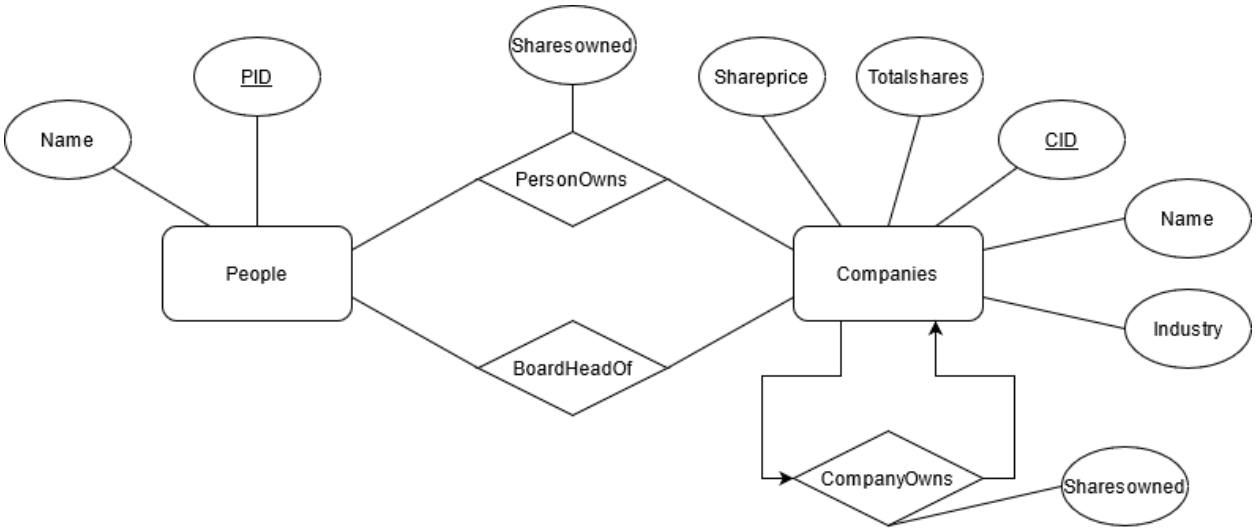
CSE532 Project 2 Report

Alexander Hu

0. Academic Honesty Agreement

Name: Alexander Hu
SBU ID: 110544388
I pledge my honor that all parts of this project were done by me individually, without collaboration with anyone, and without consulting any external sources that provide full or partial solutions to a similar project. I understand that breaking this pledge will result in an “F” for the entire course.

1. Entity Relational Diagram



2. Database Schema

The goal of the object relational model is to be able to represent the database using objects. These objects are reflected by the entities and the relations between them in the entity relational diagram. In particular, the two main classes, “people” and “companies,” are the entities of the aforementioned diagram.

In addition, the third type, “ownshares” is used to represent the relations that both a person and a company could own shares of another company. The type is designed without describing the owner of the shares because that can be described by the class enclosing any retrieved “ownshares” array.

Each relation in the entity relational model is represented as an array in the table creation. That is because each relation is a one to many relation. The industry trait for “companies” is also framed using an array because each company could have more than one industry it is part of.

3. Integrity Constraints

The primary keys for the tables “people” and “companies” is pid and cid, respectively. They are used to uniquely identify each person and company object. Thus, they also cannot be null.

The names of people and companies must also not be null.

The total shares of a company as well as the company’s price must be larger than zero, as having zero or negative total shares or share price wouldn’t make any sense in this context.

Each company must have some industry they are part of, otherwise it would imply the company doesn’t do anything at all.

One major weakness of setting up the table schema in this manner is that while it fulfills the ability to treat tables as objects with attributes, in postgresql in particular it cannot maintain certain relational restraints, especially in regards to both types and arrays. Thus, although the cid in personowns[ ] for the people table, the cid in companyowns[ ] for the companies table, and the pid for boardheads[ ] for the companies all should have foreign key relational constraints with the relevant pid or cid primary key, it is not possible to set it up in this context.

