2019

# Big Teeth Reality TV - DBMS

RELATIONAL DATABASE PROJECT – GROUP 12

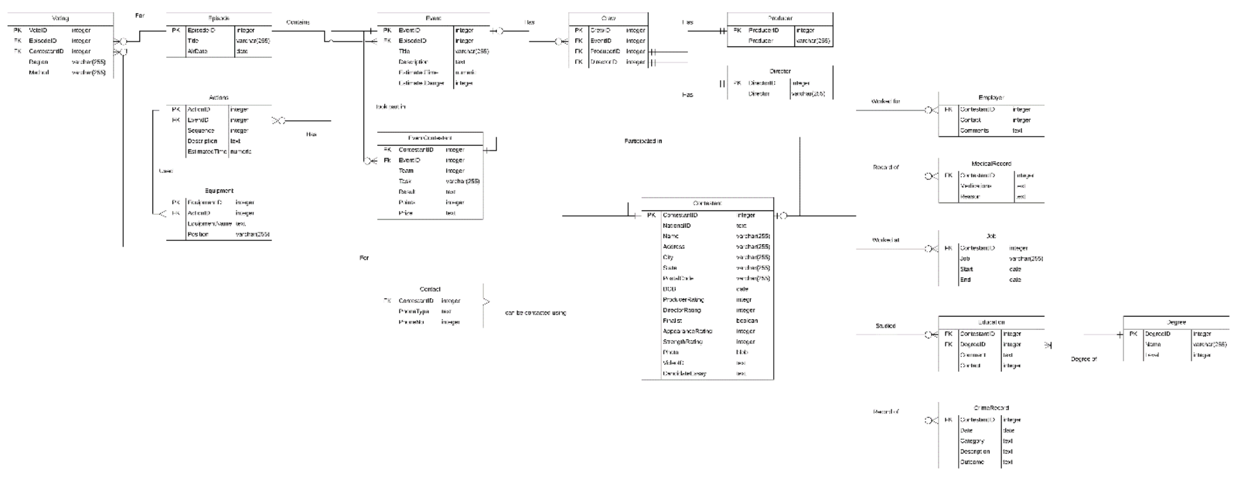
TEAM: SQL PIRATES

MEMBERS: ASAD MAHMOOD (989355987), KHUSHBOO PATEL (989354961), THEEWART JINDAPOO (989354797) AND ATIK ILMAN HOSSAIN (989354898)

# Table of Contents

## DBMS ERD

Big Teeth Reality TV
Entity Relationship Diagram



### ERD Relationship Explanation

In this subsection we will explain our ERD and the relationships between different tables of the database and different keys that are used to link them through these relationships. This ERD was created using an online tool called Lucid chart.

NOTE: Since it is not clearly visible here inside the report so we have attached it as a standalone picture along with this report.

### Column Names

There are a total of 17 columns in our database. Their names are as follows:

1. Contestant
2. Employer
3. MedicalRecord
4. Job
5. Education
6. Degree
7. CrimeRecord
8. Contact
9. Crew
10. Producer
11. Director
12. Event
13. EventContestant
14. Episode
15. Actions
16. Equipment
17. Voting

## Relationships explanation between columns

| Table1 ~ Table2 | Type of Relation | Explanation |
|---|---|---|
| Episode ~ Voting | One to Zero/Many | This relationship shows that mandatory one episode can have zero to many votes. |
| Episode ~ Event | One to One/Many | It shows that mandatory one episode contains one to many events in it. |
| Event ~ Actions | One to Zero/Many | It shows that mandatory one event can have zero to many actions involved. |
| Event ~ Crew | Zero/One to Zero/Many | It shows that in at most one event there can be zero to many crew involved. |
| Event ~ EventContestant | One to Zero/Many | It shows that mandatory one event can have zero to many contestants that took part in the event. |
| Action ~ Equipment | One to One/Many | This relationship shows that mandatory one action uses one or many equipments. |
| Contestant ~ EventContestant | Zero/One to Zero/Many | The relationship shows that zero or one contestant could have participated in zero or many events in the reality tv competition. |
| Contestant ~ Contact | One to One/Many | The relationship shows that mandatory one contestant can have mandatory one or many contact numbers through which he or she can be contacted with. |
| Contestant ~ Voting | One to Zero/Many | The relationship shows that mandatory one contestant can have zero or many votes for his/her performance in the episode. |
| Contestant ~ Employer | Zero/One to Zero/Many | The relationship shows that zero or one contestant can have zero or many employers. |
| Contestant ~ MedicalRecord | Zero/One to Zero/Many | The relationship shows that zero or one contestant can have zero or many medications prescribed to him. |
| Contestant ~ Job | Zero/One to Zero/Many | The relationship shows that zero or one contestant can have held zero or many Jobs in his life |
| Contestant ~ Education | Zero/One to Zero/Many | The relationship shows that zero or one contestant can have availed zero or many education opportunities in his life. |
| Contestant ~ CrimeRecord | Zero/One to Zero/Many | The relationship shows that zero or one contestant can have zero or many crime records under his/her name. |
| Degree ~ Education | One to One/Many | The relationship shows that mandatory one degree can be a part of one or more education track. |
| Crew ~ Director | One to One | It shows that mandatory one crew will always have mandatory one director. |
| Crew ~ Producer | One to One | It shows that mandatory one crew will always have mandatory one producer. |

# Database

## Type of database used

We have used SQLite database for our project because its lightweight, easy to install and reliable for small tables.

## Creation of Tables

A text file is attached with this report that contains the queries written for creating the tables of this database with relevant constraints. We will now show one of the queries for creating CONTESTANT table. We will explain this query after this:

