/***

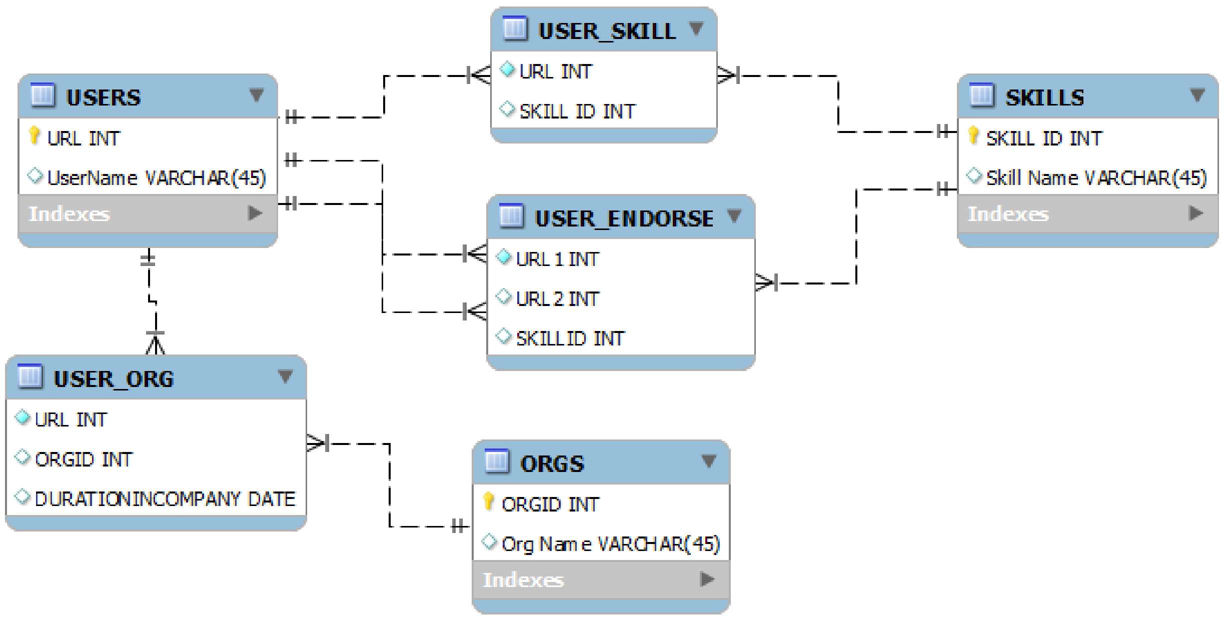
CSE -- Project 2 Report

Authors: *Gaurav Piyush(108996990)*
Sudheer Kumar Vavilapalli(109203795)

*** /

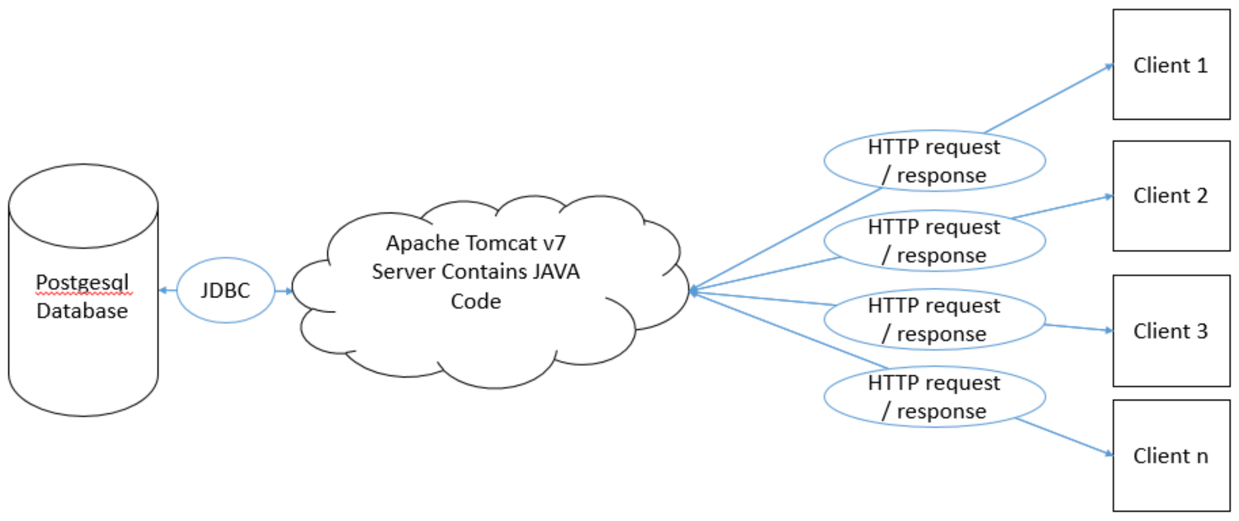
**We pledge our honor that all parts of this project were done by us alone and without collaboration with anybody else.*