If, instead of using types and arrays, entire new tables with individual entries were used here, it would’ve been possible to set up the relevant foreign key constraints. However, at that point, the object relational schema should be given up for a more standard normalized table schema, which utilizes tables in such a context by default.

4. Database Population Commands

```
CREATE TABLE people (

    pid             integer NOT NULL PRIMARY KEY,
    name            text NOT NULL,
    personowns      ownshares[]

);

CREATE TABLE companies (

    cid             integer NOT NULL PRIMARY KEY,
```

```
    name            text NOT NULL,
    totalshares     integer CHECK (totalshares > 0),
    shareprice      integer CHECK (shareprice > 0),
    industry        text[] CHECK (array_length(industry, 1) > 0),
    companyowns     ownshares[],
    boardheads      integer[]

);

CREATE TYPE ownshares AS (

    cid             integer,
    sharesowned     integer

);


INSERT INTO companies
(cid,name,totalshares,shareprice,industry,companyowns,boardheads)
VALUES
(1,'QUE',150000,30,'{"Software","Accounting"}',
'{"(2,10000)","(4,20000)","(8,30000)"}','{3,1,4}'),
(2,'RHC',250000,20,'{"Accounting"}','{}','{1,2,5}'),
(3,'Alf',10000000,700,'{"Software","Automotive"}','{"(9,-100000)",
"(4,400000)","(8,100000)"}','{6,7,1}'),
(4,'Elgog',10000000,400,'{"Software","Search"}','{"(6,5000)"}',
'{6,7,5}'),
(5,'Tfos',10000000,300,'{"Software","Hardware"}','{"(6,30000)",
"(7,50000)","(1,200000)"}','{2,5,4}'),
(6,'Ohay',180000,50,'{"Search"}','{}','{2,4,8}'),
(7,'Gnow',150000,300,'{"Search"}','{}','{2,3,4}'),
(8,'Elpa',9000000,300,'{"Software","Hardware"}','{"(5,20000)",
"(4,30000)"}','{2,3,8}'),
(9,'Ydex',5000000,100,'{"Software","Search"}','{}','{6,3,8}');

INSERT INTO people (pid,name,personowns) VALUES
(1,'Bill Doe','{"(5,30000)","(8,100000)"}'),
(2,'Bill Seth','{"(7,40000)","(4,20000)"}'),
(3,'John Smyth','{"(1,20000)","(2,20000)","(5,800000)"}'),
(4,'Anne Smyle','{"(2,30000)","(5,40000)","(3,500000)"}'),
(5,'Steve Lamp','{"(8,90000)","(1,50000)","(6,50000)","(2,70000)"}'),
(6,'May Serge','{"(8,-10000)","(9,-40000)","(3,500000)",
"(2,40000)"}'),
(7,'Bill Public','{"(7,80000)","(4,30000)","(1,30000)","(5,300000)",
"(2,-9000)"}'),
```

```
(8,'Muck Lain','{"(2,60000)","(6,-40000)","(9,-80000)",
"(8,30000)"}');
```

5. Query Code + Explanation

Query 1:

```
SELECT c.name AS CompanyName
FROM
    companies c,
    people p,
    unnest(p.personowns) AS po
WHERE
    p.pid = ANY(c.boardheads) AND
    po.cid = c.cid AND
    po.sharesowned > 0;
```

Query 2:

```
SELECT
    p.name AS PersonName,
    sum(c.shareprice * po.sharesowned) AS NetWorth
FROM
    companies c,
    people p,
    unnest(p.personowns) AS po
WHERE c.cid = po.cid AND po.sharesowned > 0
GROUP BY p.name;
```

Query 3:

```
SELECT
    c.name AS CompanyName,
    p.name AS PersonWithMostShares,
    MAX(po.sharesowned) AS SharesOwned
FROM
    companies c,
    people p,
    unnest(p.personowns) AS po
WHERE
    p.pid = ANY(c.boardheads) AND
    c.cid = po.cid AND
    po.sharesowned > 0
```

```
GROUP BY c.name, p.name;
```