**SQL Statement:**

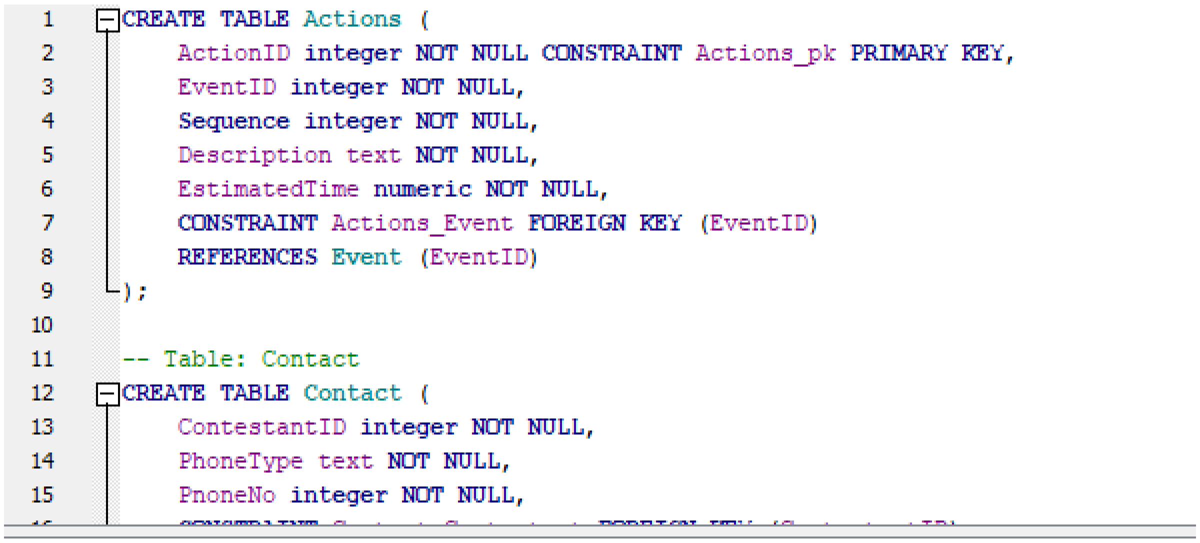
```
-- Table: Contestant
CREATE TABLE Contestant (
    ContestantID integer NOT NULL CONSTRAINT ContestantID PRIMARY KEY,
    NationalID text NOT NULL,
    Name varchar(255) NOT NULL,
    Address varchar(255) NOT NULL,
    City varchar(255) NOT NULL,
    State varchar(255) NOT NULL,
    PostalCode varchar(255) NOT NULL,
    DOB date NOT NULL,
    ProducerRating integer NOT NULL,
    DirectorRating integer NOT NULL,
    Finalist boolean NOT NULL,
    AppearanceRating integer NOT NULL,
    StrengthRating integer NOT NULL,
    Photo blob NOT NULL,
    VideoID text NOT NULL,
    CandidateEssay text NOT NULL,
    CONSTRAINT NationalID UNIQUE (NationalID)
);
```

**Explanation:**

In this Query we are creating a table called "CONTESTANT" and it has seventeen attributes. The attribute "ContestantID" is an attribute that we have created and it is assigned as PRIMARY KEY of this table, which means every entry in this column is UNIQUE and it cannot be NULL. We have assigned the data type for this table as integer. Next we have assigned NationalID as text because the assignment text clearly mentions that this attribute is to be stored in text format. Moving on, I have stored a few of the columns as VARCHAR(255). VARCHAR is basically a data type of a column; in this we can store either numbers or letters and the 255 is put because it's the largest number of characters that can be counted with an 8-bit number. Some columns have INTEGER as column data type because in those cases we only need to store numbers. There's a column for storing Photo of the contestant and that column has the datatype of "BLOB" because it is a collection of binary data stored as a single entity in a database management system used to store images, videos and audio files. Usually its bad form to store images into database but since in this case it is necessary, so we have added it. VideoID is kept as TEXT because it stores the ID of the contestant's video interview. We have also added a column called Finalist in this table which basically signifies that whether the contestant was selected to be on the show or not and has the datatype of BOOLEAN. All columns are set as NOT NULL because all this information is mandatory for the contestant to fill out in the form.

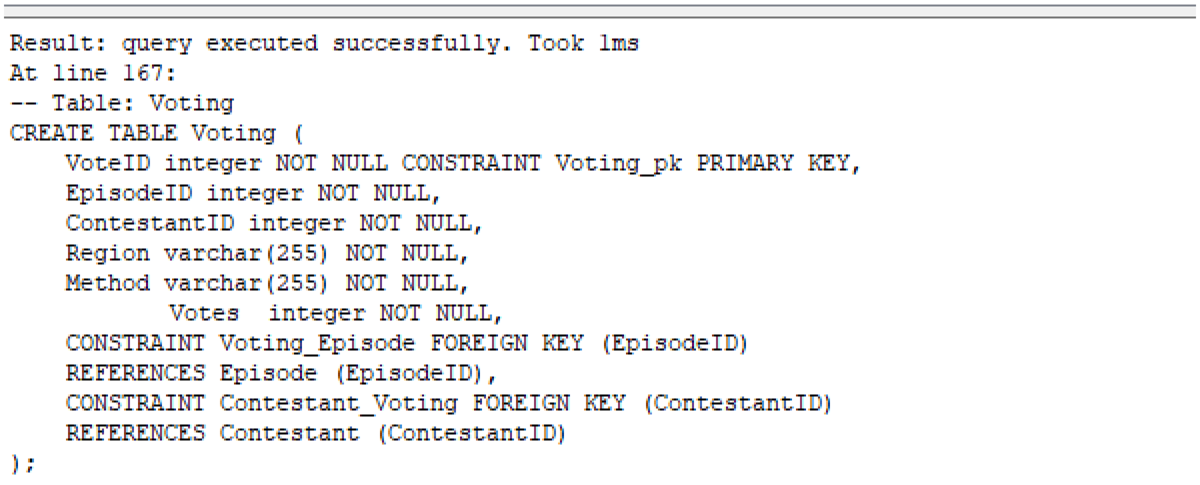
**Result:**

This screenshot shows the creation of all the database tables with the create table query.



```
CREATE TABLE Actions (
    ActionID integer NOT NULL CONSTRAINT Actions_pk PRIMARY KEY,
    EventID integer NOT NULL,
    Sequence integer NOT NULL,
    Description text NOT NULL,
    EstimatedTime numeric NOT NULL,
    CONSTRAINT Actions_Event FOREIGN KEY (EventID)
    REFERENCES Event (EventID)
);

-- Table: Contact
CREATE TABLE Contact (
    ContestantID integer NOT NULL,
    PhoneType text NOT NULL,
    PnoneNo integer NOT NULL,
```



```
Result: query executed successfully. Took 1ms
At line 167:
-- Table: Voting
CREATE TABLE Voting (
    VoteID integer NOT NULL CONSTRAINT Voting_pk PRIMARY KEY,
    EpisodeID integer NOT NULL,
    ContestantID integer NOT NULL,
    Region varchar(255) NOT NULL,
    Method varchar(255) NOT NULL,
           Votes  integer NOT NULL,
    CONSTRAINT Voting_Episode FOREIGN KEY (EpisodeID)
    REFERENCES Episode (EpisodeID),
    CONSTRAINT Contestant_Voting FOREIGN KEY (ContestantID)
    REFERENCES Contestant (ContestantID)
);
```

## Insert Data into Tables

There are a total of seventeen text files submitted with this document, each file contains an SQL statement for data insertion into each table. I will post one of those queries here and explain the process of data insertion.

