CSE 532 – Theory of Database Systems

Project 2 - Report

Arun Rajan

110921170

Stony Brook University

**Pledge**

* *I pledge my honor that all parts of this project were done by me alone and without collaboration with anybody else.*

Entity – Relationship (ER) Diagram

cname

**cid**

sname

**sid**

sDate

Judge\_Table

Artwork\_table

Show\_table

oid

oid

CONTESTANT

anum

SCORES

jName

aname

Audition\_table

oid

jid

oid

**aid**

**Architecture**

Servelet

DBMS

Login.jsp

Success.jsp

**User**

Schema & Design Description

This document outlines some of the design decisions and the Schema used for the Project 2.

As shown in the ER diagram, there are 4 entities.

1. Show show\_table
2. Contestant contestant\_table
3. Artwork artwork\_table
4. Judge judge\_table

And one relationship table, audition\_table that has primary keys of all the tables mentioned above as foreign keys. Along, with it has a tuple of score type. Score type is explained later.

Please note that the above 4 tables are typed tables and a system generated Object Identifier (Oid) is associated with each record in these tables.

**Special Note** : If you perform “Select \* “ on such tables, you would not see the “oid” column as it is hidden.

To see it, use “Select oid, \* from XYZ”

Since a type table requires a type, below is the description:

1. CREATE TYPE show AS

(

sid TEXT,

sdate DATE,

sname TEXT

);

2. CREATE TYPE artwork AS

(

aid TEXT,

aname TEXT

);

3. CREATE TYPE contestant AS

(

cid TEXT,

cname TEXT

);

4. CREATE TYPE judge AS

(

jid TEXT,

jname TEXT

);

Once, we have type, we create typed tables as follows :

***Artwork\_table***

1. CREATE TABLE artwork\_table OF artwork(PRIMARY KEY(oid)) with OIDS;
2. INSERT INTO artwork\_table

VALUES ('a1','Barcarolle'),

('a2','Giselle'),

('a3','Bumblebee'),

('a4','Became Mucho'),

('a5','Swan Lake'),('a6', 'Habanera');

***Show\_table***

1. CREATE TABLE show\_table OF show(PRIMARY KEY(oid)) with OIDS;
2. INSERT INTO show\_table

VALUES ('s1','2014-02-02','show1'),

('s2','2014-04-02','show2'),

('s3','2014-06-05','show3'),

('s4','2014-08-02','show4'),

('s5','2014-06-05','show5');

***Contestant\_table***

1. CREATE TABLE contestant\_table OF contestant(PRIMARY KEY(oid)) with OIDS;
2. INSERT INTO contestant\_table

VALUES ('c1','Joe'),

('c2','Bob'),

('c3','Mary'),

('c4','Ann'),

('c5','Bess'),

('c6', 'Tom'),

('c7', 'Don');

***Judge\_table***

1. CREATE TABLE judge\_table OF judge(PRIMARY KEY(oid)) with OIDS;
2. INSERT INTO judge\_table

VALUES ('j1','Judy’),

(‘j2','Lucy'),

('j3','Irving'),

('j4','Phil'),

('j5','Oscar');

***Audition\_table***

The Audition table has the following schema :

1. CREATE TABLE audition\_table

(

anum INTEGER,

show\_id INTEGER REFERENCES show\_table(oid),

cont\_id INTEGER REFERENCES contestant\_table(oid),

art\_id INTEGER REFERENCES artwork\_table(oid),

scores score[],

PRIMARY KEY(anum, show\_id, cont\_id, art\_id)

);

**Insert Values :**

INSERT INTO audition\_table VALUES

(1, 16624, 16609, 16595, ‘{ROW(16582,7),ROW(16583, 8),ROW(16584, 6)}’);

And so on….!!!

**NOTE:**

It is worth mentioning that having OIDs as a foreign key in the audition\_Table did not help much. As the system generates it as INT 32. This makes us to treat them like any other integer value except that they cannot change.

It would have made things really easy had we had references instead of integers.