Entity-Relationship design:



*Note -
The above ER diagram was generated using MySQL Workbench.

Project Architecture:



Database scheme:

The final schema of the database is finalized after working with the following three schemas. The rejection of first two schemas is due to the reasons mentioned under the corresponding label. The third schema is the final one that is followed for the rest of the project.

Schema 1:

Initially, the following schema has been considered.

User Defined Types:

USERTYPE(URL,Name)
SKILLTYPE(ID, Name)
ENDORSEMENTS(USERTYPE User, SKILLTYPE Skill)
ORGANIZATIONTYPE(Name, StartDate, EndDate)

Relation:

```
LINKEDOUT_USER( USERTYPE User,
                SKILLTYPE Skill ARRAY,
                ENDORSEMENTS E ARRAY,
                ORGANIZATIONTYPE O ARRAY)
```

For the above relation, User is the primary key.

Pros:

This design is completely object oriented which is expected of this project and which uses the maximum object oriented features of postgresql.

Cons:

1.Writing queries is extremely difficult.

2.Selecting USERTYPE as primary key poses two issues. Different URL and same name is considered as different key and same URL and different name is considered as different key which is undesirable.
3. With SKILLTYPE array, the skills cannot be tracked. Any value can be inserted without any constraint and it becomes a legal skill for that user. (Un)Intentional wrong entries may lead to undesirable results.
4. Same with the other two arrays-ORGANIZATIONTYPE and ENDORSEMENTS.

Hence this schema is discarded.

Schema 2:

With schema 1 failed, instead of considering the array of the SKILLTYPE, ORGANIZATIONTYPE, ENDORSEMENTS, the following schema has been considered.

User Defined Types:

USERTYPE(URL,Name)
SKILLTYPE(ID, Name)
ORGTYPE(Name, StartDate, EndDate)

Relations:

USERS Of USERTYPE;
SKILLS Of SKILLTYPE;
ORGS Of ORGTYPE;
USER_SKILL(URL, SkillID ARRAY)
USER_ORG(URL, OrgID ARRAY)
USER_ENDORSE(URL1, URL2, SkillID ARRAY)

Pros:

This schema is more object oriented and less relational using the object oriented feature of postgresql(ARRAY).

Cons:

1. Postgresql doesn’t support array for foreign keys. So, SkillId, OrgID arrays cannot refer to the SKILLS, ORGS relations.
2. Same user can work in same organization at different times. This is not addressed.

Hence this schema is also discarded.

Schema 3:

This is the final design. This is more relational than object oriented. The only object oriented feature of postgresql used is user defined types. But this addresses all the issues that are raised in the above two schemas. There are no anomalies with this design.

User Defined Types:

USERTYPE(URL,Name)
SKILLTYPE(ID, Name)
ORGTYPE(ID, Name)

DURATION(JoinDate, EndDate)

Relations:

USERS Of USERTYPE;
SKILLS Of SKILLTYPE;
ORGS Of ORGTYPE;
USER_SKILL(URL, SkillID)
USER_ORG(URL, OrgID, DurationInCompany ARRAY)
USER_ENDORSE(URL1, URL2, SkillID)

Integrity Constraints:

Entity Integrity:

Relation:	USERS - Primary key: URL
	SKILLS - Primary key: SKILLID
	ORGS - Primary key: ORGID

Referential Integrity:

Relation: USER_SKILL - URL - Reference to URL in relation USERS.
- SKILLID - Reference to SKILLID in relation SKILLS.