**SQL Statement:**

```
-- Table: Contestant -> Entered Records: 4
INSERT INTO Contestant
VALUES
(1,'1234','Asad','1222 Harrison St.','San Francisco','CA','94103','1994-11-16',4,5,'TRUE',8,10,'a.png','a.mp4','My name is Asad'),
(2,'2341','Sadia','401 Pvt. Drive','Atlanta','GA','30301','1995-12-22',5,5,'TRUE',10,10,'b.png','b.mp4','My name is Sadia'),
(3,'3421','Khushboo','50 Moon Dr.','San Francisco','CA','94101','1996-11-12',5,5,'TRUE',10,9,'c.png','c.mp4','My name is Khushboo'),
(4,'4321','Ronak','101 Sea Point','Richmond','CA','94001','1993-1-12',2,2,'FALSE',10,4,'d.png','d.mp4','My name is Ronak');
```

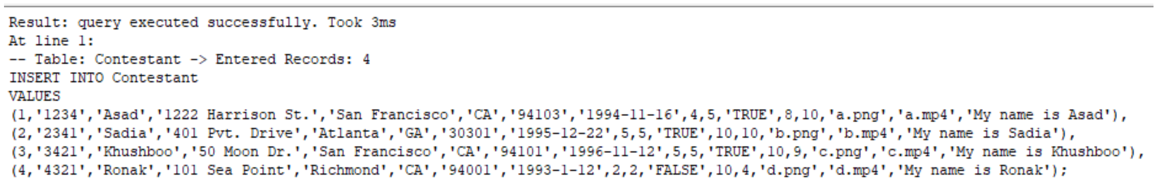
**Explanation:**

In this SQL statement I am inserting records of four different contestants. All contestants data is entered exactly in the same sequence as the table's columns are. The general rule of thumb for insertion is that if the table you are entering data into has its Primary Key used as Foreign Key in another table so you have to make sure that you first insert data into the table where there is primary key and then enter data into the table that has the foreign key. Also, you have to make sure that the foreign key data should exist in the table where it is a primary key. I entered data of three contestants of which three were selected to take part into the TV Show and one was rejected because of his crime record as it is clearly stated in the text of the project that Producers and Directors want people that have no criminal records and have a positive feedback in their back ground check from their previous employers and also have good comments from the educational institutions that they might have received an education in.

**Result:**

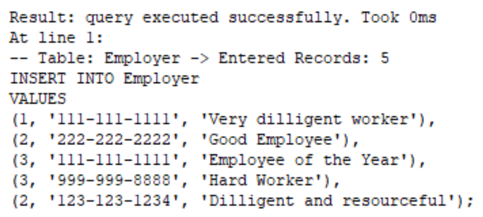
Insert Queries need to be executed separately so I am going to include 17 screenshots one for each table.

<u>Contestant:</u>



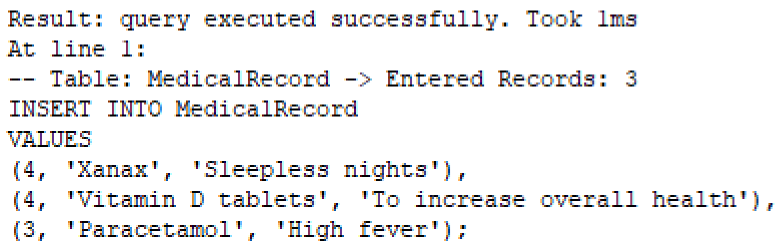
```
Result: query executed successfully. Took 3ms
At line 1:
-- Table: Contestant -> Entered Records: 4
INSERT INTO Contestant
VALUES
(1,'1234','Asad','1222 Harrison St.','San Francisco','CA','94103','1994-11-16',4,5,'TRUE',8,10,'a.png','a.mp4','My name is Asad'),
(2,'2341','Sadia','401 Pvt. Drive','Atlanta','GA','30301','1995-12-22',5,5,'TRUE',10,10,'b.png','b.mp4','My name is Sadia'),
(3,'3421','Khushboo','50 Moon Dr.','San Francisco','CA','94101','1996-11-12',5,5,'TRUE',10,9,'c.png','c.mp4','My name is Khushboo'),
(4,'4321','Ronak','101 Sea Point','Richmond','CA','94001','1993-1-12',2,2,'FALSE',10,4,'d.png','d.mp4','My name is Ronak');
```

<u>Employer:</u>



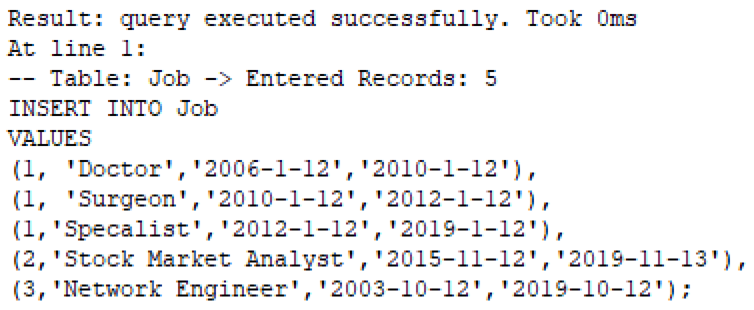
```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Employer -> Entered Records: 5
INSERT INTO Employer
VALUES
(1, '111-111-1111', 'Very dilligent worker'),
(2, '222-222-2222', 'Good Employee'),
(3, '111-111-1111', 'Employee of the Year'),
(3, '999-999-8888', 'Hard Worker'),
(2, '123-123-1234', 'Dilligent and resourceful');
```

MedicalRecord



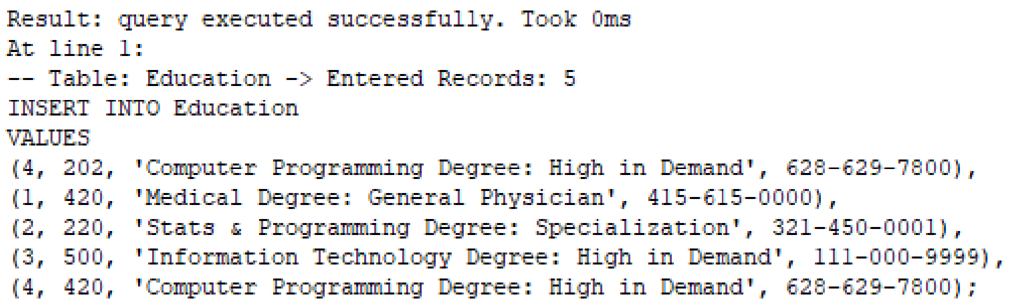
```
Result: query executed successfully. Took 1ms
At line 1:
-- Table: MedicalRecord -> Entered Records: 3
INSERT INTO MedicalRecord
VALUES
(4, 'Xanax', 'Sleepless nights'),
(4, 'Vitamin D tablets', 'To increase overall health'),
(3, 'Paracetamol', 'High fever');
```