Query 4:

```
/*
The first chunk, before the EXCEPT,
finds all companies that share an industry with each other.
The EXISTS ensures that each pair of companies occurs only once.
*/
SELECT
    comp1.name AS DominatingCompany, comp2.name AS DominatedCompany
FROM
    companies comp1,
    companies comp2
WHERE
    EXISTS (
        SELECT *
        FROM
            unnest(comp1.industry) i1,
            unnest(comp2.industry) i2
        WHERE
            comp1.cid != comp2.cid AND
            i1 = i2
    )
/*
Past the EXCEPT is a query that finds all pairs of companies
that has at least one person that is a board head of company 2 have
more shares in some company than any individual in company 1 has
of that same company.

The remaining result after applying EXCEPT is the final query
desired.
*/
EXCEPT
SELECT comp1.name, comp2.name
FROM
    companies comp1,
    companies comp2,
    people p2,
    unnest(p2.personowns) po2
WHERE
    comp1.cid != comp2.cid AND
    p2.pid = ANY(comp2.boardheads) AND
    po2.sharesowned > ALL (
        SELECT po1.sharesowned
```

```
        FROM
            people p1,
            unnest(p1.personowns) po1
        WHERE
            p1.pid = ANY(comp1.boardheads) AND
            po1.cid = po2.cid
    );
```

Query 5:

```
/*
This query is handled by a bunch of subqueries tied together.

The first query here returns the decimal value of
each person's direct control over a company.
*/
WITH RECURSIVE
    PersonDirectControl AS(
        SELECT
            p.pid AS pid,
            c.cid AS cid,
            (po.sharesowned::decimal / c.totalshares) AS
PerOfcidOwned
        FROM
            people p,
            unnest(p.personowns) po,
            companies c
        WHERE
            po.sharesowned > 0 AND
            c.cid = po.cid
    ),
/*
This next query finds the indirect control each person has over
each company. Notably, it differentiates between paths traced to
acquire the indirect control of a company, so one person could have
two or more differing indirect control values over the same company.
*/
    PersonIndirectControl (pid, cid1, cid2, PerOfcid1Owned,
PerOfcid2Owned, path) AS(
        SELECT
            PDC.pid,
            c1.cid,
            c2.cid,
            PDC.PerOfcidOwned,
            ((PDC.PerOfcidOwned * co1.sharesowned) / c2.totalshares),
```

```
            ARRAY[c2.cid]
        FROM
            PersonDirectControl PDC,
            companies c1,
            companies c2,
            unnest(c1.companyowns) co1
        WHERE
            c1.cid = PDC.cid AND
            c1.cid != c2.cid AND
            co1.sharesowned > 0 AND
            c2.cid = co1.cid
      UNION ALL
        SELECT
            PIC.pid,
            c2.cid,
            c3.cid,
            PIC.PerOfcid2Owned,
            ((PIC.PerOfcid2Owned * co2.sharesowned) /
c3.totalshares),
            path || c3.cid
        FROM
            PersonIndirectControl PIC,
            companies c2,
            companies c3,
            unnest(c2.companyowns) co2
        WHERE
            c2.cid = PIC.cid2 AND
            c2.cid != c3.cid AND
            co2.sharesowned > 0 AND
            c3.cid = co2.cid AND
            NOT (c3.cid = ANY(path))
    ),
/*
This query deals with the two or more differing results by adding
them all together as the total indirect control a person has over
a company.
*/
    PersonSumIndirectControl AS(
        SELECT
            p.pid AS pid,
            c.cid AS cid,
            sum(PIC.PerOfcid2Owned) AS ic
        FROM
            PersonIndirectControl PIC,
            companies c,
```

```
            people p
        WHERE
            c.cid = PIC.cid2 AND
            p.pid = PIC.pid
        GROUP BY p.pid, c.cid
    ),
/*
This query takes all person/company control pairs and add the direct
and indirect control values together, if such a value exists in both
tables.
*/
    PersonDirectAndIndirectControl AS(
        SELECT
            p.pid AS pid,
            c.cid AS cid,
            (PSIC.ic + PDC.PerOfcidOwned) AS dic
        FROM
            people p,
            companies c,
            PersonSumIndirectControl PSIC,
            PersonDirectControl PDC
        WHERE
            p.pid = PSIC.pid AND
            PSIC.pid = PDC.pid AND
            PSIC.cid = PDC.cid AND
            c.cid = PSIC.cid
    ),
/*
This query combines the direct, indirect, and direct+indirect tables
and lumps them into one big table.
*/
    PersonAllControl (pid, cid, percid) AS(
        SELECT * FROM PersonDirectControl
        UNION
        SELECT * FROM PersonSumIndirectControl
        UNION
        SELECT * FROM PersonDirectAndIndirectControl
    ),
/*
This query takes the union of the three tables above. For each
person/company pair, the value that is the largest between the three
tables is the final value kept in this new query.
*/
    PersonMaxControl AS(
        SELECT
```

```
            PAC.pid AS pid,
            PAC.cid AS cid,
            MAX(PAC.percid) AS percid
        FROM
            PersonAllControl PAC
        GROUP BY
            PAC.pid, PAC.cid
    ),
/*
This last query can now finally, after all of the above work, find
the people/company control values that are over 10%
*/
    Query5 AS (
        SELECT
            p.name AS PersonName,
            c.name AS CompanyName,
            TRUNC(PMC.percid * 100,4) AS Percentage
        FROM
            people p,
            companies c,
            PersonMaxControl PMC
        WHERE
            p.pid = PMC.pid AND
            c.cid = PMC.cid AND
            PMC.percid * 100 > 10
    )
SELECT * FROM Query5;
```

