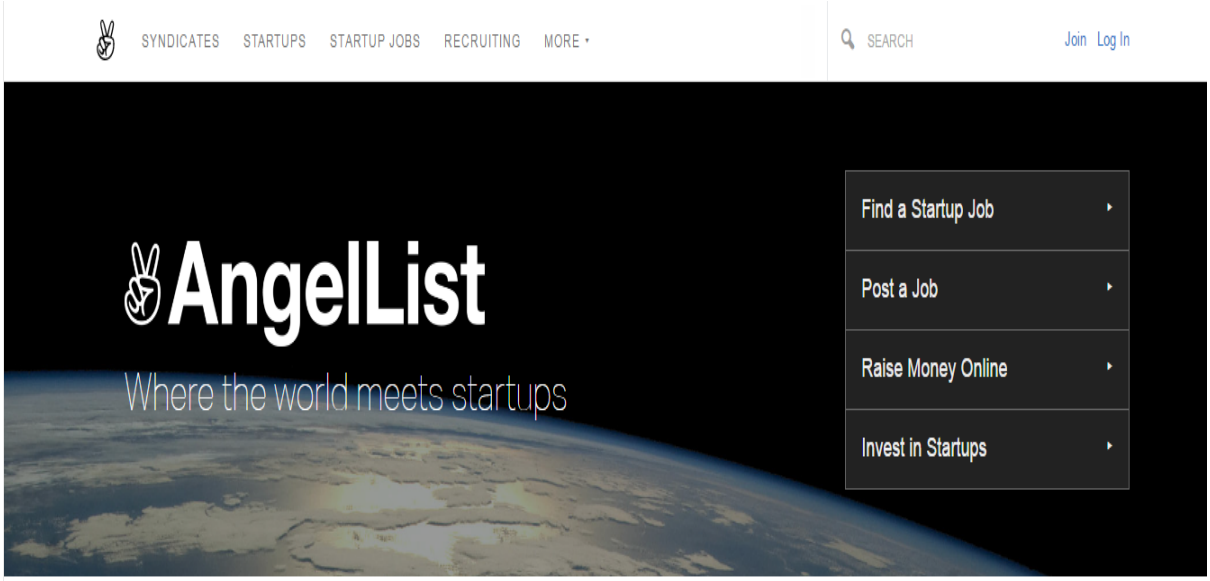


Database Design for Angel List

CS 6360.002 – Final Project

Table of Contents:

REQUIREMENTS:

What does AngelList do?

AngelList is a platform for start-ups to raise money online, recruit employees, and apply for funding without any other middleman.

Objectives and Functions of AngelList:
There are three main services provided by AngelList:

1. Finding a job at a start-up.
    a. Applicant applies for a job.
2. Posting a job (by start-up).
    a. Recruited posts a job.
    b. Company posts a job
3. Investing in the start-ups.
    a. Start-ups has different stages of funding.
    b. An investor invests in the Start-ups.

Other functionalities apart from these three on a sub-level:

1. A person can follow another person.
2. A person can message another person.
3. A person can follow a company.

Start-up(Company):

A start up is similar to a small company. Each start-up has a management team comprising of a CEO and other subordinates, which we are calling employees here.

Person:

According to our analysis of AngelList there are 4 types of people who can be involved in the operations in the AngelList, each kind of person has specific functions, there is an overlapping relation between all the 4 types:

- Employee: An employee is an entity on AngelList. An employee relates to a company or a start-up. An Employee can have different functions in the company (for e.g. CEO, CTO, supervisor, intern etc.). There are two major relations of an employee 1) **Manages** – an employee could be a manager of a company, this detail is shown on the company portfolio page hence an exclusive relation is required. 2) **Works_For** - An employee could be working for any company.

- Applicant: An applicant is a person who applies or wishes to apply for a job at a start-up. This is the core of AngelList as its one of the main features is to connect applicants to the start-ups without any middleman. An applicant mainly have one attribute exclusive to itself i.e. projects. It is similar to having a resume listing all the work experience and the projects a person has done in the past.

- Recruiter: A recruiter is a person whose main job is to look over the recruitment process of a company or a start-up. There could be two types of recruiters 1) Working for the company posting the jobs. 2) Third Party Recruiters: People who connect the applicants to the companies who look to hire people. The main relation of a recruiter hence is recruiting for a job posting.

