California University of PA

Zachary Smith

Strike-A-Spark Application

Spring 2019 Semester Project

CSC/CIS 322 Database Application Development

Zachary Smith

California University of PA

CSC/CIS 322 Database Application Development

Spring Semester Project

Table of Contents

PAGE

Database Individual Assignment 1: Database Description

For this project I decided to up the ante by designing a database for the Center of Undergraduate Research at California University of Pennsylvania. Every year the center hosts the Strike-A-Spark conference in which students from across the university display their research through one-of-three mediums, a written paper, an oral presentation, or a poster presentation. Regarding the latter most category there is an additional component whereby a presenter can opt into having their poster evaluated by a group of judges. After several meetings and a few e-mails with the Chair of the center, I was able to take their vision and construct a meaningful model that would satisfy their needs.

The management systems used for this project were Oracle and MySQL; the reason being was that the university for their business-end servers use Oracle while MySQL is part of their academic tools to teach students how to design databases. However, regardless of the management system used both have the same general schema and table design. For this project there was five tables created: judge, poster, presenter, scores, and owns. The tables judge, poster, and presenter are the objects of interest while the latter two tables are the relationships. The scores table is a many-to-many relationship between the judge table and the poster table, since every judge can score several posters and every poster can be scored by several judges. The owns table is the many-to-many relationship between the posters and presenters, since a poster can be a collaboration of many presenters and a presenter can present several posters. Fortunately for me the scores and presenter tables would store the most significant information and their fields were mostly predetermined as the evaluation criterion and registration criterion respectively.

Database Individual Project 1: DDL

```
create table judge
(    judge_ID int auto_increment
,    judge_name varchar(50)
,    role varchar(50)
,    primary key (judge_ID, judge_name)
);
```

Since this database is used for an application where judges score different kinds of presentations, this table is used to keep track of the individual judges. The primary key for this table is the combination of judge_ID and judge_name where I figured judge_ID values would eventually get recycled on a yearly basis as that is the frequency of the conference. Furthermore, the role field assigns whether the judge is strictly a judge or has additional privileges too, e.g. setting passwords.

```
create table scores
(    judge_ID int
,    judge_name varchar(50)
,    poster_ID int
,    visual int
,    clarity int
,    thoroughness int
,    breadth int
,    depth int
,    quality int
,    discussion int
,    understanding int
,    overall int
,    comment varchar(150)
,    constraint FK_JudgeScores
     foreign key (judge_ID, judge_name) references judge (judge_ID, judge_name)
,    constraint FK_PosterScores
     foreign key (poster_ID) references poster (poster_ID)
);
```

The scores table is the heart and soul of this database. As mentioned above we need a table that can store the many criterion in which the presentations will be evaluated on. Although this table does not have a primary key it does, however, have two foreign keys FK_JudgeScores and FK_PosterScores, which establishes the relationship between judges and posters via the scores given by judges to the posters.

```
create table poster
(    poster_ID int
,    title varchar(50)
,    submission_type varchar(1)
,    primary key (poster_ID)
);
```

In retrospect the name for this table needs alter. Initially the clients only wanted to store the posters that would be submitted in the competition. However, they eventually realized they could store all presentations, written, oral, or poster, which we could make a new field called submission type. Moreover, the primary key ought to be modified to be a combination of poster_ID and submission_type because each submission_type can have their own integer values to keep track, e.g. p12, w12, o12, there could be twelve presentations per category with a combinational primary key this would not create a conflict within the database.

```
create table owns
(    poster_ID int
,    presenter_ID int
,    presenter_name varchar(50)
,    constraint FK_PosterOwns
     foreign key (poster_ID) references poster (poster_ID)
,    constraint FK_PresenterOwns
     foreign key (presenter_ID, presenter_name)
     references presenter (presenter_ID, presenter_name)
);
```

The owns table is another relationship table between presenters and posters since each poster can have multiple presenters and a presenter can have multiple presentations. Hence, we have the constraints representing the foreign keys FK_PosterOwns and FK_PresenterOwns respectively.

```
create table presenter
(    presenter_ID int auto_increment
,    presenter_name varchar(50)
,    title varchar(50)
,    department varchar(50)
,    category varchar(50)
,    academic_level varchar(50)
,    class_level varchar(50)
,    area_of_study varchar(50)
,    sponsor varchar(50)
,    primary key (presenter_ID, presenter_name)
);
```

Finally, we have the presenter table. The primary key is again a combination of two fields, the name of the presenter and the presenter_ID, which is auto_incremented to prevent the issue of Jane Doe entering twice. Furthermore, only a few of these fields are relevant for the queries we will implement for the application, namely, area_of_study and department. Area_of_study includes multi-disciplinary, science technology business, liberal arts, education health science, then the department field would be the individual majors belonging to each area of study.