**Integrity Constraints, Referential Integrity Constraints and CHECK-constraints**

Each typed table has oid as its primary key. The snippets below explain it further.

For example, “\d+ artwork\_table”

Indexes:

"artwork\_table\_pkey" PRIMARY KEY, btree (oid)

Referenced by:

TABLE "audition\_table" CONSTRAINT "audition\_table\_art\_id\_fkey" FOREIGN KEY (art\_id) REFERENCES artwork\_table(oid)

Typed table of type: artwork

Has OIDs: yes

“\d+ show\_table

Indexes:

"show\_table\_pkey" PRIMARY KEY, btree (oid)

Referenced by:

TABLE "audition\_table" CONSTRAINT "audition\_table\_show\_id\_fkey" FOREIGN KEY (show\_id) REFERENCES show\_table(oid)

Typed table of type: show

Has OIDs: yes

And finally, If we give the following command “\d+ audition\_table”

We can see something like :

Indexes:

"audition\_table\_pkey" PRIMARY KEY, btree (anum, show\_id, cont\_id, art\_id)

"audition\_table\_anum\_key" UNIQUE CONSTRAINT, btree (anum)

Foreign-key constraints:

"audition\_table\_art\_id\_fkey" FOREIGN KEY (art\_id) REFERENCES artwork\_table(oid)

"audition\_table\_cont\_id\_fkey" FOREIGN KEY (cont\_id) REFERENCES contestant\_table(oid)

"audition\_table\_show\_id\_fkey" FOREIGN KEY (show\_id) REFERENCES show\_table(oid)

**Project Queries**

1. **Find all pairs of contestants who auditioned the same artwork on the same date and got the same score from at least one judge (not necessarily the same judge).**
2. First we create a view, that extracts score from scores tuple for each record in audition\_table

*CREATE TEMPORARY VIEW* **judge\_score** *AS*(

*SELECT* anum,

*ARRAY*(SELECT (s).score FROM UNNEST(scores) s) arr

*FROM* audition\_table

);

1. Now use this query to get the results

SELECT DISTINCT A.cname, B.cname

FROM contestant\_table as A, contestant\_table as B,

audition\_table as C, audition\_table as D,

judge\_score as E,judge\_score F,

show\_table G, show\_Table H

WHERE C.cont\_id <> D.cont\_id

AND C.show\_id = G.oid AND D.show\_id = H.oid

AND G.sdate = H.sdate AND D.art\_id = C.art\_id

AND C.cont\_id = A.oid AND D.cont\_id = B.oid

AND E.arr && F.arr AND E.anum = C.anum

AND F.anum = D.anum AND A.cname < B.cname;

1. **Find all pairs of contestants who happened to audition the same artwork (in possibly different shows) and got the same maximal score and the same minimal score for that audition (from possibly different judges).**
2. CREATE TEMPORARY VIEW maxmin as

( SELECT show\_id,

cont\_id,

art\_id,

(SELECT MAX((s).score) from unnest(scores) s) as maxscore,

(SELECT MIN((s).score) from unnest(scores) s) as minscore

FROM audition\_table

);

b)

SELECT A.cname, B.cname

FROM contestant\_table as A, contestant\_table as B,

maxmin as C, maxmin as D

WHERE A.oid = C.cont\_id AND B.oid = D.cont\_id

AND A.cname<B.cname AND C.maxscore = D.maxscore

AND C.minscore = D.minscore

AND C.art\_id = D.art\_id order by A.cname;

1. **Find all pairs of contestants who auditioned the same artwork in (possibly different) shows that had the same number of judges and the two contestants received the same average score for that audition.**

**This query also involves aggregates.**

a)

CREATE TEMPORARY VIEW Q3\_aux as

(SELECT show\_id,

cont\_id,

art\_id,

(select count((s).score) from unnest(scores) s) as noj,

(Select AVG((s).score) from unnest(scores) s) as avgscore

FROM audition\_table);

b)

