_SEQ.nextval, p\_member\_id, p\_connected\_member\_id, p\_comment, sysdate, 'A' );

dbms\_output.put\_line('Member recommendation has been added successfully!');

end if;

end ;

-- Execution Script

EXEC LISP\_Add\_Recommendation(20,5, 'Manisha is good in Oracle databases, I strongly recommend her for the job');

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

/\*

11. Allow a member to endorse a skill for one of his/her 1st-degree connections. If the individual is not a 1st-degree

connection, an error message should be displayed.

\*/

create or replace procedure LISP\_Add\_Endorsement

(

p\_member\_skill\_id integer,

p\_ENDORSER\_MEMBER\_ID integer,

p\_ENDORSED\_COMMENT varchar2

)

is

v\_count integer;

v\_member\_skill\_id integer;

begin

select count(\*) into v\_count from LI\_Connections con, LI\_MEMBER\_SKILL ms, LI\_PROFESSIONAL\_PROFILES pp

where

ms.prof\_id = pp.prof\_id and con.connected\_member\_id = pp.member\_id and

ms.member\_skill\_id= p\_member\_skill\_id

and con.member\_id = p\_ENDORSER\_MEMBER\_ID and

connected\_member\_id = pp.member\_id;

select member\_skill\_id into v\_member\_skill\_id from LI\_member\_skill where

member\_skill\_id=p\_member\_skill\_id;

if(v\_count=0) then

dbms\_output.put\_line('No action is taken as there is no connection exists between members');

else

insert into LI\_endorsements (ENDORSEMENT\_ID, MEMBER\_SKILL\_ID, ENDORSER\_MEMBER\_ID, ENDORSED\_COMMENT, ENDORSED\_DATETIME)

values

(LI\_endorsements\_SEQ.nextval, p\_MEMBER\_SKILL\_ID, p\_ENDORSER\_MEMBER\_ID, p\_ENDORSED\_COMMENT, sysdate);

dbms\_output.put\_line('Member Skill is Endorsed');

end if;

exception

when no\_data\_found then

dbms\_output.put\_line('Invalid Skill');

end ;

-- Execution Script:

exec LISP\_Add\_Endorsement(1009, 1003, 'This test endorsement');

select \* from LI\_Member\_Skill;

select \* from Li\_Endorsements

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

/\*

12. Allow a member to retrieve the complete list of his/her 2nd-degree connections.

\*/

-- Procedure 1: First Degree Connection

create or replace PROCEDURE LISP\_FIRSTDEGCON

(

p\_member\_id IN Integer

) AS

cursor c1 is SELECT CONNECTED\_MEMBER\_ID, EMAIL, FIRST\_NAME ||' ' || UM.LAST\_NAME as M\_Name FROM LI\_USER\_MASTER UM, LI\_CONNECTIONS CON1 where CON1.CONNECTED\_MEMBER\_ID = UM.USER\_ID AND CON1.MEMBER\_ID=p\_member\_id;

v\_count integer;

BEGIN

v\_count := 0;

dbms\_output.put\_line('Connected Members to the Member ID - ' || p\_member\_id || ' are:' );

for deg1con in c1 loop

exit when c1%notfound;

v\_count := 1;

dbms\_output.put\_line('Member ID: ' || deg1con.CONNECTED\_MEMBER\_ID || ', Member Name: ' || deg1con.M\_Name || ', Member Email: ' || deg1con.Email );

end loop;

if v\_count = 0 then

dbms\_output.put\_line( 'No First Degree Connections were found for the member!');

end if;

END;

-- Execution Script:

-- Use below script to test the procedure

EXEC LISP\_FIRSTDEGCON (1003);

EXEC LISP\_FIRSTDEGCON (1007);

--Procedure 2:

create or replace PROCEDURE LISP\_SECONDDEGCON

(

p\_member\_id IN Integer

) AS

cursor c1 is SELECT CONNECTED\_MEMBER\_ID, EMAIL, FIRST\_NAME ||' ' || UM.LAST\_NAME as M\_Name