Database Individual Project 1: DML

/* Judge table */

```
insert into judge
values (001, "Leeroy Jenkins"),
(007, "James Bond"),
(002, "Spongebob Squarepants"),
(003, "Eric Aundrey"),
(004, "John Doe");
```

1	Leeroy Jenkins
2	Spongebob Squarepants
3	Eric Aundrey
4	John Doe
7	James Bond

/* Poster Table */

```
insert into poster
values (1, "Investigations into Fibonacci Inequalities", "p"),
(2, "tractatus logico-philosophicus", "p"),
(3, "The Principles of Mathematics", "p"),
(4, "Principia Mathematica", "p"),
(5, "Esquisse d'un programme", "p");
```

1	Investigations into Fibonacci Inequalities	p
2	tractatus logico-philosophicus	p
3	The Principles of Mathematics	p
4	Principia Mathematica	p
5	Esquisse d'un programme	p

/* Presenter Table */

```
insert into presenter
(presenter_name, department, category,
 academic_level, class_level, area_of_study, sponsor)
Values
("Kyle One", "Biology and Environmental Science", "Class project", "Undergraduate", "Senior",
"Liberal Arts", "Dr. Rebecca Regeth"),
```

("Kyle Two", "Communication Disorders", "Class project", "Graduate", "Graduate", "Education, Health Sciences", "Dr. BeesKnees"),
("Kyle Three", "Mathematics", "Individual project", "Undergraduate", "Freshmen", "Science Technology Business", "Dr. PeanutButter"),
("Kyle Four", "Biology", "Individual project", "Undergraduate", "Junior", "Liberal Arts", "Dr. Soandso"),
("Kyle Five", "Chemistry", "Class project", "Graduate", "Graduate", "Science Technology Business", "Dr. Potato");

1 Kyle One Biology and Environmental Science Class project
Undergraduate Senior Liberal Arts Dr. Rebecca Regeth
2 Kyle Two Communication Disorders Class project
Graduate Graduate Education, Health Sciences Dr. BeesKnees
3 Kyle Three Mathematics Individual project
Undergraduate Freshmen Science Technology Business Dr. PeanutButter
4 Kyle Four Biology Individual project
Undergraduate Junior Liberal Arts Dr. Soandso
5 Kyle Five Chemistry Class project
Graduate Graduate Science Technology Business Dr. Potato

/* Owns Table */

```
insert into owns
 values (1, 1, "Kyle One"),
 (2, 2, "Kyle Two"),
 (3, 3, "Kyle Three"),
 (4, 4, "Kyle Four"),
 (5, 5, "Kyle Five");
```