USER_ORG - URL - Reference to URL in relation USERS.
- ORGID - Reference to ORGID in relation ORGS.

USER_ENDORSE - URL1, URL2 - Reference to URL in relation USERS.
-SKILLID - Reference to SKILLID in relation SKILLS.

Check Constraints:

Relation:	USERS - Check if URL > 0
	SKILLS - check if SKILLID > 0
	ORGS - Check if ORGID >0
	USER_SKILL - URL not null
	USER_ORG - URL not null
	USER_ENDORSE - URL1 not null

POSTGRESQL QUERIES

Create Statements

Based on the schema the following types were created:
User type consists of URL and username. URL is unique identity of the user so it is made as the primary key when the table was created using this type.

```
CREATE TYPE USERTYPE AS(
	URL INT,
	USERNAME VARCHAR(80)
);
```

Skill type consists of skill id and skill name. The skill id is unique to every skill and is hence made the primary key when table Skill table is created.

```
CREATE TYPE SKILLTYPE AS(
	SKILLID INT,
	SKILLNAME VARCHAR(80)
);
```

Org type consists of the id and name of the organization. The orgid is unique to an organization and hence is the choice for a primary key.

```
CREATE TYPE ORGTYPE AS(
	ORGID INT,
	ORGNAME VARCHAR(80)
);
```

A duration type is created to signify the start and end date of an employee in an organization.

```
CREATE TYPE DURATION AS(
	JOINDATE DATE,
	ENDDATE DATE
);
```

Based on types tables were designed:

```
CREATE TABLE USERS OF USERTYPE(
	PRIMARY KEY(URL)
);

CREATE TABLE SKILLS OF SKILLTYPE(
	PRIMARY KEY(SKILLID)
);

CREATE TABLE ORGS OF ORGTYPE(
	PRIMARY KEY(ORGID));
```

Following tables derive data from the above tables:

USER_SKILL table defines association of a user with a particular skill.

```
CREATE TABLE USER_SKILL(
	URL INT NOT NULL REFERENCES USERS(URL),
	SKILLID INT REFERENCES SKILLS(SKILLID)
);
```

USER_ORG defines association of user with an organization he works or has worked in. The field DURATIONINCOMPANY was kept as an array as a user could have worked in a particular organization over several periods of time in his/her career.

```
CREATE TABLE USER_ORG(
        URL INT NOT NULL REFERENCES USERS(URL),
        ORGID INT REFERENCES ORGS(ORGID),
        DURATIONINCOMPANY DURATION ARRAY
);
```

USER_ENDORSE defines the relation of a user with other users when one user endorses another user for a particular skill.

```
CREATE TABLE USER_ENDORSE(
        URL1 INT NOT NULL REFERENCES USERS(URL),
        URL2 INT REFERENCES USERS(URL),
        SKILLID INT REFERENCES SKILLS(SKILLID)
);
```