- Investor: Investor is the prime component of the third function of AngelList i.e. 'Investing in the start-ups'. An investment in a start-up can be done in two ways:

    - By a company.
    - *By a person*.

  Here the term investor which is part of the specialization of a person is the one from the second category mentioned above. An investor can be of different types and is assigned a Investment ID based on the investment made.

Each person has two weak entities identified by itself

- Experience: A person can have certain years of experience which is reflected on his profile portfolio page at AngelList.
- Messages: A person can message another person, this is similar to any social media feature which involves exchange of texts between two different people.

Investment:

Every start-up is at some stage of funding which they require to keep up with their functions. AngelList helps investors in investing in the start-ups they think which could be profitable to them based on the investments made in a start-up the value of a start-up changes. An investment in a start-up can be done in two ways:

- Investment by an Investor: When a single person want to invest in a start-up.
- Investment by a Company: When a company wants to invest in a start-up.

Jobs:

AngelList works as a middleman for both: applicants as well as the companies(start-ups) in posting the jobs they want to hire people for. AngelList allows companies of different sectors and market areas to post jobs for the positions open in the different vacancies. A job can have different aspects to it like, Job Description, Job Type, Job Location, Job IDs etc. a similar job can be posted by a company for several different locations, each job posted is related with 3 entities:

- Company: Posts the jobs.
- Applicant: Applies for jobs listed.
- Recruiter: Matches applicant to job description and jobs posted.

A job posting also consists of salaries and equity that a hire would be offered, hence AngelList acts as a platform for job posting too.

Activity:

An activity is defined as any kind of content that is posted by a person, company or press. This is similar to the social media features like posts on Facebook or Linked In, tweets on Twitter, posts on Instagram etc. An activity can be of different types:

- A long post (consisting of text and images).
- A comment on someone else's post.
- Status.

An activity could help different types of people in following ways:

- A CEO could posts updates about their quarterly performance of the company or the round of investing they get inducted into.
- An applicant can post his/her requirement regarding the start-up they wish to work for or a company which matches their profile.
- A recruiter could comment on the applicant's post or themselves could post about the job openings at various start-ups .
- An investor could comment/post regarding the already made non-confidential investments.

An activity helps a user of AngelList get involved by posting about the things they wish to share, putting up a status or by commenting on status/posts of others.

We're using Blob for storing content related to images in our database. Every activity has a timestamp in the database to show on the screen about the data/time the activity was posted by a particular type of user.

Connections:

A connection is another social media feature similar to 'being friends' on Facebook, Following on Twitter, Connecting on Linked In etc. AngelList provides this functionality at a person level. A person is allowed to become a connection of another person by sending them connection request via their email if the person they wish to connect with are not part of AngelList at the time the connection request was sent.

The Connection feature helps a user to follow the activities posted by other people. This feature becomes really useful as a person wanting to know the updates or posts of a company or a recruiter then they can do so by connecting with them on AngelList.

SQL Statements to create and alter Tables in DB and Add Constraints:

DDL:

```
CREATE TABLE person (
   personid      INTEGER,
   salary        INTEGER,
   description   VARCHAR2(400),
   location      VARCHAR2(50),
   education     VARCHAR2(50),
   socialmedia   VARCHAR2(100),
   name          VARCHAR2(20),
   PRIMARY KEY ( personid )
);

CREATE TABLE employees (
   personid    INTEGER,
   role        VARCHAR2(100),
   dname       VARCHAR2(50),
   companyid   INTEGER,
   PRIMARY KEY ( personid )
);

CREATE TABLE applicant (
   personid   INTEGER,
   skills     VARCHAR2(200),
   PRIMARY KEY ( personid )
);

CREATE TABLE investor (
   personid   INTEGER,
   type       VARCHAR2(20),
   PRIMARY KEY ( personid )
);

CREATE TABLE recruiter (
   personid   INTEGER,
   PRIMARY KEY ( personid )
);

CREATE TABLE reference (
   personid    INTEGER,
   reference   VARCHAR2(400),
   PRIMARY KEY ( personid )
);

CREATE TABLE company (
   companyid     INTEGER,
   cname         VARCHAR2(20),
   founder       VARCHAR2(20),
   description   VARCHAR2(400),
   funding       INTEGER,
   empcount      INTEGER,
   website       VARCHAR2(100),
   startdate     DATE,
   managerid     INTEGER,
   PRIMARY KEY ( companyid )
);

CREATE TABLE companyfunding (
```

```
   funding   INTEGER,
   stage     VARCHAR2(10),
   PRIMARY KEY ( funding )
);

CREATE TABLE companylocation (
   companyid   INTEGER,
   location    VARCHAR2(20),
   PRIMARY KEY ( companyid )
);

CREATE TABLE job (
   jobid           INTEGER,
   jobdescription   VARCHAR2(400),
   jobpostdate      DATE,
   personid         INTEGER,
   companyid        INTEGER,
   jobname          VARCHAR2(20),
   location         VARCHAR2(20),
   category         VARCHAR2(20),
   jobtype          VARCHAR2(20),
   salary           INTEGER,
   equity           DECIMAL(2,2),
   PRIMARY KEY ( jobid )
);

CREATE TABLE message (
   messageid   INTEGER,
   personid    INTEGER,
   content     BLOB,
   timestamp   DATE,
   PRIMARY KEY ( messageid,
                 personid )
);

CREATE TABLE experience (
   personid       INTEGER,
   experienceid   INTEGER,
   companyid      INTEGER,
   duration       VARCHAR2(20),
   description    VARCHAR2(200),
   title          VARCHAR2(50),
   PRIMARY KEY ( experienceid,
                 personid )
);

CREATE TABLE activity (
   activityid     INTEGER,
   activitydate   DATE,
   content        BLOB,
   type           VARCHAR2(20),
   personid       INTEGER,
   companyid      INTEGER,
   PRIMARY KEY ( activityid )
);

CREATE TABLE activityowner (
   personid    INTEGER,
   companyid   INTEGER,
   owner       VARCHAR2(20),
   PRIMARY KEY ( personid,
                 companyid )
);
```

```
CREATE TABLE follows (
   personid    INTEGER,
   companyid   INTEGER,
   PRIMARY KEY ( personid,
                companyid )
);

CREATE TABLE investment (
   personid          INTEGER,
   companyid         INTEGER,
   amount            INTEGER,
   investmentid      INTEGER,
   stage             VARCHAR2(20),
   dateofinvestment  DATE,
   description       VARCHAR2(400),
   PRIMARY KEY ( personid,
                companyid )
);

CREATE TABLE appliesfor (
   personid          INTEGER,
   jobid             INTEGER,
   dateofapplication DATE,
   PRIMARY KEY ( personid,
                jobid )
);

CREATE TABLE connection (
   personid1   INTEGER,
   personid2   INTEGER,
   PRIMARY KEY ( personid1,
                personid2 )
);

CREATE TABLE companytype (
   companyid   INTEGER,
   type        VARCHAR2(20),
   PRIMARY KEY ( companyid )
);

CREATE TABLE product (
   companyid   INTEGER,
   product     VARCHAR2(40),
   PRIMARY KEY ( companyid )
);

CREATE TABLE companyexperience ( --  companyID is multivalued attribute
   companyid      INTEGER,
   personid       INTEGER,
   experienceid   INTEGER,
   PRIMARY KEY ( experienceid,
                personid,
                companyid )
);
```

DML:

```
alter table employees add CONSTRAINT fke1 FOREIGN KEY (personid) REFERENCES person(personid)
DEFAULT 1;
alter table employees add CONSTRAINT fke2 FOREIGN KEY (companyID) REFERENCES
company(companyID) ON DELETE CASCADE;
```

alter table applicant add CONSTRAINT fke3 FOREIGN KEY (personid) REFERENCES person(personid) ON DELETE CASCADE;

alter table investor add CONSTRAINT fke4 FOREIGN KEY (personid) REFERENCES person(personid) ON DELETE CASCADE;

alter table recruiter add CONSTRAINT fke5 FOREIGN KEY (personid) REFERENCES person(personid) ON DELETE CASCADE;

alter table company add CONSTRAINT fke6 FOREIGN KEY (MANAGERID) REFERENCES person(personid) ON DELETE SET NULL;

alter table job add CONSTRAINT fke7 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;
alter table job add CONSTRAINT fke8 FOREIGN KEY (COMPANYID) REFERENCES company(COMPANYID) ON DELETE SET NULL;

alter table message add CONSTRAINT fke9 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;