1 1 Kyle One
2 2 Kyle Two
3 3 Kyle Three
4 4 Kyle Four
5 5 Kyle Five

/* Scores Table */

```
insert into scores
values
(1, "Leeroy Jenkins", 1, 1, 2, 3, 4, 3, 2, 1, 1, 10),
(1, "Leeroy Jenkins", 2, 1, 1, 2, 2, 3, 3, 2, 2, 1),
(2, "Spongebob Squarepants", 1, 2, 3, 1, 2, 3, 1, 1, 3, 2),
(2, "Spongebob Squarepants", 3, 1, 1, 1, 1, 1, 1, 1, 1, 1),
(3, "Eric Aundrey", 1, 1, 1, 1, 2, 3, 2, 2, 3, 1),
```

```
(3, "Eric Aundrey", 2, 1, 2, 3, 2, 2, 3, 1, 2, 5),
(3, "Eric Aundrey", 3, 2, 2, 2, 3, 3, 3, 1, 1, 6),
(3, "Eric Aundrey", 4, 1, 1, 2, 3, 3, 1, 2, 2, 5),
(3, "Eric Aundrey", 5, 3, 3, 3, 2, 1, 2, 3, 2, 10),
(4, "John Doe", 1, 1, 2, 3, 4, 2, 1, 2, 1, 3),
(4, "John Doe", 2, 2, 3, 4, 2, 2, 3, 1, 1, 7),
(4, "John Doe", 3, 2, 2, 1, 3, 4, 2, 2, 2, 4),
(4, "John Doe", 4, 2, 1, 1, 3, 2, 3, 2, 3, 2),
(4, "John Doe", 5, 1, 1, 1, 1, 1, 1, 1, 1, 1),
(7, "James Bond", 1, 5, 5, 5, 5, 5, 3, 2, 5, 8),
(7, "James Bond", 2, 2, 3, 4, 2, 3, 4, 2, 3, 4),
(7, "James Bond", 3, 4, 5, 3, 4, 5, 3, 4, 5, 7),
(7, "James Bond", 4, 5, 6, 4, 5, 6, 7, 5, 6, 7),
(7, "James Bond", 5, 2, 3, 4, 2, 4, 3, 5, 6, 10);
```

```
1  Leeroy Jenkins  1  1  2  3  4  3  2  1  1  10
1  Leeroy Jenkins  2  1  1  2  2  3  3  2  2  1
2  Spongebob Squarepants  1  2  3  1  2  3  1  1  3  2
2  Spongebob Squarepants  3  1  1  1  1  1  1  1  1  1
3  Eric Aundrey  1  1  1  1  2  3  2  2  3  1
3  Eric Aundrey  2  1  2  3  2  2  3  1  2  5
3  Eric Aundrey  3  2  2  2  3  3  3  1  1  6
3  Eric Aundrey  4  1  1  2  3  3  1  2  2  5
3  Eric Aundrey  5  3  3  3  2  1  2  3  2  10
4  John Doe  1  1  2  3  4  2  1  2  1  3
4  John Doe  2  2  3  4  2  2  3  1  1  7
4  John Doe  3  2  2  1  3  4  2  2  2  4
4  John Doe  4  2  1  1  3  2  3  2  3  2
4  John Doe  5  1  1  1  1  1  1  1  1  1
7  James Bond  1  5  5  5  5  5  3  2  5  8
7  James Bond  2  2  3  4  2  3  4  2  3  4
7  James Bond  3  4  5  3  4  5  3  4  5  7
7  James Bond  4  5  6  4  5  6  7  5  6  7
7  James Bond  5  2  3  4  2  4  3  5  6  10
```

Database Individual Project II: Queries

```
/* Query 1 */

SELECT a.presenter_name, a.poster_title, SUM(a.rec_tot) AS total
FROM
(SELECT
   presenter.presenter_name AS presenter_name,
   poster.title AS poster_title,
   (visual + clarity + thoroughness
    + breadth + depth + quality
    + discussion + understanding + overall)
   AS rec_tot
   FROM scores
   INNER JOIN judge ON judge.judge_id = scores.judge_ID
   INNER JOIN poster ON poster.poster_ID = scores.poster_ID
   INNER JOIN owns ON poster.poster_ID = owns.poster_ID
   INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
   ORDER BY presenter.academic_level, presenter.area_of_study, presenter.category
) AS a
GROUP BY a.presenter_name;
```

	presenter_name	poster_title	total
▶	Kyle One	Investigations into Fibonacci Inequalities	123
	Kyle Two	tractatus logico-philosophicus	90
	Kyle Three	The Principles of Mathematics	94
	Kyle Four	Principia Mathematica	90
	Kyle Five	Esquisse d'un programme	77

Query 1 as stated above is the essential output as requested by the client, which is to produce the following: we are collecting the judge's scores of a poster and they will be stored as an individual record. Furthermore, how it works is the subquery inside the FROM clause generates a temporary table that takes the name 'a,' said table is the sum of every record stratified by the presenter's academic level, graduate or undergraduate; area of study the mathematics, education, etc; and the submission category Liberal Arts, Science Business & Technology, etc. Moreover, instead of displaying every individual sum of every record I decided a grand-total would clearly display the presenter with the greatest total points for every

condition. This grand-total is processed in the parent query by passing the values for every record-total with the presenter's name associated to it.