Insert Statements

Commands used to inserting test data (given) into the tables above:

```
INSERT INTO Users VALUES(1,'Alice');
INSERT INTO Users VALUES(2,'Bob');
INSERT INTO Users VALUES(3,'Carol');
INSERT INTO Users VALUES(4,'Dave');
INSERT INTO Users VALUES(5,'Eve');
INSERT INTO Users VALUES(6,'Frank');
INSERT INTO Users VALUES(7,'Mallory');
INSERT INTO Users VALUES(8,'Thor');

INSERT INTO Skills VALUES(1,'Basketball');
INSERT INTO Skills VALUES(2,'Hammer Wielding');
INSERT INTO Skills VALUES(3,'Instructor');
INSERT INTO Skills VALUES(4,'Programming');
INSERT INTO Skills VALUES(5,'Quality Assurance');

INSERT INTO ORGS VALUES(1,'Asgard');
INSERT INTO ORGS VALUES(2,'Ebay');
INSERT INTO ORGS VALUES(3,'Google');
INSERT INTO ORGS VALUES(4,'IBM');
INSERT INTO ORGS VALUES(5,'Microsoft');
INSERT INTO ORGS VALUES(6,'Oracle');
INSERT INTO ORGS VALUES(7,'Paypal');
INSERT INTO ORGS VALUES(8,'Stonybrook');
INSERT INTO USER_ORG VALUES (1,8,
ARRAY[ROW(TO_DATE('2013-01-01','YYYY-MM-DD'),TO_DATE('2013-09-30','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES (2,5,
ARRAY[ROW(TO_DATE('2014-01-01','YYYY-MM-DD'),TO_DATE('2014-12-31','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES (2,8,
ARRAY[ROW(TO_DATE('2013-08-01','YYYY-MM-DD'),TO_DATE('2013-12-31','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(3,4,
ARRAY[ROW(TO_DATE('2013-01-06','YYYY-MM-DD'),TO_DATE('2013-12-31','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(4,4,
ARRAY[ROW(TO_DATE('2013-01-06','YYYY-MM-DD'),TO_DATE('2013-07-15','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(5,3,
ARRAY[ROW(TO_DATE('2012-04-01','YYYY-MM-DD'),TO_DATE('2014-02-01','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(5,7,
ARRAY[ROW(TO_DATE('2008-01-05','YYYY-MM-DD'),TO_DATE('2012-03-01','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(6,3,
ARRAY[ROW(TO_DATE('2012-06-01','YYYY-MM-DD'),TO_DATE('2014-01-01','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(6,2,
ARRAY[ROW(TO_DATE('2006-12-05','YYYY-MM-DD'),TO_DATE('2012-05-20','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(7,2,
ARRAY[ROW(TO_DATE('2006-12-05','YYYY-MM-DD'),TO_DATE('2007-12-20','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(7,6,
ARRAY[ROW(TO_DATE('2008-08-30','YYYY-MM-DD'),TO_DATE('2014-03-30','YYYY-MM-DD'))
]::DURATION[]);
```

```
INSERT INTO USER_ORG VALUES(7,7,
ARRAY[ROW(TO_DATE('2008-01-05','YYYY-MM-DD'),TO_DATE('2007-12-20','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_ORG VALUES(8,1,
ARRAY[ROW(TO_DATE('0000-01-01','YYYY-MM-DD'),TO_DATE('9999-12-31','YYYY-MM-DD'))
]::DURATION[]);

INSERT INTO USER_SKILL VALUES(1,3);
INSERT INTO USER_SKILL VALUES(1,4);
INSERT INTO USER_SKILL VALUES(2,4);
INSERT INTO USER_SKILL VALUES(4,4);
INSERT INTO USER_SKILL VALUES(6,4);
INSERT INTO USER_SKILL VALUES(7,4);
INSERT INTO USER_SKILL VALUES(7,1);
INSERT INTO USER_SKILL VALUES(8,2);
INSERT INTO USER_SKILL VALUES(8,3);
INSERT INTO USER_SKILL VALUES(8,4);

INSERT INTO USER_ENDORSE VALUES(2,1,3);
INSERT INTO USER_ENDORSE VALUES(2,1,4);
INSERT INTO USER_ENDORSE VALUES(2,3,4);
INSERT INTO USER_ENDORSE VALUES(3,1,4);
INSERT INTO USER_ENDORSE VALUES(3,2,4);
INSERT INTO USER_ENDORSE VALUES(4,3,4);
INSERT INTO USER_ENDORSE VALUES(6,5,4);
INSERT INTO USER_ENDORSE VALUES(6,5,5);
INSERT INTO USER_ENDORSE VALUES(7,5,4);
INSERT INTO USER_ENDORSE VALUES(7,6,4);
INSERT INTO USER_ENDORSE VALUES(7,7,4);
INSERT INTO USER_ENDORSE VALUES(8,4,4);
```

Queries:

Query1: Find all pairs of different users such that user 1 endorsed user 2 for some skill. Furthermore, both users must be working for the same organization on September 18, 2013.

```
SELECT DISTINCT U1.USERNAME, U2.USERNAME
FROM USERS U1, USERS U2, USER_ENDORSE UE1, USER_ORG UO1, USER_ORG UO2
WHERE
U1.URL = UE1.URL1 AND U2.URL = UE1.URL2 AND U1.URL <> U2.URL AND
UO1.URL = UE1.URL1 AND
UO1.ORGID = UO2.ORGID AND
(UO1.DURATIONINCOMPANY)[1].JOINDATE < TO_DATE('2013-09-18','YYYY-MM-DD') AND
(UO1.DURATIONINCOMPANY)[1].ENDDATE >  TO_DATE('2013-09-18','YYYY-MM-DD') AND
UO2.URL = UE1.URL2 AND
(UO2.DURATIONINCOMPANY)[1].JOINDATE < TO_DATE('2013-09-18','YYYY-MM-DD') AND
(UO2.DURATIONINCOMPANY)[1].ENDDATE >  TO_DATE('2013-09-18','YYYY-MM-DD');
```

RESULT -

```
        Frank endorses Eve
        Bob endorses Alice
```

Query2: Find all pairs of different users such that user 1 endorsed user 2 for a skill that user 1 also has and both of these users were endorsed for that same skill by the same third user.

First we create a view to display all users (user 1) that were endorsed by another user (user 2) for a skill that user 1 also had.

```
CREATE OR REPLACE VIEW SKILLENDORSEDTHATUSERHAS (URL1, URL2, SKILLID) AS
SELECT U1.URL, U2.URL, US1.SKILLID
FROM USERS U1, USERS U2, USER_ENDORSE UE1, USER_SKILL US1, SKILLS S
WHERE U1.URL = UE1.URL1 AND U2.URL = UE1.URL2 AND U1.URL <> U2.URL AND
US1.URL = U1.URL AND US1.SKILLID = UE1.SKILLID AND S.SKILLID=US1.SKILLID;
```

We use the result of view created above to determine the users from the above view that were endorsed by a third user having the same skill as outputted by the above view.

```
SELECT DISTINCT U1.USERNAME, U2.USERNAME
FROM SKILLENDORSEDTHATUSERHAS SE, USERS U1, USERS U2, USERS U3,
USER_ENDORSE UE1,
USER_ENDORSE UE2, SKILLS S
WHERE U1.URL = SE.URL1 AND U2.URL = SE.URL2 AND UE1.URL1 = U3.URL AND
UE1.URL1 <> UE1.URL2 AND UE1.URL2 = SE.URL1 AND UE2.URL1 = U3.URL AND
UE2.URL2 = SE.URL2 AND UE2.URL1 <> UE2.URL2 AND UE1.SKILLID = SE.SKILLID AND
UE2.SKILLID = SE.SKILLID AND S.SKILLID = UE2.SKILLID;
```

RESULT -

```
Bob endorses Alice
Frank endorses Eve
```

Query3. Find all users who do not have a certain skill, but who were endorsed for that skill by at least two other users.

We create a view to find all users that were endorsed by other users for a skill. We output the user 1 url and the skill id of the skill they were endorsed for.

```
CREATE OR REPLACE VIEW USERSENDORSEUSER (URL,SKILLID) AS
SELECT U1.URL,UE1.SKILLID
FROM USERS U1, USERS U2, USERS U3, USER_ENDORSE UE1, USER_ENDORSE UE2,
SKILLS S
WHERE UE1.URL1 = U2.URL AND UE1.URL2 = U1.URL AND UE1.SKILLID = S.SKILLID AND
UE2.URL1 = U3.URL AND UE2.URL2 = U1.URL AND UE2.SKILLID = S.SKILLID AND
UE1.SKILLID = UE2.SKILLID;
```

We use the result of the above created view to find the users that do not have a skill for which they were endorsed.

```
SELECT DISTINCT U1.USERNAME
FROM USERSENDORSEUSER UEU, USERS U1, SKILLS S
WHERE UEU.SKILLID = S.SKILLID AND U1.URL = UEU.URL AND
UEU.SKILLID NOT IN
(SELECT US.SKILLID
FROM USER_SKILL US
WHERE US.SKILLID = S.SKILLID AND U1.URL = US.URL);
```

RESULT -

```
Carol
Eve
```

Query4. Find all pairs of different users u1; u2 such that u1 is more skilled than u2. More skilled here means that all of the following conditions apply:

a. u1 has every skill that u2 has.

b. u1 has a skill that u2 does not have.