Job



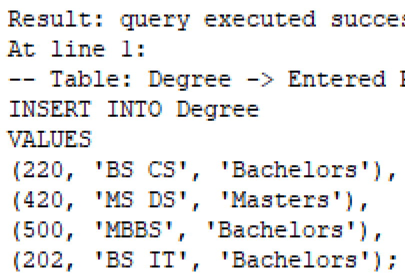
```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Job -> Entered Records: 5
INSERT INTO Job
VALUES
(1, 'Doctor','2006-1-12','2010-1-12'),
(1, 'Surgeon','2010-1-12','2012-1-12'),
(1,'Specalist','2012-1-12','2019-1-12'),
(2,'Stock Market Analyst','2015-11-12','2019-11-13'),
(3,'Network Engineer','2003-10-12','2019-10-12');
```

Education



```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Education -> Entered Records: 5
INSERT INTO Education
VALUES
(4, 202, 'Computer Programming Degree: High in Demand', 628-629-7800),
(1, 420, 'Medical Degree: General Physician', 415-615-0000),
(2, 220, 'Stats & Programming Degree: Specialization', 321-450-0001),
(3, 500, 'Information Technology Degree: High in Demand', 111-000-9999),
(4, 420, 'Computer Programming Degree: High in Demand', 628-629-7800);
```

Degree



```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Degree -> Entered Records: 4
INSERT INTO Degree
VALUES
(220, 'BS CS', 'Bachelors'),
(420, 'MS DS', 'Masters'),
(500, 'MBBS', 'Bachelors'),
(202, 'BS IT', 'Bachelors');
```

CrimeRecord

```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: CrimeRecord -> Entered Records: 2
INSERT INTO CrimeRecord
VALUES
(4,'2006-11-12','Theft','Grand Theft Auto', 'Jailed'),
(4,'2003-1-12','Cybercrime', 'Fraud in stocks', 'Fined');
```

Contact

```
Result: query executed successfully. Took 1ms
At line 1:
-- Table: Contact -> Entered Records: 8
INSERT INTO Contact
VALUES
(1, 'Daytime Phone', '628-629-4840'),
(1, 'Nighttime Phone', '628-629-4841'),
(2, 'Daytime Phone', '628-629-4842'),
(2, 'Nighttime Phone', '628-629-4843'),
(3, 'Daytime Phone', '628-629-4844'),
(3, 'Nighttime Phone', '628-629-4845'),
(4, 'Daytime Phone', '628-629-4846'),
(4, 'Nighttime Phone', '628-629-4847');
```

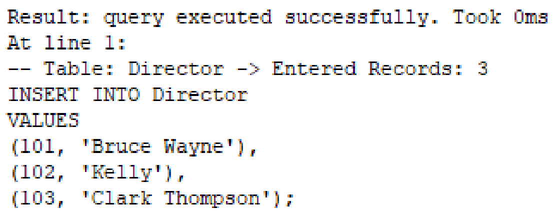
Crew

```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Crew -> Entered Records: 6
INSERT INTO Crew
VALUES
(1, 101, 202, 101),
(2, 102, 201, 103),
(3, 103, 202, 101),
(4, 105, 202, 101),
(5, 110, 203, 103),
(6, 114, 202, 102);
```

Producer

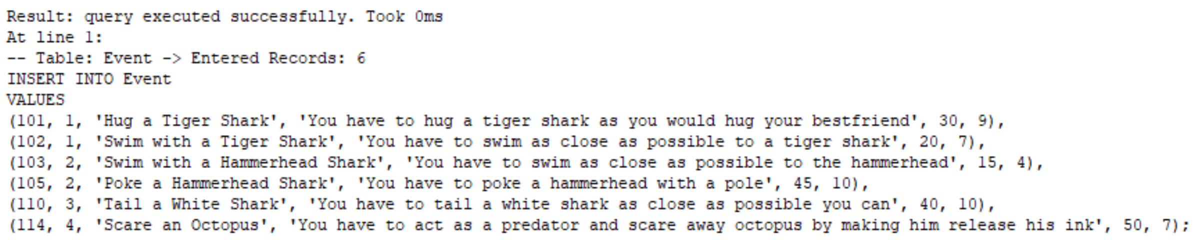
```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Producer -> Entered Records: 3
INSERT INTO Producer
VALUES
(201, 'Brian Miller'),
(202, 'Carl Lucus'),
(203, 'Xander Cage');
```

## Director



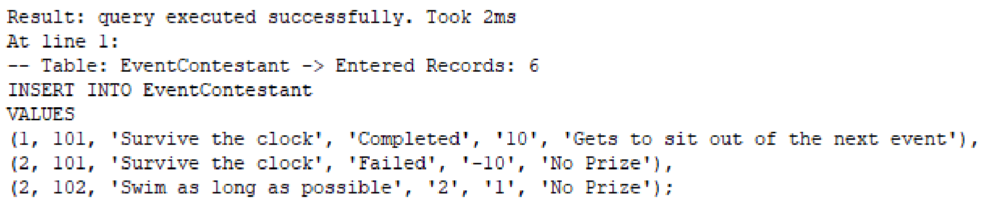
```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Director -> Entered Records: 3
INSERT INTO Director
VALUES
(101, 'Bruce Wayne'),
(102, 'Kelly'),
(103, 'Clark Thompson');
```

## Event



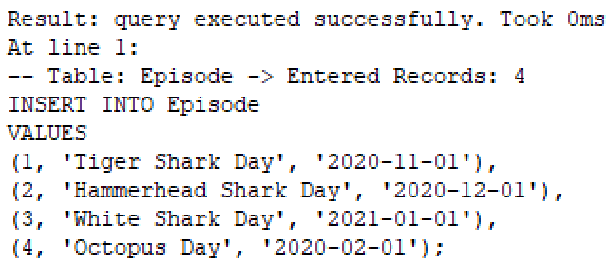
```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Event -> Entered Records: 6
INSERT INTO Event
VALUES
(101, 1, 'Hug a Tiger Shark', 'You have to hug a tiger shark as you would hug your bestfriend', 30, 9),
(102, 1, 'Swim with a Tiger Shark', 'You have to swim as close as possible to a tiger shark', 20, 7),
(103, 2, 'Swim with a Hammerhead Shark', 'You have to swim as close as possible to the hammerhead', 15, 4),
(105, 2, 'Poke a Hammerhead Shark', 'You have to poke a hammerhead with a pole', 45, 10),
(110, 3, 'Tail a White Shark', 'You have to tail a white shark as close as possible you can', 40, 10),
(114, 4, 'Scare an Octopus', 'You have to act as a predator and scare away octopus by making him release his ink', 50, 7);
```