FROM LI\_USER\_MASTER UM, LI\_CONNECTIONS Deg2 where Deg2.CONNECTED\_MEMBER\_ID = UM.USER\_ID and Deg2.CONNECTED\_MEMBER\_ID!=p\_member\_id and Deg2.MEMBER\_ID in

(

SELECT CONNECTED\_MEMBER\_ID FROM LI\_CONNECTIONS Deg1 where Deg1.MEMBER\_ID=p\_member\_id

);

v\_count integer;

BEGIN

v\_count := 0;

dbms\_output.put\_line('Connected Members to the Member ID - ' || p\_member\_id || ' are:' );

for deg2con in c1 loop

exit when c1%notfound;

v\_count := 1;

dbms\_output.put\_line('Member ID: ' || deg2con.CONNECTED\_MEMBER\_ID || ', Member Name: ' || deg2con.M\_Name || ', Member Email: ' || deg2con.Email );

end loop;

if v\_count = 0 then

dbms\_output.put\_line( 'No Second Degree Connections were found for the member!');

end if;

END;

-- Execution Script - Use below script to test the procedure

EXEC LISP\_SECONDDEGCON (1002);

EXEC LISP\_SECONDDEGCON (1003);

EXEC LISP\_SECONDDEGCON (1004);

EXEC LISP\_SECONDDEGCON (1005);

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

/\*

13. Allow a member to use multiple keywords to retrieve the list of his/her 1st-degree connections whose Summary section of profiles match the provided keywords. You can decide to implement either an “AND” semantic (i.e., all the keywords must appear) or an “OR” semantic (i.e., at least one keyword must appear).

\*/

create or replace PROCEDURE LISP\_SearchConnections

(p\_member\_id integer, p\_search\_keyword varchar2)

as

cursor c1 is

SELECT substr(main\_string, position\_from + 1, decode(position\_to, 0, length(main\_string),position\_to - position\_from - 1)) as kw

FROM (SELECT main\_string,

decode(rownum - 1, 0, 0, instr(main\_string, ' ', 1, rownum - 1)) position\_from,

instr(main\_string, ' ', 1, rownum) position\_to

FROM (SELECT p\_search\_keyword main\_string

--FROM (SELECT '111 123 aaabbb 555 AAA' main\_string

FROM dual)

CONNECT BY LEVEL <= length(main\_string))

WHERE position\_to > 0 or position\_from > 0;

d\_sql varchar2(1000);

TYPE CurTyp IS REF CURSOR;

c2 CurTyp;

v\_record integer;

v\_member\_count integer :=0;

begin

d\_sql := 'select CONNECTED\_MEMBER\_ID from LI\_CONNECTIONS con, LI\_PROFESSIONAL\_PROFILES pp where (con.Connected\_Member\_ID = pp.MEMBER\_ID and con.Member\_ID=' || p\_member\_id || ' and pp.Member\_ID != ' || p\_member\_id || ')';

d\_sql := d\_sql || ' and (Summary like ''%' || p\_search\_keyword || '%'' ';

for c in c1 loop

exit when c1%notfound;

-- v\_count := 1;

--if d\_sql is not null then

d\_sql := d\_sql || ' or ';

--end if;

d\_sql := d\_sql || ' Summary like ''%' || c.kw ||'%''';

--dbms\_output.put\_line(c.kw);

--dbms\_output.put\_line('Member ID: ' || deg1con.CONNECTED\_MEMBER\_ID || ', Member Name: ' || deg1con.M\_Name || ', Member Email: ' || deg1con.Email );

end loop;

d\_sql := d\_sql || ') ';

--dbms\_output.put\_line(d\_sql); -- used for debugging

--dbms\_output.put\_line('Below list of members matched with the keyword ' || p\_search\_keyword || ':'); -- used for debugging

OPEN c2 FOR d\_sql;