Create a view that contains the list of users such that user 1 has a particular skill that user 2 does not have.

```
CREATE OR REPLACE VIEW SKILLUSER1NOTINUSER2(URL1,URL2) AS
SELECT US1.URL,US2.URL
FROM USER_SKILL US1, USER_SKILL US2, SKILLS S
WHERE
US2.SKILLID = S.SKILLID AND
US1.URL NOT IN (SELECT U.URL FROM USER_SKILL U WHERE U.SKILLID = S.SKILLID);
```

Create a view that lists pair of users such that user 2 is not in USER_SKILL table i.e. user 2 does not have any skills.

```
CREATE OR REPLACE VIEW MORESKILLEDDEF(URL1,URL2) AS
SELECT U1.URL,U2.URL
FROM USERS U1,USERS U2
WHERE U2.URL NOT IN(SELECT URL FROM USER_SKILL) AND
          U1.URL IN(SELECT URL FROM USER_SKILL);
```

Create a view that lists a pair of users such that user 1 is more skilled than user 2.

```
CREATE OR REPLACE VIEW MORESKILLED12(URL1,URL2) AS
SELECT US1.URL,US2.URL
FROM USER_SKILL US1, USER_SKILL US2,SKILLS S,SKILLUSER1NOTINUSER2 SNU
WHERE
US1.URL = SNU.URL2 AND US2.URL = SNU.URL1 AND
US1.URL NOT IN (SELECT SNU1.URL1 FROM SKILLUSER1NOTINUSER2 SNU1 WHERE
SNU1.URL1 = US1.URL AND SNU1.URL2 = US2.URL);
```

Here we create a view that includes all set of users that are more skilled than a user that has no skill at all. We also combine the result of the above view (MORESKILLED12) to list all users that are more skilled than other users.

```
CREATE OR REPLACE VIEW MORESKILLED(URL1,URL2) AS
SELECT U1.URL,U2.URL
```

```
FROM USERS U1,USERS U2, MORESKILLED12 MS, MORESKILLEDDEF MSD
WHERE
(U1.URL = MS.URL1 AND U2.URL = MS.URL2) OR
(U1.URL = MSD.URL1 AND U2.URL = MSD.URL2);
```

Finally we list the names of the users present in the view MORESKILLED.

```
SELECT DISTINCT U1.USERNAME, U2.USERNAME
FROM USERS U1, USERS U2, MORESKILLED MS
WHERE
U1.URL = MS.URL1 AND U2.URL = MS.URL2;
```

RESULT -

```
        Alice is more skilled than Bob
        Alice is more skilled than Carol
        Alice is more skilled than Dave
        Alice is more skilled than Eve
        Alice is more skilled than Frank
        Bob is more skilled than Carol
        Bob is more skilled than Eve
        Dave is more skilled than Carol
        Dave is more skilled than Eve
        Frank is more skilled than Carol
        Frank is more skilled than Eve
        Mallory is more skilled than Bob
        Mallory is more skilled than Carol
        Mallory is more skilled than Dave
        Mallory is more skilled than Eve
        Mallory is more skilled than Frank
        Thor is more skilled than Alice
        Thor is more skilled than Bob
        Thor is more skilled than Carol
        Thor is more skilled than Dave
        Thor is more skilled than Eve
        Thor is more skilled than Frank
```

Query5. Find all pairs of users u1; u2 such that u1 is more certied than u2. More certified means that all of the following conditions apply:

a. u1 is more skilled than u2.

b. u1 was endorsed for each of his/her skills.