alter table experience add CONSTRAINT fke10 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;

alter table activity add CONSTRAINT fke11 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;
alter table activity add CONSTRAINT fke12 FOREIGN KEY (COMPANYID) REFERENCES company(COMPANYID) ON DELETE CASCADE;

alter table activityowner add CONSTRAINT fke13 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;
alter table activityowner add CONSTRAINT fke14 FOREIGN KEY (COMPANYID) REFERENCES company(COMPANYID) ON DELETE CASCADE;

alter table follows add CONSTRAINT fke15 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;
alter table follows add CONSTRAINT fke16 FOREIGN KEY (COMPANYID) REFERENCES company(COMPANYID) ON DELETE CASCADE;

alter table investment add CONSTRAINT fke17 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;
alter table investment add CONSTRAINT fke18 FOREIGN KEY (COMPANYID) REFERENCES company(COMPANYID) ON DELETE CASCADE;

alter table Appliesfor add CONSTRAINT fke19 FOREIGN KEY (PERSONID) REFERENCES person(personid) ON DELETE CASCADE;
alter table Appliesfor add CONSTRAINT fke20 FOREIGN KEY (jobid) REFERENCES job(jobid ON DELETE CASCADE);

alter table connection add CONSTRAINT fke22 FOREIGN KEY (PERSONID1) REFERENCES person(personid) ON DELETE CASCADE;
alter table connection add CONSTRAINT fke21 FOREIGN KEY (PERSONID2) REFERENCES person(personid) ON DELETE CASCADE;

alter table companytype add CONSTRAINT fke23 FOREIGN KEY (companyid) REFERENCES company(companyid) ON DELETE CASCADE;

alter table product add CONSTRAINT fke24 FOREIGN KEY (companyid) REFERENCES company(companyid) ON DELETE CASCADE;

```
alter table companyexperience add CONSTRAINT fke25 FOREIGN KEY (companyid) REFERENCES
company(companyid) ON DELETE CASCADE;
alter table companyexperience add CONSTRAINT fke26 FOREIGN KEY (personid) REFERENCES
person(personid) ON DELETE CASCADE;
alter table companyexperience add CONSTRAINT fke27 FOREIGN KEY (EXPERIENCEID,personid)
REFERENCES experience(EXPERIENCEID,personid) ON DELETE CASCADE;
```

Normalization of Relational Schema:

Below dependencies make the relational schema to be in 3NF:

PERSON {PersonID , Salary, Description, Location, Education, SocialMedia, Name}

EMPLOYEE { PersonID, Role, DepartmentName, CompanyID}

APPLICANTS {PersonID, Skills}

INVESTOR{PersonID, Type}

RECRUITER{PersonID}

COMPANY{CompanyID, Name, Founders, Description, Funding, No.OfEmployees, Website, StartDate,MgrID}

COMPANYFUNDING{Funding, Stage}

COMPANYLOCATION{CompanyID, Location}

JOB{JobID, Description, JobPostDate, PersonID, CompanyID, JobName, Location, Category, Type,
Salary, Equity}

MESSAGE{MessageID, PersonID, Content, Timestamp}

EXPERIENCE {PersonID, ExperienceID, CompanyID, Duration, Duration, Title}

ACTIVITY{ActivityID, Date, Content, Type, PersonID, CompanyID}

ACTIVITYOWNER{PersonID, CompanyID, Owner}

FOLLOWS{PersonID, CompanyID}

INVESTMENT{PersonID, CompanyID, amount, InvestmentID, stage, DateofInvestment, Description}