## EventContestant



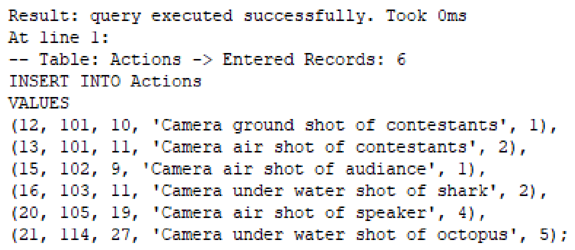
```
Result: query executed successfully. Took 2ms
At line 1:
-- Table: EventContestant -> Entered Records: 6
INSERT INTO EventContestant
VALUES
(1, 101, 'Survive the clock', 'Completed', '10', 'Gets to sit out of the next event'),
(2, 101, 'Survive the clock', 'Failed', '-10', 'No Prize'),
(2, 102, 'Swim as long as possible', '2', '1', 'No Prize');
```

## Episode



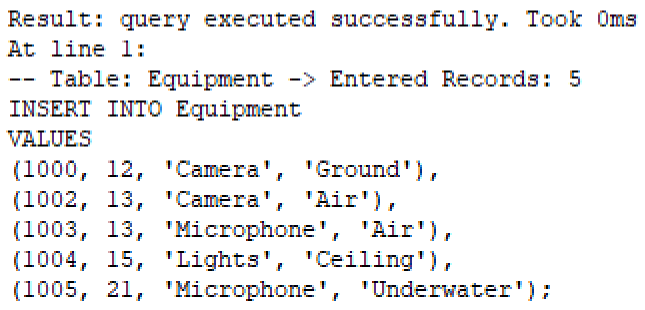
```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Episode -> Entered Records: 4
INSERT INTO Episode
VALUES
(1, 'Tiger Shark Day', '2020-11-01'),
(2, 'Hammerhead Shark Day', '2020-12-01'),
(3, 'White Shark Day', '2021-01-01'),
(4, 'Octopus Day', '2020-02-01');
```

## Actions



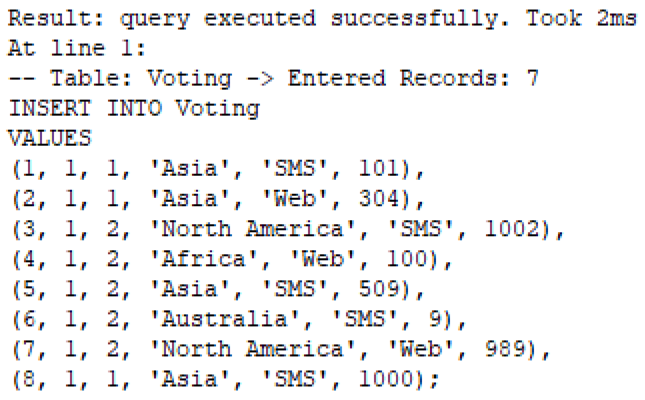
```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Actions -> Entered Records: 6
INSERT INTO Actions
VALUES
(12, 101, 10, 'Camera ground shot of contestants', 1),
(13, 101, 11, 'Camera air shot of contestants', 2),
(15, 102, 9, 'Camera air shot of audiance', 1),
(16, 103, 11, 'Camera under water shot of shark', 2),
(20, 105, 19, 'Camera air shot of speaker', 4),
(21, 114, 27, 'Camera under water shot of octopus', 5);
```

Equipment



```
Result: query executed successfully. Took 0ms
At line 1:
-- Table: Equipment -> Entered Records: 5
INSERT INTO Equipment
VALUES
(1000, 12, 'Camera', 'Ground'),
(1002, 13, 'Camera', 'Air'),
(1003, 13, 'Microphone', 'Air'),
(1004, 15, 'Lights', 'Ceiling'),
(1005, 21, 'Microphone', 'Underwater');
```

Voting



```
Result: query executed successfully. Took 2ms
At line 1:
-- Table: Voting -> Entered Records: 7
INSERT INTO Voting
VALUES
(1, 1, 1, 'Asia', 'SMS', 101),
(2, 1, 1, 'Asia', 'Web', 304),
(3, 1, 2, 'North America', 'SMS', 1002),
(4, 1, 2, 'Africa', 'Web', 100),
(5, 1, 2, 'Asia', 'SMS', 509),
(6, 1, 2, 'Australia', 'SMS', 9),
(7, 1, 2, 'North America', 'Web', 989),
(8, 1, 1, 'Asia', 'SMS', 1000);
```

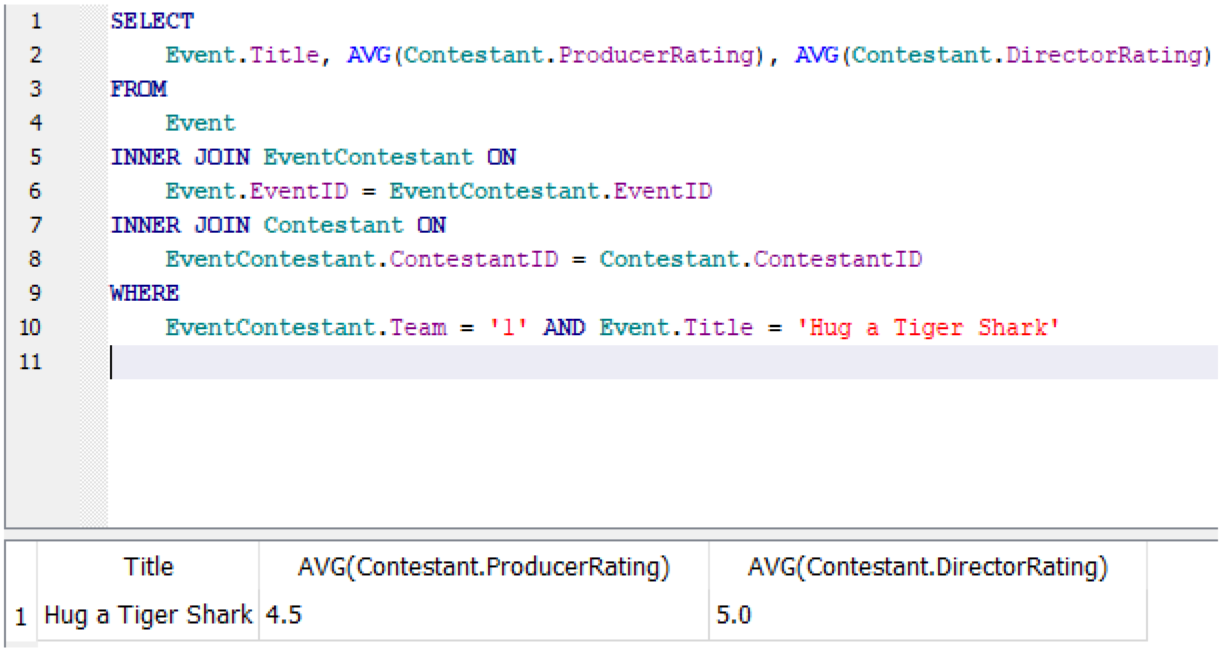
## DBMS QUERIES

**Query 1:** Calculate the average producer and director ratings for a specific event's team members (you pick the event title).