We create a view to list users that have a skill that was not endorsed by any other user.

```
CREATE OR REPLACE VIEW SKILLNOTENDORSED(URL) AS
SELECT DISTINCT U.URL
FROM USERS U,USER_SKILL US
WHERE
US.URL = U.URL AND
U.URL NOT IN(SELECT URL2 FROM USER_ENDORSE WHERE SKILLID = US.SKILLID) AND
U.URL IN(SELECT URL FROM USER_SKILL WHERE SKILLID = US.SKILLID);
```

We then use the view (MORESKILLED) created while writing query 4 and also the above created view (SKILLNOTENDORSED) to create the following view. Here we take out the users that are present in SKILLNOTENDORSED view and whatever left after is the result which is displayed by the SELECT statement that follows the view.

```
CREATE OR REPLACE VIEW MORECERTIFIED(URL1, URL2) AS
SELECT DISTINCT U1.URL,U2.URL
FROM USERS U1, USERS U2, MORESKILLED MS
WHERE
MS.URL1 = U1.URL AND MS.URL2 = U2.URL AND
U1.URL NOT IN (SELECT URL FROM SKILLNOTENDORSED);

SELECT DISTINCT U1.USERNAME, U2.USERNAME
FROM USERS U1, USERS U2, MORECERTIFIED MC
WHERE
U1.URL = MC.URL1 AND U2.URL = MC.URL2;
```

RESULT -

```
Alice is more certified than Bob
Alice is more certified than Carol
Alice is more certified than Dave
Alice is more certified than Eve
Alice is more certified than Frank
```

```
Bob is more certified than Carol
Bob is more certified than Eve
Dave is more certified than Carol
Dave is more certified than Eve
Frank is more certified than Carol
Frank is more certified than Eve
```

Query6. Find all instances of indirect endorsement as follows: x is an indirect endorser of y iff there is a chain x = a[0], a[1], ..., a[n-1], a[n] = y, where n > 0, such that each a[i+1] is (directly) endorsed by a[i] (for some skill), but not the other way around (i.e., a[i+1] didn't endorse a[i] for any skills). The answer must be a set of all pairs (x; y) such that y is indirectly endorsed by x.

We create a recursive view to find users that were indirectly endorsed by other users. The base case is to find users that were directly endorsed by other users and not vice versa.

```
CREATE OR REPLACE RECURSIVE VIEW INDIRECTENDORSEMENT(URL1,URL2) AS
SELECT UE1.URL1,UE1.URL2
FROM USER_ENDORSE UE1
WHERE UE1.URL1 NOT IN(SELECT UE2.URL2 FROM USER_ENDORSE UE2 WHERE
UE2.URL1=UE1.URL2 AND UE2.URL2=UE1.URL1)
UNION
SELECT UE.URL1,IE.URL2
FROM USER_ENDORSE UE, INDIRECTENDORSEMENT IE
WHERE UE.URL2 = IE.URL1;
```

We use the select statement to display the names of users present in the above recursive view.

```
SELECT DISTINCT U1.USERNAME, U2.USERNAME
FROM USERS U1, USERS U2, INDIRECTENDORSEMENT IE
WHERE
U1.URL = IE.URL1 AND U2.URL = IE.URL2;
```

RESULT -

```
Bob indirectly endorses Alice
Carol indirectly endorses Alice
Dave indirectly endorses Alice
Dave indirectly endorses Carol
Frank indirectly endorses Eve
Mallory indirectly endorses Eve
Mallory indirectly endorses Frank
Thor indirectly endorses Alice
Thor indirectly endorses Carol
Thor indirectly endorses Dave
```

Query7. Similar to the previous query but, in addition, for each 0 to i < n, a[i] should be more skilled than a[i+1].