APPLIESFOR{PersonID, JobID, DateOfApplication}

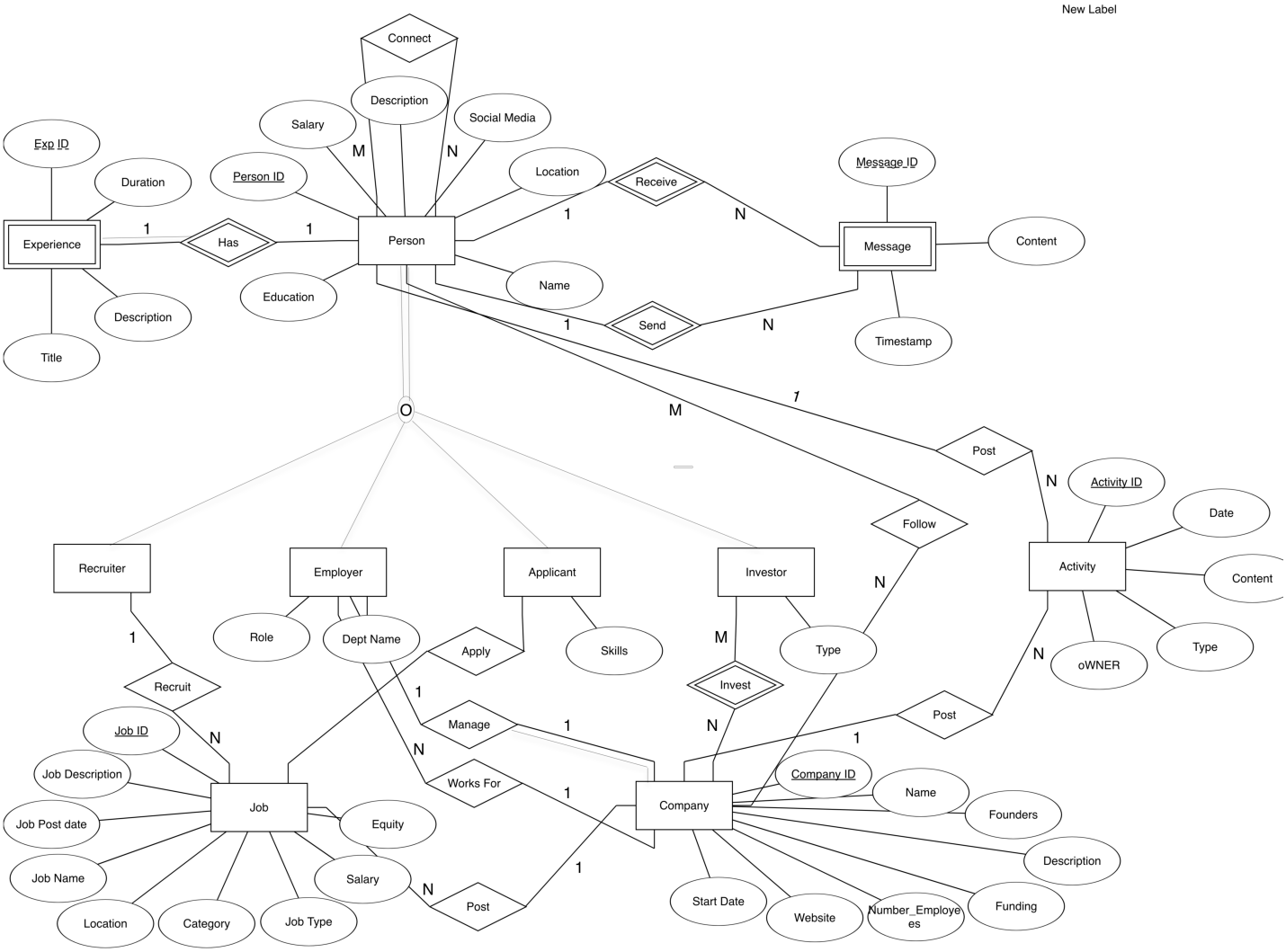
CONNECTS{PersonID1, PersonID2}

COMPANYTYPE{CompanyID, type}

PRODUCT{CompanyID, Product}

COMPANYEXPERIENCE{PersonID, ExperienceID, CompanyID}

Modeling of Requirements as ER-Diagram:



The requirements can be summarized/ derived from ERD as

1. There are two weak entities present : Experience and Message
2. One person can send and receive multiple messages (1:N)
3. One company can post multiple jobs and one recruiter can recruit for multiple jobs.(1:N)
4. Multiple applicants can apply for multiple jobs. Multiple investors can invest in multiple companies and multiple persons can follow multiple companies(M:N)
5. Multiple persons can connect with multiple persons(M:N)
6. More than one Employers work for a company.(M:1)
7. An employee manages a company (1:1)

Mapping of ERD in Relational Schema

Person

Person ID
Salary
Description
Location
Education
Social Media
Name

Primary Key: PersonID Foreign Keys: None

Employee

Person ID (FK)
Role
Dept Name
Company ID(FK)