Loop

fetch c2 into v\_record;

exit when c2%notfound;

if v\_member\_count = 0 then

-- Print the heading

dbms\_output.put\_line('Below list of members matched with the keyword ''' || p\_search\_keyword || ''':');

end if;

v\_member\_count := 1;

dbms\_output.put\_line(v\_record);

end loop;

close c2;

if v\_member\_count = 0 then

dbms\_output.put\_line( 'No connected members found with the Keyword ''' || p\_search\_keyword || '');

end if;

end;

-- Execution Script:

EXEC LISP\_SearchConnections (1003, 'Java Oracle Baltimore';

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

/\*

System Management

14. Allow an administrator to manually set a member’s account to be inactive.

\*/

-- Proc 1 as common procedure used for both activation and deactivation

create or replace Procedure LISP\_ActivateDeactivateAccount -- Feature 14 Main procedure

(

p\_member\_id integer,

p\_active\_status char

)

as

v\_count integer;

begin

select count(\*) into v\_count from LI\_MEMBER where member\_id = p\_member\_id;

if v\_count=0 then

dbms\_output.put\_line('Member Does not exist in the LinkedIN System');

else

select count(\*) into v\_count from LI\_MEMBER where member\_id = p\_member\_id and ACCOUNT\_STATUS = p\_active\_status; -- find out if account is already in the status

if v\_count > 0 then -- account is already in the status

if p\_active\_status='A' then

dbms\_output.put\_line('Member is already active! No Action has been performed');

else

dbms\_output.put\_line('Member is already deactivated! No Action has been performed');

end if;

else

update LI\_Member set Account\_Status = p\_active\_status where MEMBER\_ID = p\_member\_id;

if p\_active\_status='A' then

dbms\_output.put\_line('Member has been activated!');

else

dbms\_output.put\_line('Member has been deactivated!');

end if;

end if;

end if;

end;

-- Proc 2 to make the account inactive

create or replace Procedure LISP\_DeactivateMember -- Feature 14 - Deactivate Member

(

p\_member\_id integer

)

as

begin

LISP\_ActivateDeactivateAccount (p\_member\_id, 'D');

end;

-- Execution script:

EXEC LISP\_DeactivateMember(1001);

EXEC LISP\_DeactivateMember(1002);

-- Proc 3: this will activate the account if admin want to

create or replace Procedure LISP\_ActivateMember -- Feature 14 Activate member

(

p\_member\_id integer

)

as

begin

LISP\_ActivateDeactivateAccount (p\_member\_id, 'A');

end;

/\*

Test the Feature as below

a) LISP\_DeactivateMember will deactivate the member if exists or if not already deactivated

exec LISP\_DeactivateMember(1003);

a) LISP\_ActivateMember will activate the member if exists or if not already activated

exec LISP\_ActivateMember(1003);

\*/

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

/\*

15. Allow an administrator to generate a summary report of the myLinkedIn system, including the following:

• The number of active members

• The number of inactive members

• The top-3 popular skills that are possessed by the myLinkedIn members

• The average number of 1st-degree connections the members have

• The average number of 2nd-degree connections the members have

\*/

create or replace PROCEDURE LISP\_SYSTEMSUMMARYREPORT AS

v\_active\_members integer;

v\_inactive\_members integer;

v\_top3\_skills integer;

v\_ave1st\_deg integer;

v\_ave2nd\_deg integer;

cursor c\_v\_top3\_skills is select SKILL\_NAME, sum(mk.SKILL\_ID) from LI\_MEMBER\_SKILL mk, LI\_SKILL\_MASTER sm where mk.skill\_id=sm.skill\_id

group by SKILL\_NAME order by

sum(mk.SKILL\_ID) desc;

BEGIN

/\*Allow an administrator to generate a summary report of the myLinkedIn system, including the following:

• The number of active members

• The number of inactive members

• The top-3 popular skills that are possed by the myLinkedIn members - todo

• The average number of 1st-degree connections the members have

• The average number of 2nd-degree connections the members have - todo

\*/

Select Count(\*) into v\_active\_members from LI\_USER\_MASTER um, LI\_Member m where um.User\_ID=m.MEMBER\_ID and m.ACCOUNT\_STATUS='A';