Here our base case is the list of pair of users s.t. user 1 endorses user 2 for a skill but not vice versa and user 1 is also more skilled than user 2. We used the view (MORESKILLED) created for query 4 to get the pairs of skilled users w.r.t each other.

```
CREATE OR REPLACE RECURSIVE VIEW INDIRECTENDORSEMENTSKILL(URL1,URL2)
AS
SELECT UE1.URL1,UE1.URL2
FROM USER_ENDORSE UE1, MORESKILLED MS
WHERE UE1.URL1 NOT IN(SELECT UE2.URL2 FROM USER_ENDORSE UE2 WHERE
UE2.URL1=UE1.URL2 AND UE2.URL2=UE1.URL1) AND
UE1.URL1 = MS.URL1 AND UE1.URL2=MS.URL2
UNION
SELECT UE.URL1,IES.URL2
FROM USER_ENDORSE UE, INDIRECTENDORSEMENTSKILL IES
WHERE UE.URL2 = IES.URL1;
```

We list the names of the users listed in the above view using the select view.

```
SELECT DISTINCT U1.USERNAME, U2.USERNAME
FROM USERS U1, USERS U2, INDIRECTENDORSEMENTSKILL IES
WHERE
U1.URL = IES.URL1 AND U2.URL = IES.URL2;
```

RESULT -

```
Dave indirectly endorses Carol
Frank indirectly endorses Eve
Mallory indirectly endorses Eve
Mallory indirectly endorses Frank
Thor indirectly endorses Carol
Thor indirectly endorses Dave
```

Note :

In JAVA we used executeQuery() to get all the rows while making SELECT queries to postgresql. We used executeUpdate() to create the views required to obtain the results for the above listed 7 queries. Once the result set was generated we pass it using request attributes to the frontend where they were handled as an array of elements using the jstl core libraries.

User Guide

Platform -

Windows 8 64 bit

Windows 7 64 bit

Installation -

1. Prerequisites
   a. Eclipse IDE
      Download Eclipse Keplar from here.
      Extract the files.
      Open Eclipse.exe to run the IDE
   b. Apache Tomcat Server
      Download Apache Tomcat 7 for Windows 64 bit machines from here.
      Extract the files.
      Open Eclipse
      Go to Windows > Show View > Other..
      Type Servers and hit Enter.
      In the server view right click and choose New > Server
      Choose Apache Tomcat 7.0 Server click Finish.
      You may be asked to specify the location of Apache Tomcat server. Just specify the location of your Apache Tomcat directory.
   c. postgresql server
      Download postgresql installer for Windows 64 bit from here.
      Run the .exe file and follow steps to install.
      Set the password if asked for.

2. Initializing the database
   a. Open cmd
   b. Type the following
      psql.exe -h 127.0.0.1 -U postgres
   c. You will be prompted to postgres console where you will be asked for password. Type in the password used during the time of installation.
   d. Create the LinkedOut database.
   e. Type \q to exit.
   f. Now type the following command to open LinkedOut database
      psql.exe -h 127.0.0.1 -U postgres linkedout
   g. Create table and insert values into tables using the commands (CREATE and INSERT) mentioned in the section for POSTGRESQL QUERIES. After you have done this the database is created and initialized.

3. Running the Server
   a. Open eclipse and open LinkedOut project in it.
   b. Right click on Project name and click Run As > 1 Run on Server

c. Choose Apache Tomcat 7 Server from existing servers and hit Next. Add the project in the list of projects to be run on the server and then hit Finish.

4. Running the Client
    a. Open any browser.
    b. Type the following link
        http://localhost:8080/LinkedOut/
        Apache Tomcat is running on localhost on port 8080 over http protocol
        Name of the project is LinkedOut

DIVISION OF WORK:

Both of us worked on the schema design. Once the schema design is finalised, explored postgresql, understood the syntax and started writing the queries. Both of us discussed and developed the web backend.
Gaurav Piyush: Query 2, Query 4, Query 6 and their corresponding work in ApiManager.java & index.jsp.
Sudheer Kumar Vavilapalli: Query 1, Query 3, Query 5, Query7 and their corresponding work in ApiManager.java & index.jsp.

References:

[1] Postgresql documentation:
http://www.postgresql.org/docs/9.3/static/tutorial.html
[2] Postgresql JDBC documentation:
http://jdbc.postgresql.org/documentation/head/index.html