6. Web Application User Guide

You need the following:

- Postgresql server to host the database - complete with a user and a password ready to access the database.
- Netbeans - in order to host the server.
- Apache Tomcat - the server that will be used to host the webpages.
- JDBC Postgres Driver
- JSTL Taglibs file
- Ability to edit access permissions for files/directories, if necessary.

All necessary files can be found in the CSE532Project2Files directory.

Procedure:

1. Create a postgresql server. This server must have a user with a password that can access it - we will need this user's credentials later. The server name, user, and password used by this project is "CSE532" "postgres" "cse532" respectively, with no quotations.
2. Populate the server using the SQL commands provided in the "tableconstruct" file.
3. Install NetBeans.
4. Create the CSE532Project2 Java Project using the Ant format on Netbeans. Call the main file "QueryToList" and the package "com.cse532project2". The code necessary for the Java file is found in the "QueryToList.java" file. Make sure to also import the jdbc postgres driver .jar file into the library of this project as well.
5. Install Tomcat. Make sure to install the admin version of whichever version of Tomcat you are using as well. Set up the manager roles as necessary. You may need to edit permissions in various folders/files in order to achieve this.
6. Make a copy of the jdbc postgres driver .jar file. Insert this new copy in the CATALINA_HOME/lib directory so Tomcat has access to the driver.
7. Create Query1Web, Query2Web, Query3Web, Query4Web, Query5Web, and QueryMasterPage Web Projects in Netbeans, using the Tomcat server to host them. Again, you may need to change the permissions of various folders in order for Netbeans to properly read and write to various files/directories.
8. For Query1Web-Query5Web, make sure to replace the relevant context.xml in the web/META-INF folder and the relevant .jsp files for the default ones. Note that if you use a different server name, user, and/or password for the postgres database, you must reflect those changes in the context.xml file. In addition, make sure the jstl jar file and the CSE532Project2 project jar file is inserted into the libraries of each web project.
9. Start (or restart) the Tomcat server in Netbeans. Open your browser and type in http://localhost:8080/QueryToMaster/ to reach the index page, which will take you to any query as described by the project parameters.