**SQL Statement:**

```
SELECT
        Event.Title, AVG(Contestant.ProducerRating), AVG(Contestant.DirectorRating)
FROM
        Event
INNER JOIN EventContestant ON
        Event.EventID = EventContestant.EventID
INNER JOIN Contestant ON
        EventContestant.ContestantID = Contestant.ContestantID
WHERE
        EventContestant.Team = '1' AND Event.Title = 'Hug a Tiger Shark'
```

**Result:**



```
SELECT
    Event.Title, AVG(Contestant.ProducerRating), AVG(Contestant.DirectorRating)
FROM
    Event
INNER JOIN EventContestant ON
    Event.EventID = EventContestant.EventID
INNER JOIN Contestant ON
    EventContestant.ContestantID = Contestant.ContestantID
WHERE
    EventContestant.Team = '1' AND Event.Title = 'Hug a Tiger Shark'
```

| | Title | AVG(Contestant.ProducerRating) | AVG(Contestant.DirectorRating) |
|---|---|---|---|
| 1 | Hug a Tiger Shark | 4.5 | 5.0 |

**Explanation:**

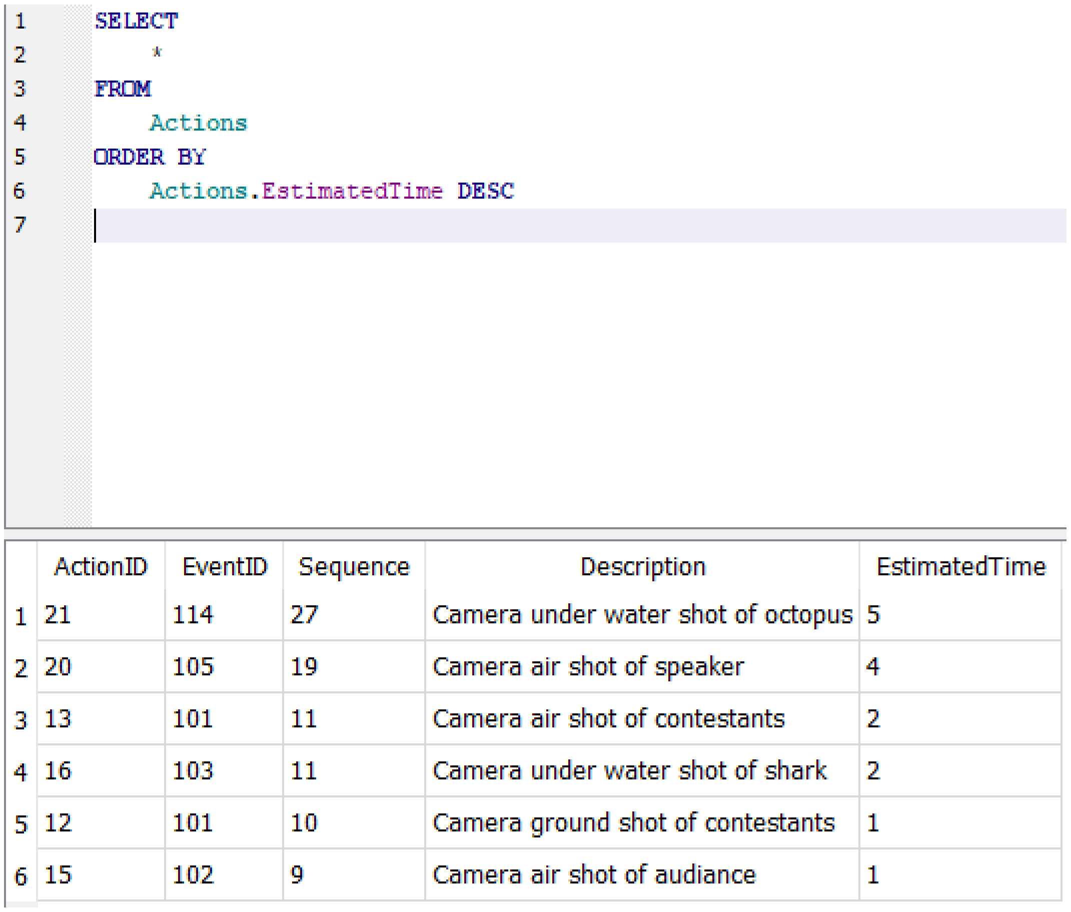
In this query we wanted to show the average of producer rating and director rating in the event called "Hug a Tiger Shark" of team number 1. We joined the following tables "Event", "EventContestant" and "Contestant" by using Inner Join because we wanted records that were present in all three tables. This also enabled us to get the following columns, such as Event.title that is present in "Event" table and calculate the average of both producer rating and director rating which is present in "Contestant" table. Then as we need to show the average of producer and director rating of a specific event of our choice for a specific team. Hence, we chose only team number 1 and title and the event = 'Hug a Tiger Shark' to get the relevant results.

**<u>Query 2:</u>** Determine which actions will take the longest. Rank all actions from longest to shortest.

**SQL Statement:**

```
SELECT
        *
FROM
        Actions
ORDER BY
        Actions.EstimatedTime DESC
```

**Result:**



```
SELECT
    *
FROM
    Actions
ORDER BY
    Actions.EstimatedTime DESC
```

| | ActionID | EventID | Sequence | Description | EstimatedTime |
|---|---|---|---|---|---|
| 1 | 21 | 114 | 27 | Camera under water shot of octopus | 5 |
| 2 | 20 | 105 | 19 | Camera air shot of speaker | 4 |
| 3 | 13 | 101 | 11 | Camera air shot of contestants | 2 |
| 4 | 16 | 103 | 11 | Camera under water shot of shark | 2 |
| 5 | 12 | 101 | 10 | Camera ground shot of contestants | 1 |
| 6 | 15 | 102 | 9 | Camera air shot of audiance | 1 |