SELECT A.cname, B.cname

FROM contestant\_table as A, contestant\_table as B,

Q3\_aux as C, Q3\_aux as D

WHERE C.avgscore = D.avgscore AND C.noj = D.noj

AND C.art\_id = D.art\_id AND A.cname < B.cname

AND C.cont\_id <> D.cont\_id AND C.cont\_id = A.oid

AND D.cont\_id = B.oid order by A.cname, B.cname;

1. **Find all pairs of contestants (by name) such that the first contestant in each pair performed in all the shows in which the second contestant did (possibly performing different artworks).**
2. For each contestant, first, we group all the shows he/she performed

CREATE TEMPORARY VIEW Q4\_aux as

(

SELECT cont\_id, show\_id

FROM audition\_table

GROUP BY cont\_id,show\_id

);

1. Now, we use the result of part A, to calculate a view such that it has two entries.

first, the contestants and second, an array of all the shows he/she atteneded.

CREATE TEMPORARY VIEW q4\_aux\_1 as

(

SELECT cont\_id, array\_agg(show\_id) allshows

FROM q4\_aux group by cont\_id

);

C) Now the problem remains just of checking the membership of one contestant's show list into other's

SELECT A.cname, B.cname

FROM contestant\_table as A, contestant\_table as B,

q4\_aux\_1 as C, q4\_aux\_1 as D

WHERE A.oid = C.cont\_id AND B.oid = D.cont\_id

AND C.cont\_id <> D.cont\_id

AND ((C.allshows @> D.allshows) OR (C.allshows = D.allshows))

ORDER by A.cname, B.cname;

5.

**Find all close rivals. The close rivals relation is the transitive closure of the following binary relation: X and Y are direct close rivals iff they both performed the same artwork in the same show and their overall average scores are within 0.2 of each other.**

1. First, for each contestant, find his/her overall average score: **Q5\_avg VIEW**

CREATE TEMPORARY VIEW q5\_avg as(

SELECT cont\_id, sum(sumscores)/sum(count1) avgscore

FROM (SELECT cont\_id, (select count((s).score) FROM unnest(scores) s) as count1,

(SELECT sum((s).score) FROM unnest(scores) s) as sumscores

FROM audition\_table) as Z group by z.cont\_id

);

1. Take a join with audition table and get corresponding 'showobj', 'artobj' for each contestant

CREATE TEMPORARY VIEW q5\_aux as(

SELECT A.cont\_id, A.show\_id, A.art\_id, B.avgscore

FROM audition\_Table as A,

q5\_avg as B

WHERE A.cont\_id = B.cont\_id

);

1. Find the binary relation (X, Y), as an intermediate step, such that X and Y performed in the same artwork, in the same show and their over-all average scores are within 0.2 of each other.

CREATE TEMPORARY VIEW Q5\_binary as (

SELECT DISTINCT A.cname X, B.cname Y

FROM contestant\_table as A, contestant\_table as B,

Q5\_aux as C, Q5\_aux as D

WHERE A.oid = C.cont\_id AND B.oid = D.cont\_id

AND C.art\_id = D.art\_id AND C.show\_id = D.show\_id

AND C.cont\_id <> D.cont\_id

AND @(C.avgscore - D.avgscore) <= 0.2);

1. Finally using a recursive query we find the transitive closure.

that is if xRy, yRz ---> xRz

WITH RECURSIVE Q5\_3(X, Y) AS

(

SELECT X, Y from q5\_binary

UNION

SELECT A.X, B.Y FROM q5\_binary as A, Q5\_3 as B

WHERE A.Y = B.X)

SELECT \* FROM Q5\_3 where X<Y order by X, Y;

**Guide to run the program**

It is really easy to run and see the demo for this project.

Steps:

1. Import the project in the IDE Eclipse.
2. Make sure the dependencies are there like JSTL 1.2 jar, PSQL JDBC jar etc
3. Turn the Psql Server on.
4. Click on the title of the Project and run on Server. It would start the Tom Cat server

On port 8080

Hit the URL http://localhost:8080/DBMS\_P2/