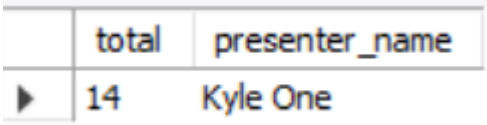
```
/* Query 2 */

SELECT a.presenter_name, AVG(a.understanding) AS average
FROM
(SELECT
   presenter.presenter_name AS presenter_name,
   scores.understanding AS understanding,
   (visual + clarity + thoroughness
    + breadth + depth + quality
    + discussion + understanding + overall)
   AS rec_tot
   FROM scores
   INNER JOIN judge ON judge.judge_id = scores.judge_ID
   INNER JOIN poster ON poster.poster_ID = scores.poster_ID
   INNER JOIN owns ON poster.poster_ID = owns.poster_ID
   INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
   ORDER BY presenter.academic_level, presenter.area_of_study, presenter.category
) AS a
GROUP BY a.presenter_name;
```

presenter_name	average
Kyle One	2.6000
Kyle Two	2.0000
Kyle Three	2.2500
Kyle Four	3.6667
Kyle Five	3.0000

Query 2 checks the average of the field "understanding" for every participant. The idea being understanding is supposed to represent the overall knowledge on the topic they are presenting on a scale from 0-5. The resulting averages seemed to hover around three, which I think is an indication that the presenters knew the topic well enough to participate like they had but lack substantial knowledge of their topics.

```
/* Query 3 */

SELECT COUNT(*) AS total, presenter.presenter_name
```

```
FROM scores
INNER JOIN judge ON judge.judge_id = scores.judge_ID
INNER JOIN poster ON poster.poster_ID = scores.poster_ID
INNER JOIN owns ON poster.poster_ID = owns.poster_ID
INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
WHERE scores.breadth = 1 OR scores.clarity = 1 OR scores.depth = 1
OR scores.discussion = 1 OR scores.overall = 1 OR scores.quality = 1
OR scores.thoroughness = 1 OR scores.understanding = 1 OR scores.visual = 1;
```



	total	presenter_name
▶	14	Kyle One

Query 3 takes a count of all records in which any field value within the scores table has a value of 1. This would signify that of any attribute a presentation is being judged on the student has failed in that regard. Moreover, there is only a single presenter that does not fail.

```
/* Query 4 */

SELECT a.judge_name, SUM(a.rec_tot) AS total
FROM
(SELECT
   presenter.presenter_name AS presenter_name,
   scores.overall AS overall,
   scores.understanding AS understanding,
   judge.judge_name AS judge_name,
   (visual + clarity + thoroughness
    + breadth + depth + quality
    + discussion + understanding + overall)
   AS rec_tot
   FROM scores
   INNER JOIN judge ON judge.judge_id = scores.judge_ID
   INNER JOIN poster ON poster.poster_ID = scores.poster_ID
   INNER JOIN owns ON poster.poster_ID = owns.poster_ID
   INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
   ) AS a
GROUP BY a.judge_name
ORDER BY total DESC;
```

judge_name	total
James Bond	200
Eric Aundrey	109
John Doe	94
Leeroy Jenkins	44
Spongebob Squarepants	27

The query above displays the total points given by each judge in descending order with judge James Bond awarding the most points while judge Spongebob Squarepants. Although the range is from 200 to 27 none of which out statistical outliers. In short this simply means Spongebob is a frugal judge.

```
/* Query 5 */

SELECT
judge.judge_name,
(visual + clarity + thoroughness
+ breadth + depth + quality
+ discussion + understanding + overall)
AS rec_tot
FROM scores
INNER JOIN judge ON judge.judge_id = scores.judge_ID
INNER JOIN poster ON poster.poster_ID = scores.poster_ID
INNER JOIN owns ON poster.poster_ID = owns.poster_ID
INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
ORDER BY judge_name DESC;
```

judge_name	rec_tot
Spongebob Squarepants	18
Spongebob Squarepants	9
Leeroy Jenkins	27
Leeroy Jenkins	17
John Doe	19
John Doe	25
John Doe	22
John Doe	19
John Doe	9
James Bond	43
James Bond	27
James Bond	40
James Bond	51
James Bond	39
Eric Aundrey	16
Eric Aundrey	21
Eric Aundrey	23
Eric Aundrey	20
Eric Aundrey	29

Between query 4 and query 5 we can break down the judges' sum; from this we see the range from the previous query might lead us to believe there is an outlier. But once we break down the query 4 into the query 5, we see that an outlier is truly not present.

```
/* Query 6 */

SELECT
presenter.presenter_name, judge.judge_name,
        (visual + clarity + thoroughness + breadth +
 depth + quality + discussion + understanding + overall) as total
from scores
INNER JOIN judge ON judge.judge_id = scores.judge_ID
INNER JOIN poster ON poster.poster_ID = scores.poster_ID
```

```
INNER JOIN owns ON poster.poster_ID = owns.poster_ID
INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID;
```