**Explanation:**

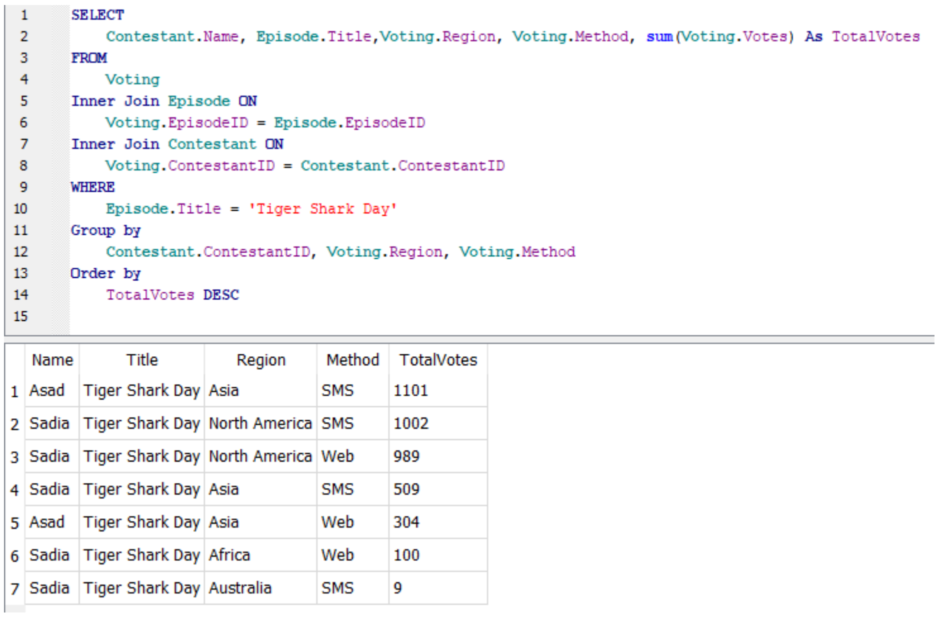
In this query, we wanted to show the highest estimated time of event by descending order. To calculate this, we used only one table called “Actions”. Since its not mentioned to show any specific columns here apart from “EstimatedTime”. We have decided to show every column in the table and we order the records in descending order using “EstimatedTime column. This query gave us all rows in Actions table in descending order of estimated time.

**<u>Query 3:</u>** List each contestant's total votes by region and method for a specific episode (you pick the episode title). Rank this from highest to lowest.

**SQL Statement:**

```
SELECT
        Contestant.Name, Episode.Title,Voting.Region, Voting.Method, sum(Voting.Votes) As
TotalVotes
FROM
        Voting
Inner Join Episode ON
        Voting.EpisodeID = Episode.EpisodeID
Inner Join Contestant ON
        Voting.ContestantID = Contestant.ContestantID
WHERE
        Episode.Title = 'Tiger Shark Day'
Group by
        Contestant.ContestantID, Voting.Region, Voting.Method
Order by
        TotalVotes DESC
```

**Result:**



```
SELECT
    Contestant.Name, Episode.Title,Voting.Region, Voting.Method, sum(Voting.Votes) As TotalVotes
FROM
    Voting
Inner Join Episode ON
    Voting.EpisodeID = Episode.EpisodeID
Inner Join Contestant ON
    Voting.ContestantID = Contestant.ContestantID
WHERE
    Episode.Title = 'Tiger Shark Day'
Group by
    Contestant.ContestantID, Voting.Region, Voting.Method
Order by
    TotalVotes DESC
```

| | Name | Title | Region | Method | TotalVotes |
|---|---|---|---|---|---|
| 1 | Asad | Tiger Shark Day | Asia | SMS | 1101 |
| 2 | Sadia | Tiger Shark Day | North America | SMS | 1002 |
| 3 | Sadia | Tiger Shark Day | North America | Web | 989 |
| 4 | Sadia | Tiger Shark Day | Asia | SMS | 509 |
| 5 | Asad | Tiger Shark Day | Asia | Web | 304 |
| 6 | Sadia | Tiger Shark Day | Africa | Web | 100 |
| 7 | Sadia | Tiger Shark Day | Australia | SMS | 9 |

**Explanation**

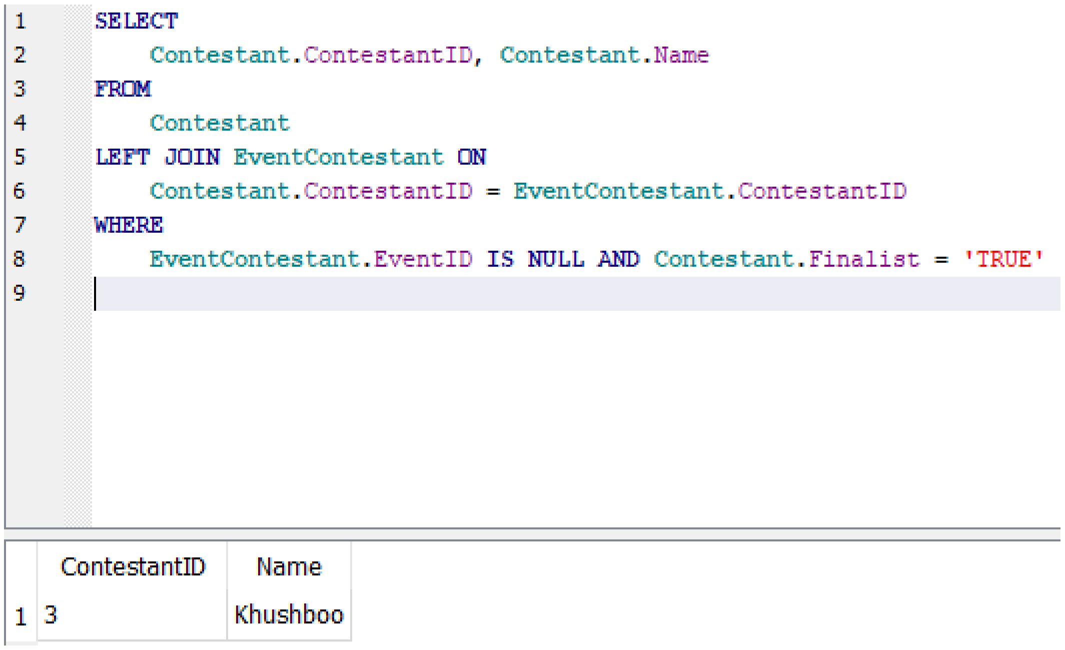
In this query we were asked to list votes for each contestant by “Region” and “Method”. Now, we wanted to show the contestant’s name, title of the episode, region, voting method and number of votes of the episode called “Tiger Shark Day”. We use inner join to get matching records from “Voting”, “Episode” and “Contestant” and chose “Tiger Shark Day” episode. We used group by command to group the records by Contestant’s ID, Voting’s region and method. We used the sum command to get total number of votes and we named the column as “TotalVotes” and then ordered the result by number of votes in descending order.