Select Count(\*) into v\_inactive\_members from LI\_USER\_MASTER um, LI\_Member m where um.User\_ID=m.MEMBER\_ID and m.ACCOUNT\_STATUS!='A';

select avg(con\_counts) into v\_ave1st\_deg from

(select member\_id, count(\*) con\_counts from LI\_Connections group by member\_id) a;

select avg(con\_counts) into v\_ave2nd\_deg from

(select member\_id, count(\*) con\_counts from LI\_Connections where LI\_Connections.Connected\_Member\_ID in

(select member\_id from LI\_Connections) group by member\_id) a;

dbms\_output.put\_line('Active Members : ' || v\_active\_members);

dbms\_output.put\_line('Inactive Members : ' || v\_inactive\_members);

dbms\_output.put\_line('The average number of 1st-degree connections the members have : ' || v\_ave1st\_deg);

dbms\_output.put\_line('The average number of 2st-degree connections the members have : ' || v\_ave2nd\_deg);

dbms\_output.put\_line('The top-3 popular skills that are possessed by the myLinkedIn members: ');

for c in c\_v\_top3\_skills loop

exit when c\_v\_top3\_skills%notfound or c\_v\_top3\_skills%RowCount>3;

dbms\_output.put\_line(c.Skill\_Name);

end loop;

END;

-- Execution Script:

EXEC LISP\_SYSTEMSUMMARYREPORT;

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

/\*

16. Allow an administrator to update the current time (the variable used to simulate the time of the system) so that we can test those timing-related features without waiting for days (or even years) in real life. This feature needs to set some members’ accounts to be inactive (if the last-login time is more than 365 days ago). You can call Feature 14 to set an account to be inactive. This feature also needs to set some recommendations to be expired if they were posted more than 730 days ago.

\*/

CREATE OR REPLACE PROCEDURE LISP\_UpdateCurrentSysTime (p\_given\_date\_time timestamp)

is

cursor c\_member IS SELECT LAST\_LOGINTIME, Member\_ID FROM LI\_user\_master u, LI\_member m where u.user\_id=m.member\_id and u.Last\_Logintime is not null and m.ACCOUNT\_STATUS='A';

cursor c\_recommendation is select RECOMMENDATION\_ID, REC\_DATETIME, RECOMMENDEE\_MEM\_ID, RECOMMENDER\_MEM\_ID from LI\_recommendations where ACTIVE\_STATUS = 'A';

v\_count integer;

begin

dbms\_output.put\_line('-----Start of result file-----');

--1. update sys time

update LI\_CURRENT\_TIME\_SETTING set current\_time=p\_given\_date\_time;

dbms\_output.put\_line('Simulated system time has been updated');

-- deactivate members based on timestamp

dbms\_output.put\_line('Start of member deactiations:');

v\_count := 0;

for mem in c\_member loop

--exit when c\_member%notfound;

--dbms\_output.put\_line( LIFN\_TimeStampToDate(p\_given\_date\_time) - LIFN\_TimeStampToDate(mem.LAST\_LOGINTIME));

if ( to\_date(LIFN\_TimeStampToDate(LIFN\_TimeStampToDate(p\_given\_date\_time))) - to\_date(LIFN\_TimeStampToDate(LIFN\_TimeStampToDate(mem.LAST\_LOGINTIME)))) >= 365 then

--update LI\_member set account\_status='I' where member\_id = v\_member\_id;

LISP\_DeactivateMember (mem.member\_id); -- call the deactivate user procedure

dbms\_output.put\_line(LIFN\_MemberFullName (mem.member\_id) || ' has been deactivated!');

v\_count := v\_count + 1;

end if;

end loop;

dbms\_output.put\_line('End of member deactiations:');

dbms\_output.put\_line('Number of members deactivated in the request:' || v\_count);

