CSE 532 – Theory of Database Systems

# PROJECT 2 - REPORT

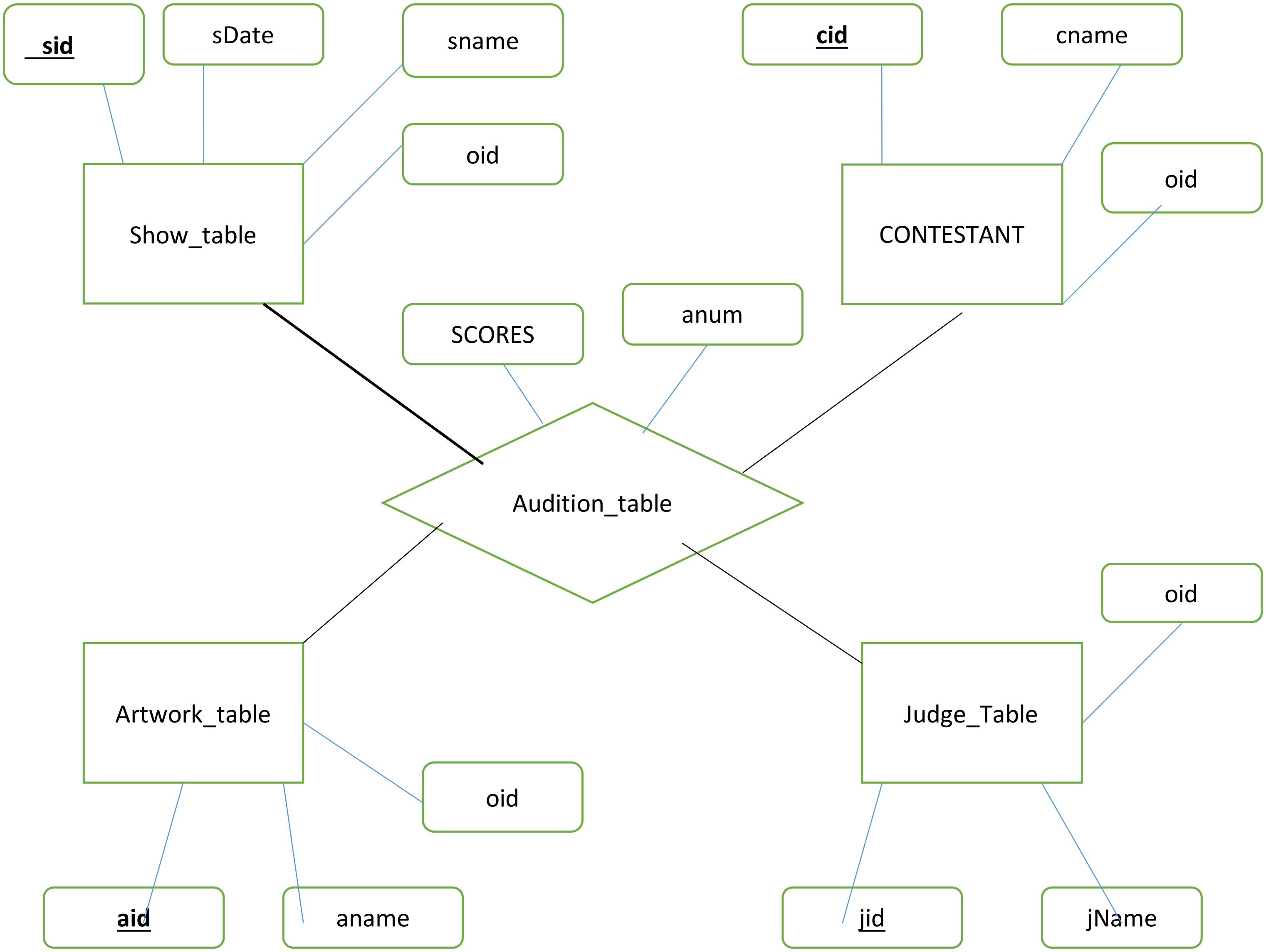
Arun Rajan

110921170
Stony Brook University

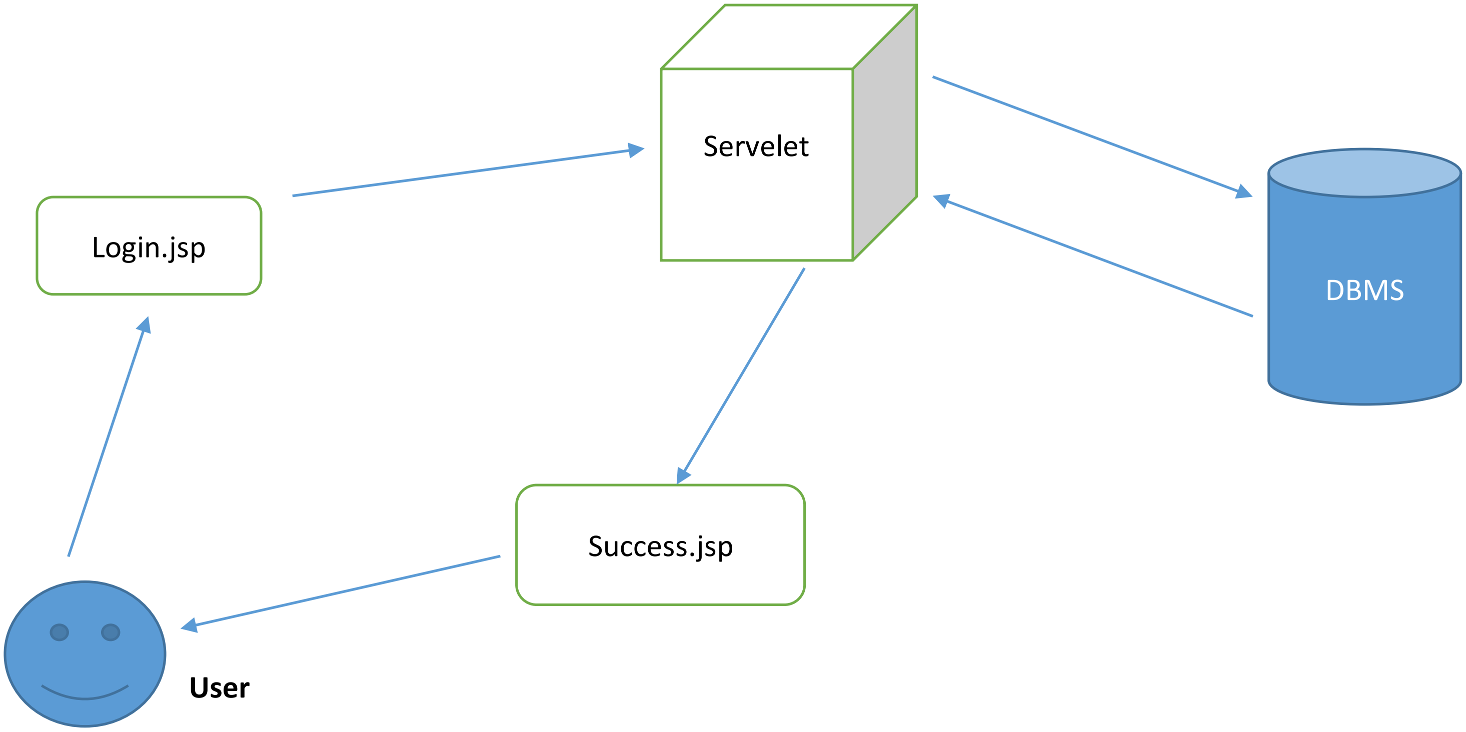
# Pledge

. *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



# Architecture



# 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).**

a) 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
 );
```

b) 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;
```

2. **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).**

a)
```
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;
```

3. **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;
```

4. **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).**

a) 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
);
```

b) 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.**

a) 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
 );
```

b) 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
 );
```

c) 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);
```

d) 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/