**<u>Query 4:</u>** Identify which contestants have not participated in any events.

**SQL Statement:**

```
SELECT
	Contestant.ContestantID, Contestant.Name
FROM
	Contestant
LEFT JOIN EventContestant ON
	Contestant.ContestantID = EventContestant.ContestantID
WHERE
	EventContestant.EventID IS NULL AND Contestant.Finalist = 'TRUE'
```

**Result:**



```
SELECT
    Contestant.ContestantID, Contestant.Name
FROM
    Contestant
LEFT JOIN EventContestant ON
    Contestant.ContestantID = EventContestant.ContestantID
WHERE
    EventContestant.EventID IS NULL AND Contestant.Finalist = 'TRUE'

```

|  | ContestantID | Name |
|---|---|---|
| 1 | 3 | Khushboo |

**Explanation:**

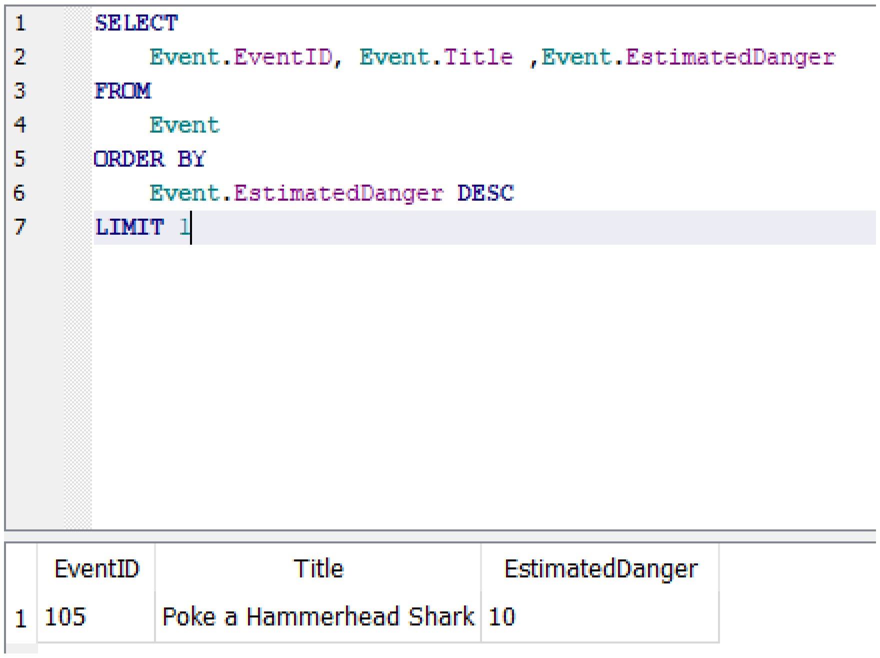
In this query, we were asked to identify contestants that weren't a part of any event. So we used left join in this case to join the "Contestant" table with "EventContestant" table because left join includes all rows from the left hand side and matching rows from right hand side. This will help us identifying contestants that aren't in "EventContestant" table. Then we used the condition IS NULL with EventContestant.EventID to see which contestant did not have information in it along with it we also made sure if the person was a finalist too. Basically, finalist here refers to the fact that did the producer and director select the contestant for the show or not. Having "TRUE" in finalist means that they were selected and "FALSE" means they weren't selected to be in the show.

**<u>Query 5:</u>** What is the highest estimated danger level for any event?

**SQL Statement:**

```
SELECT
        Event.EventID, Event.Title ,Event.EstimatedDanger
FROM
        Event
ORDER BY
        Event.EstimatedDanger DESC
LIMIT 1
```

**Result:**



```
SELECT
    Event.EventID, Event.Title ,Event.EstimatedDanger
FROM
    Event
ORDER BY
    Event.EstimatedDanger DESC
LIMIT 1
```

| | EventID | Title | EstimatedDanger |
|---|---|---|---|
| 1 | 105 | Poke a Hammerhead Shark | 10 |

**Explanation:**

In this query we had to show the highest "EstimatedDanger" of any event. All the information we need to calculate this was present in "Event" table. We selected the relevant columns called as "Event.EventID", "Event.Title" and "Event.EstimatedDanger" in Event table and order them by the column "EstimatedDanger" in descending order.

**<u>Query 6:</u>** Show a relational expression for any one query

The chosen query for this question is **Query 1** which is as follows:

**Statement:**

```
SELECT
       Event.Title, AVG(Contestant.ProducerRating), AVG(Contestant.DirectorRating)
FROM
       Event
INNER JOIN EventContestant ON
       Event.EventID = EventContestant.EventID
INNER JOIN Contestant ON
       EventContestant.ContestantID = Contestant.ContestantID
WHERE
       EventContestant.Team = '1' AND Event.Title = 'Hug a Tiger Shark'
```

**Expression:**

R1= EventContestant $\bowtie_{\text{<ContestantID = ContestantID >}}$ Contestant

R2= Event $\bowtie_{\text{<EventID = EventID>}}$ (R1)

R3= $\sigma_{\text{<Title = “Hug a Tiger Shark”>}}$ (R2)

R4= $\pi_{\text{<Title, avg(ProducerRating), avg(DirectorRating)>}}$ (R3)

<u>Result:</u>

$\pi_{\text{<Title, avg(ProducerRating), avg(DirectorRating)>}}$ ($\sigma_{\text{<Title = “Hug a Tiger Shark” , Team = “1”>}}$ (Event $\bowtie_{\text{<EventID = EventID>}}$ (EventContestant $\bowtie_{\text{<ContestantID = ContestantID >}}$ Contestant)))

## Project Presentation

### Presentation Tool

We used google slides to create our project presentation

### Presentation Link

https://docs.google.com/presentation/d/1F264ImfRz5S5mej6_6nkH5p-IMnh4U0CjZm4a5zbmQ8/edit?usp=sharing

The slides are also attached with this submission.

## Project Video

It is stored on google drive.

URL: https://drive.google.com/open?id=1QJO6PfVLsPQoIUBK9Aq0wxhIPgk4PtgN

It is also attached with this submission.

## Work Distribution

The portion of work done by each group member is listed below

1. Asad Mahmood
    a. ERD
    b. Database Creation and Insertion of Dummy Data
    c. Proof checking all the work done by members
2. Theewart Jindapoo
    a. Database Queries
3. Khushboo Patel
    a. Project Report
4. Atik Ilman Hossain
    a. Compiling all the work
    b. Project Video
    c. Project Slides
    d. Proof checking all the work done by members

## References

[1]"SQLite Advantages and Disadvantages - javatpoint", www.javatpoint.com, 2019. [Online]. Available: https://www.javatpoint.com/sqlite-advantages-and-disadvantages. [Accessed: 12- Dec- 2019].

[2]"Binary large object", En.wikipedia.org, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Binary_large_object. [Accessed: 12- Dec- 2019].