-- Deactivate Recommendations

v\_count := 0;

dbms\_output.put\_line('Start of Endorsement deactiations:');

for rec in c\_recommendation loop

--exit when c\_recommendation%notfound;

if ( to\_date(LIFN\_TimeStampToDate(LIFN\_TimeStampToDate(p\_given\_date\_time))) - to\_date(LIFN\_TimeStampToDate(LIFN\_TimeStampToDate(rec.REC\_DATETIME)))) >= 730 then

update LI\_RECOMMENDATIONS set ACTIVE\_STATUS='I' where RECOMMENDATION\_ID = rec.RECOMMENDATION\_ID;

dbms\_output.put\_line('Recommendation from ' || LIFN\_MemberFullName (rec.RECOMMENDER\_MEM\_ID) || ' for member ' || LIFN\_MemberFullName (rec.RECOMMENDEE\_MEM\_ID) || ' has been deactivated!');

v\_count := v\_count + 1;

end if;

end loop;

dbms\_output.put\_line('End of Endorsement deactiations:');

dbms\_output.put\_line('Number of recommendations deactivated in this request:' || v\_count);

dbms\_output.put\_line('----End of result file-----');

--EXEC LISP\_UpdateCurrentSysTime (to\_timestamp('02-DEC-19', 'DD-MM-YYYY'));

END;

-- Execution Script:

--EXEC LISP\_UpdateCurrentSysTime (to\_timestamp('02-DEC-16', 'DD-MM-YYYY'));

-- select \* from LI\_Member;

-- update LI\_Member set account\_status = 'A';

-- select \* from LI\_Recommendations;

-- update LI\_Recommendations set Active\_Status = 'A';

/\*

17. People You May Know . People You May Know (PYMK) is LinkedIn’s link prediction system and one of the site’s most recognizable features. In this course project, you are asked to design a meaningful method to

implement this feature in the fictional myLinkedIn system. This feature takes an input of a specific member, and the output is a list (i.e., up to 10) of persons the member may know (but has not been connected in

myLinkedIn yet).

\*/

-- Feature 17

create table li\_mid (member\_id integer);

create or replace procedure LISP\_PYMK\_ORGANIZATION (v\_memid int) is

-- feature 17.1

cursor c1 is select lwe2.member\_id from li\_work\_experience lwe1,li\_work\_experience lwe2 where v\_memid = lwe1.member\_id and

lwe1.organization=lwe2.organization and v\_memid <> lwe2.member\_id;

count1 int;

count2 int;

v\_c1mid int;

begin

--recommend someone working in the same company.

open c1;

loop

fetch c1 into v\_c1mid;

exit when c1%notfound;

select count(\*) into count2 from li\_connections where MEMBER\_ID=v\_memid and CONNECTED\_MEMBER\_ID=v\_c1mid;--check if members are already connected

if count2=0 then --members are not connected so can be connected.

insert into li\_mid values(v\_c1mid);

end if;

end loop;

close c1;

end;

/

create or replace procedure LISP\_PYMK\_UNIVERSITY (v\_memid int) is

-- feature 17.2

cursor c1 is select led2.member\_id from li\_education\_details led1,li\_education\_details led2 where v\_memid = led1.member\_id and

led1.university=led2.university and v\_memid<>led2.member\_id;

count1 int;

count2 int;

v\_c1mid int;

begin

--recommend someone working in the same university.

open c1;

loop

fetch c1 into v\_c1mid;

exit when c1%notfound;

select count(\*) into count2 from li\_connections where MEMBER\_ID=v\_memid and CONNECTED\_MEMBER\_ID=v\_c1mid;

if count2=0 then

insert into li\_mid values (v\_c1mid);

end if;

end loop;

close c1;

end;

/

create or replace procedure LISP\_PYMK\_GROUPS (v\_memid int) is

-- feature 17.3

cursor c1 is select lgm.member\_id from li\_group\_members lgm,li\_groups lg1, li\_groups lg2 where v\_memid = lgm.member\_id and

lg1.GROUP\_NAME=lg2.GROUP\_NAME and lgm.group\_id=lg1.group\_id and lgm.group\_id=lg2.group\_id and v\_memid<>lgm.member\_id;

count1 int;

count2 int;

v\_c1mid int;

begin

--recommend someone having same groups.