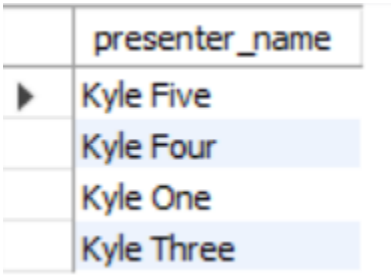
presenter_name	judge_name	total
Kyle One	Leeroy Jenkins	27
Kyle One	Spongebob Squarepants	18
Kyle One	Eric Aundrey	16
Kyle One	John Doe	19
Kyle One	James Bond	43
Kyle Two	Leeroy Jenkins	17
Kyle Two	Eric Aundrey	21
Kyle Two	John Doe	25
Kyle Two	James Bond	27
Kyle Three	Spongebob Squarepants	9
Kyle Thre	Aundrey	23
Kyle Three	John Doe	22
Kyle Three	James Bond	40
Kyle Four	Eric Aundrey	20
Kyle Four	John Doe	19
Kyle Four	James Bond	51
Kyle Five	Eric Aundrey	29
Kyle Five	John Doe	9
Kyle Five	James Bond	39

Kyle Three

Query 6 displays three fields, the presenters name, the judge name, and the sum of their overall evaluation by the judge. Such a query I think could help give the presenter a chance to see if there was a judge in their field then they could ask for guidance on how to work better from someone within their industry.

```
/* Query 7 */

SELECT presenter.presenter_name
FROM scores
INNER JOIN judge ON judge.judge_id = scores.judge_ID
INNER JOIN poster ON poster.poster_ID = scores.poster_ID
```

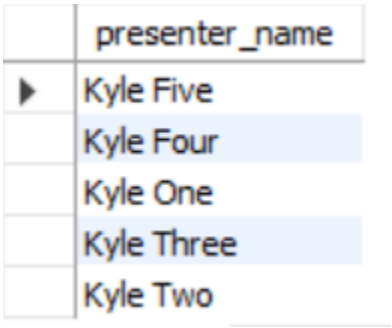
```
INNER JOIN owns ON poster.poster_ID = owns.poster_ID
INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
WHERE scores.visual >= 3
GROUP BY presenter.presenter_name
ORDER BY presenter.presenter_name;
```



presenter_name
Kyle Five
Kyle Four
Kyle One
Kyle Three

Query 7 displays if any student received at least an average score on their visual presentation. The significance being that often presenting raw data or without thought behind the visual aspect can have inverse effects with one's interlocutor. Between statistical consulting and the employer often they won't be familiar with the jargon of your discipline, however, presenting everything in an easy to digest way can provide insight that a business would be interested in.

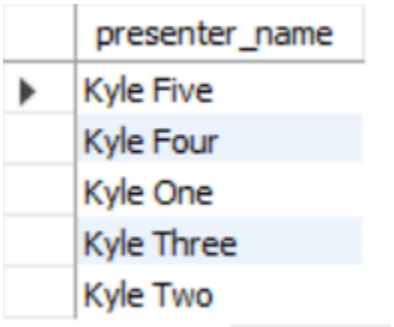
```
/* Query 8 */

SELECT presenter.presenter_name
FROM scores
INNER JOIN judge ON judge.judge_id = scores.judge_ID
INNER JOIN poster ON poster.poster_ID = scores.poster_ID
INNER JOIN owns ON poster.poster_ID = owns.poster_ID
INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
WHERE scores.understanding >= 3
GROUP BY presenter.presenter_name
ORDER BY presenter.presenter_name;
```



presenter_name
Kyle Five
Kyle Four
Kyle One
Kyle Three
Kyle Two

Query 8 describes if a presenter has at least minimally demonstrated an understanding of how to conduct research. For instance, students can apply for a grant from the Center of Undergraduate Research. In doing so a student and their sponsor must come up with a weekly schedule that outlines how they will manage their time. If a student were to have gone through this process, they should clearly state it, or it would be evident in their presentation mannerisms.

```
/* Query 9 */

SELECT presenter.presenter_name
FROM scores
INNER JOIN judge ON judge.judge_id = scores.judge_ID
INNER JOIN poster ON poster.poster_ID = scores.poster_ID
INNER JOIN owns ON poster.poster_ID = owns.poster_ID
INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
WHERE scores.depth > 3
GROUP BY presenter.presenter_name
ORDER BY presenter.presenter_name;
```

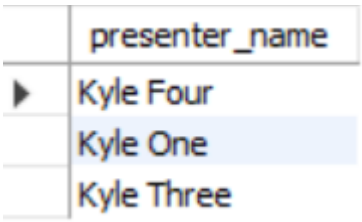


presenter_name
Kyle Five
Kyle Four
Kyle One
Kyle Three
Kyle Two

Query 9 checks for a greater than the average score in depth. This metric, as well as breadth, which we will see shortly are at odds with one another and as such give a great insight into how the presenter decided to organize their topic for research. Depth considers rather or not the research delves deeply into a niche without mentioning similar topics.