Primary Key: Person ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID), FOREIGN KEY (Company ID) REFERENCES Company(Company ID)

Applicant

Person ID (FK)
Skills

Primary Key: PersonID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID)

Investor

Person ID(FK)
Type

Primary Key: PersonID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID)

Recruiter

Person ID(FK)

Primary Key: PersonID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID)

Company

Company ID
Name
Founers
Description
Funding
Number of Employess
Website
Start Date
Mgr ID(FK)

Primary Key: Company ID Foreign Keys: FOREIGN KEY (Mgr ID) REFERENCES Person(Person ID)

Company Funding

Funding
Stage

Primary Key: Funding Foreign Keys: None

Company Location

Company ID(FK)
Location

Primary Key: Company ID Foreign Keys: FOREIGN KEY (Company ID) REFERENCES

Job

Job ID
Job Description
Job Post Date
Person ID(FK)
Company ID(FK)
Job Name
Location
Category
Job Type
Salary
Equity

Primary Key: Job ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID), FOREIGN KEY (Company ID) REFERENCES Company(Company

Message

Message ID
Person ID(FK)
Content
Timestamp

Primary Key: Message ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID)

Experience

Person ID(FK)
Exp ID
Company ID
Duration
Description
Title

Primary Key: Person ID, Exp ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES

Activity

Activity ID
Date
Content
Type
Person ID(FK)
Company ID(FK)

Primary Key: Activity ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID),(Company ID) REFERENCES Company(Company ID)

Activity Owner

Person ID(FK)
Company ID(FK)
Owner

Primary Key: Person ID, Company ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID), FOREIGN KEY (Company ID) REFERENCES Company(Company ID)

Follows

Person ID(FK)
Company ID(FK)

Primary Key: Person ID, Company ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID), FOREIGN KEY (Company ID) REFERENCES Company(Company ID)

Investment

Person ID(FK)
Company ID(FK)
Investment ID
Amount
Stage
Date of Investment
Description

Primary Key: Person ID, Company ID, Investment ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID), FOREIGN KEY (Company ID) REFERENCES Company(Company ID)

Applies For

Person ID(FK)
Job ID(FK)

Primary Key: Person ID, Job ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID), FOREIGN KEY (Job ID) REFERENCES Job(Job ID)

Connects

Person1 ID(FK)
Person2 ID(FK)

Primary Key: Person1 ID, Person2 ID Foreign Keys: FOREIGN KEY (Person1 ID) REFERENCES Person(Person ID), FOREIGN KEY (Person2 ID) REFERENCES Person(Person ID),

Company Type

Company ID(FK)
Type

Primary Key: Company ID Foreign Keys: FOREIGN KEY (Company ID) REFERENCES Company(Company ID)

Product

Company ID(FK)
Product

Primary Key: Company ID Foreign Keys: FOREIGN KEY (Company ID) REFERENCES Company(Company ID)

Company Experience

Person ID(FK)
Experience ID(FK)
Company ID(FK)

Primary Key: Person ID, Experience ID, Company ID Foreign Keys: FOREIGN KEY (Person ID) REFERENCES Person(Person ID), FOREIGN KEY (Experience ID) REFERENCES Experience(Experience ID), FOREIGN

Procedures and Triggers:

1. PL/SQL

    1.1. Stored Procedures

1.1.1.Procedure 1: getJobCompFromDesc

Find the Job and corresponding company given specified job description.

- Call: getJobCompFromDesc(description)
- Parameters: Description

```
CREATE OR REPLACE PROCEDURE getJobCompFromDesc(tmp_desp IN JOB.JOBDESCRIPTION%type,
    tmp_jobname OUT JOB.JOBNAME%type,
    tmp_companyname OUT COMPANY.CNAME%type) AS
BEGIN
    SELECT JOBNAME, CNAME INTO tmp_jobname, tmp_companyname
    FROM JOB J NATURAL JOIN COMPANY C
    WHERE J.JOBDESCRIPTION LIKE '%'||tmp_desp||'%';
END;
```

Result of finding Job whose description contains 'Java' and its corresponding Companu.

```
SET SERVEROUTPUT ON;
DECLARE
    jobname JOB.JOBNAME%type;
    comname VARCHAR2(20 BYTE);
BEGIN
getJobCompFromDesc('Java', jobname, comname);
  dbms_output.put_line('Job name:' || jobname);
  dbms_output.put_line('Company name:' || comname);
END;
```

Script Output

Task completed in 0.073 seconds

```
Job name:Java Developer
Company name:VeryLarge


PL/SQL procedure successfully completed.
```

1.1.2.Procedure 2: getJobCompFromSaLo

Find the Job and corresponding company given specified location and a salary that lower bounding the job's salary.