open c1;

loop

fetch c1 into v\_c1mid;

exit when c1%notfound;

select count(\*) into count2 from li\_connections where MEMBER\_ID=v\_memid and CONNECTED\_MEMBER\_ID=v\_c1mid;

if count2=0 then

insert into li\_mid values (v\_c1mid);

end if;

end loop;

close c1;

end;

/

create or replace procedure LISP\_PYMK\_2degree (v\_memid int) is

-- feature 17.4

cursor c1 is select lic2.member\_id from li\_connections lic1,li\_connections lic2 where v\_memid = lic1.member\_id and

v\_memid<>lic2.member\_id and lic1.member\_id<>lic2.member\_id;

count1 int;

count2 int;

v\_c1mid int;

begin

--recommend someone who is second degree connection for a member.

open c1;

loop

fetch c1 into v\_c1mid;

exit when c1%notfound;

select count(\*) into count2 from li\_connections where MEMBER\_ID=v\_memid and CONNECTED\_MEMBER\_ID=v\_c1mid;

if count2=0 then

insert into li\_mid values (v\_c1mid);

end if;

end loop;

close c1;

end;

/

create or replace procedure LISP\_PYMK\_FINAL (v\_memid int)is

-- Feature 17 Final Procedure

cursor c1 is select \* from li\_mid;

v\_mid int;

cnt int;

count1 int;

begin

select count(\*)into count1 from li\_member where v\_memid=member\_id;--check --validity of member

if count1=0 then dbms\_output.put\_line('Invalid member!');

else

PYMK\_ORGANIZATION(v\_memid);

PYMK\_UNIVERSITY(v\_memid);

PYMK\_GROUPS(v\_memid);

PYMK\_2DEGREE(v\_memid);

delete from li\_mid where rowid not in (select max(rowid) from li\_mid group by member\_id);--remove duplicate member\_id from li\_mid

select count(\*)into cnt from li\_mid;

if cnt=0 then

dbms\_output.put\_line('You are connected to almost everyone! No members available to recommend for connection');

else

open c1;

dbms\_output.put\_line('People you may know: ');

loop

fetch c1 into v\_mid;--fetch member\_id

exit when c1%notfound;

dbms\_output.put\_line(v\_mid);

end loop;

close c1;

end if;

delete from li\_mid;--delete all entries from li\_mid.

end if;

end;

-- Execution Script

EXEC LISP\_PYMK\_FINAL (1001);

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

/\*

Additional common functions:

\*/

-- Function to check if member is valid

CREATE OR REPLACE FUNCTION LIFN\_ValidateMember(p\_member\_id integer) RETURN INTEGER AS

v\_count integer;

BEGIN

select count(\*) into v\_count from LI\_MEMBER where MEMBER\_ID = p\_member\_id and ACCOUNT\_STATUS = 'A';

return v\_count;

END;

-- select LIFN\_Validatemember (1002) from dual;

/

-- Function to get Member Name

create or replace FUNCTION LIFN\_MemberFullName(p\_member\_id integer) RETURN varchar2 AS

v\_count integer;

v\_full\_name varchar2(50);

BEGIN

select first\_name||' '||last\_name into v\_full\_name from LI\_MEMBER, LI\_USER\_MASTER where LI\_MEMBER.MEMBER\_ID = LI\_USER\_MASTER.USER\_ID and MEMBER\_ID = p\_member\_id;

return v\_full\_name;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE (p\_member\_id || ' is not a valid member id');

return 'Not a valid Member';

END;

-- Select LIFN\_MemberFullName(1000) from dual;

-- select LIFN\_MemberFullName (1002) from dual;

-- FUNCTION TO CONVERT FROM TIMESTAMP TO DATE FORMAT

CREATE OR REPLACE FUNCTION LIFN\_TimeStampToDate

(p\_timestamp timestamp)

RETURN date AS

BEGIN

return TO\_DATE (TO\_CHAR (p\_timestamp, 'YYYY-MON-DD HH24:MI:SS'), 'YYYY-MON-DD HH24:MI:SS' );

END;

-- SELECT LIFN\_TimeStampToDate(SYSTIMESAMP) FROM DUAL;