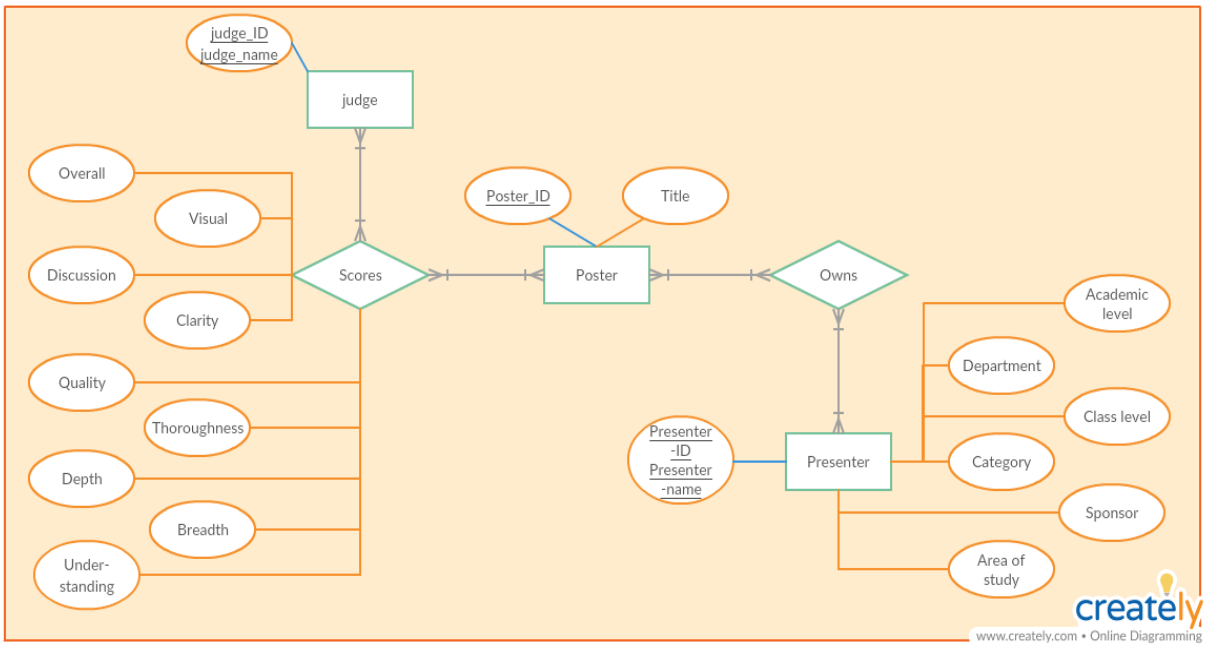
```
/* Query 10 */

SELECT presenter.presenter_name
FROM scores
INNER JOIN judge ON judge.judge_id = scores.judge_ID
INNER JOIN poster ON poster.poster_ID = scores.poster_ID
INNER JOIN owns ON poster.poster_ID = owns.poster_ID
```

```
INNER JOIN presenter ON presenter.presenter_ID = owns.presenter_ID
WHERE scores.breadth > 3
GROUP BY presenter.presenter_name
ORDER BY presenter.presenter_name;
```



presenter_name
Kyle Four
Kyle One
Kyle Three

Finally query 10, as stated in the last description, query 9, the concept of depth and breadth are at odds with one another. In this attribute instead of being concerned with how far a presentation goes within its field of study, this score is trying to represent how it ties into related material. This of course would present its self in the poster as a “big picture” perspective.


judge_ID
judge_name
judge
Overall
Visual
Discussion
Clarity
Scores
Poster_ID
Title
Poster
Owns
Quality
Thoroughness
Depth
Breadth
Under-
standing
Presenter
-ID
Presenter
-name
Presenter
Academic
level
Department
Class level
Category
Sponsor
Area of
study
creately
www.creately.com • Online Diagramming