- Call: getJobCompFromSaLo(salary, location, jobname, companyname)
- Parameters: salary, location, jobname, companyname

```
CREATE OR REPLACE PROCEDURE getJobCompFromSaLo(tmp_salary IN JOB.SALARY%type,
    tmp_location IN JOB.LOCATION%type,
    tmp_jobname OUT JOB.JOBNAME%type,
    tmp_companyname OUT COMPANY.CNAME%type) AS
BEGIN
    SELECT JOBNAME, CNAME INTO tmp_jobname, tmp_companyname
    FROM JOB J NATURAL JOIN COMPANY C
    WHERE J.LOCATION LIKE tmp_location
        AND J.SALARY < tmp_salary;
END;
```

Result of finding jobs which is in 'Plano' and has salary larger than 20000.

```
SET SERVEROUTPUT ON;
DECLARE
    jobname JOB.JOBNAME%type;
    cname COMPANY.CNAME%type;
BEGIN
getJobCompFromSaLo(20000, 'Plano', jobname, cname);
  dbms_output.put_line('Job name:' || jobname);
  dbms_output.put_line('Company name:' || cname);
END;
```

Script Output

Task completed in 0.044 seconds

```
Job name:Java Developer
Company name:Large


PL/SQL procedure successfully completed.
```

1.2.Triggers

1.2.1.Trigger 1: highSalary

Whenever the inserted and updated items in JOB table contain a high salary, it gives warning. High salary means salary larger than 50000 for jobtype 'Software Engineer' and salary larger than 30000 for jobtype 'Sales'.

```
CREATE OR REPLACE TRIGGER highSalary
BEFORE UPDATE OR INSERT ON JOB
FOR EACH ROW
DECLARE
BEGIN
    CASE :NEW.JOBTYPE
    WHEN 'Software Engineer' THEN
        IF :NEW.SALARY > 50000 THEN
            dbms_output.put_line('New Salary' || :NEW.SALARY || ' is too high for ' || :NEW.JOBTYPE);
        END IF;
    WHEN 'Sales' THEN
        IF :NEW.SALARY > 30000 THEN
            dbms_output.put_line('New Salary' || :NEW.SALARY || ' is too high for ' || :NEW.JOBTYPE);
        END IF;
    END CASE;
END;
```

Result of negative test case when one item in JOB table is updated to have salary 60000 which is larger than 50000. (Red rectangle bounds the test case code and below is the console output.)

```
UPDATE Job
    set SALARY = 60000
    where JOBID = 1

SET SERVEROUTPUT ON;
DECLARE
    jobname JOB.JOBNAME%type;
    cname COMPANY.CNAME%type;
BEGIN
getJobCompFromSaLo(20000, 'Plano', jobname, cname);
```

Script Output

Task completed in 0.035 seconds

```
New Salary60000 is too high for Software Engineer


1 row updated.
```

1.2.2.Trigger 2: editFounder

Whenever the founder of a company is changed, it gives warning.

```
CREATE OR REPLACE TRIGGER editFounder
AFTER UPDATE OF FOUNDER ON COMPANY
FOR EACH ROW
DECLARE
BEGIN
    dbms_output.put_line('The founder of ' || :NEW.CNAME || ' has been changed from ' || :OLD.FOUNDER || ' to ' || :NEW.FOUNDER);
END;
```

Result of negative test case where ‘Large’ company’s founder has been changed from ‘Ci Ci’ to ‘Jack’. (Red rectangle bounds the test case code and below is the console output.)

```
UPDATE Company
    set founder = 'Jack'
    where founder = 'Ci Ci'

SET SERVEROUTPUT ON;
DECLARE
    jobname JOB.JOBNAME%type,
    cname COMPANY.CNAME%type
BEGIN
getJobCompFromSaLo(, '', jobname, cname);
```

Script Output

Task completed in 0.121 seconds

```
The founder of Large has been changed from Ci Ci to Jack


1 row updated.
```