Outreach

Emmanuel Menier
Leonardo Orea-Amador
Ivan Perez
Jaime Salas

Document version: Assignment 3

Table of Contents

Database Scope:

We will develop a system for The University of Texas at El Paso's Computer Science Department to aid in the tracking of volunteer hours and promote awareness of Outreach events and create reports. The goal is to manage community service hours, future and past events, and demographics of event turnout. The system will provide information about events where volunteer hours may be available, how many volunteer hours a student has completed, how many hours all students have completed, and volunteer's information(skills, genders, ID, languages, classification, name, email, volunteer hours) in a profile. The website will follow UTEP design rules and regulations which can be found at https://www.utep.edu/university-communications/resources/graphic-identity-guide.html.

The system will provide a way for faculty, administration, and students to log in and view corresponding information. Faculty will be able to access all volunteer information, verify volunteer events, and promote said events. Volunteers will be able to log their volunteer hours, sign up for new events, view their current volunteer hours, as well as repeal the sign up for an event upon admin approval. The administration will be able to upload new events, promote them, and make sure logged hours for events are accurate. Any UTEP student can have access to this portal but are limited to event requirements that event organizers see fit.

Requirements:

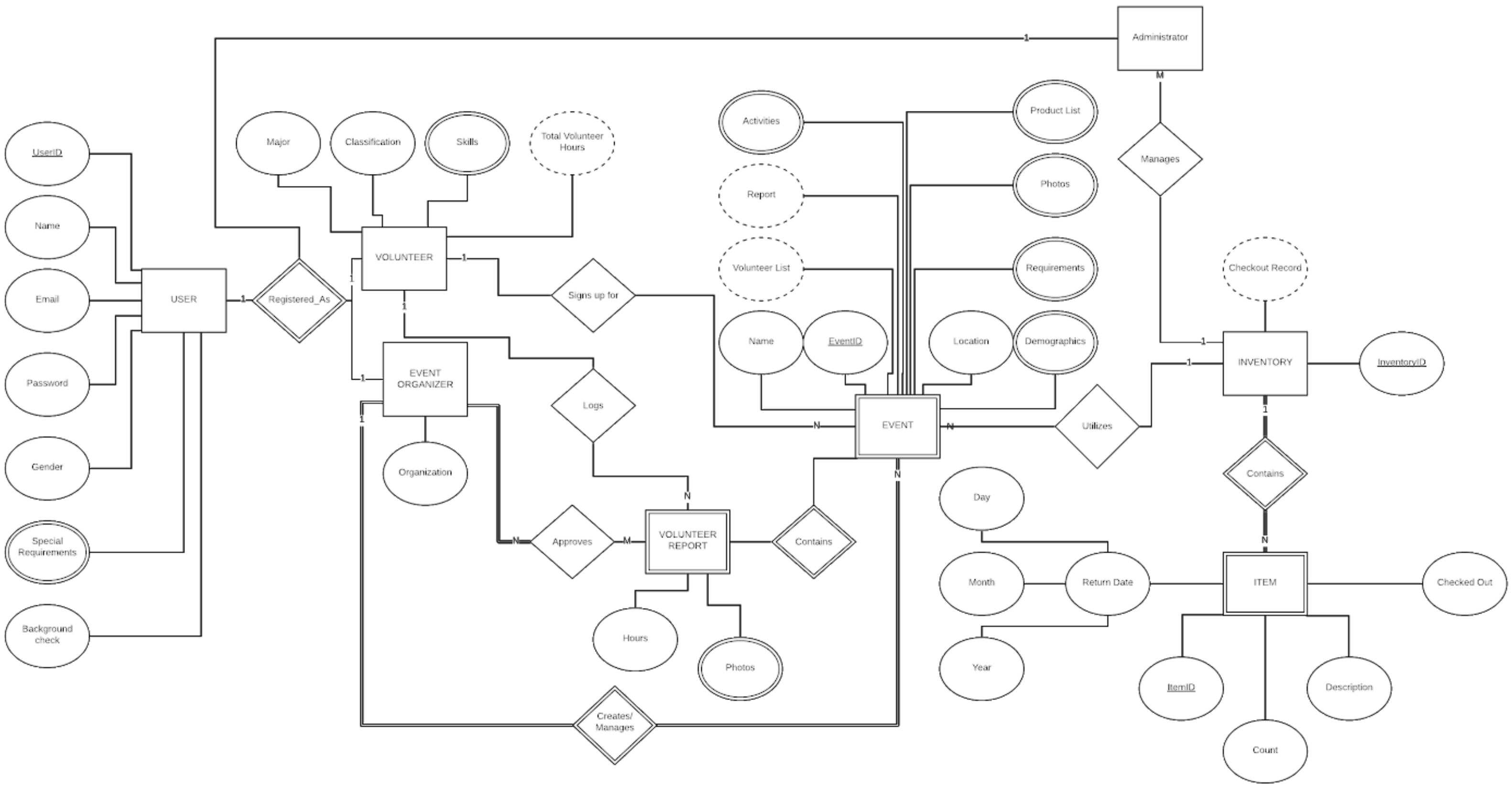
1. The system shall keep a documentation record of events.
2. The system shall be able to organize and describe individual events.
3. The system shall allow admins to manage, edit, and delete the events.
4. The system shall allow admins to sign in and manage events
5. The system shall allow students to sign up as members for specific events.
6. The system shall be able to capture optional or required information of the event members like age, classification, ID numbers, college, etc.
7. The system shall be able to annotate the requirements for the event attendees such as writing material, equipment, or other types of required equipment.
8. The system shall be able to capture special requirements from the attendees (e.g. wheelchair, gluten-free, etc).
9. The system shall allow the member attendees to report their attendance and participation hours.
10. The system shall allow the admins to approve the member attendance reports based on their participation hours.

11. The system shall keep track of all hours reported by the members.
12. The system shall keep a record of activities realized at the event or any worthwhile incident.
13. The system shall be able to create and distribute a log of pictures taken during the event.
14. The system shall allow the event members to upload pictures to their respective events
15. The system shall keep track of the total number of attendees for every event and be able to deliver reports of attendance.
16. The system must keep track of where the impact is being made the most, and what areas are being visited the most.
17. The system will advertise to students about future events, for students or schools being outreached too.
18. The system will also notify the users whether they require a background check
19. The system shall provide the option to allocate an inventory with items and their description
20. The system shall be able to verify the flow of items, whether the items have been checked in or out from the inventory as well as any important description (e.g. damage or returning date).
21. The system shall have a report function executed by admins which lets them obtain a report by an event.

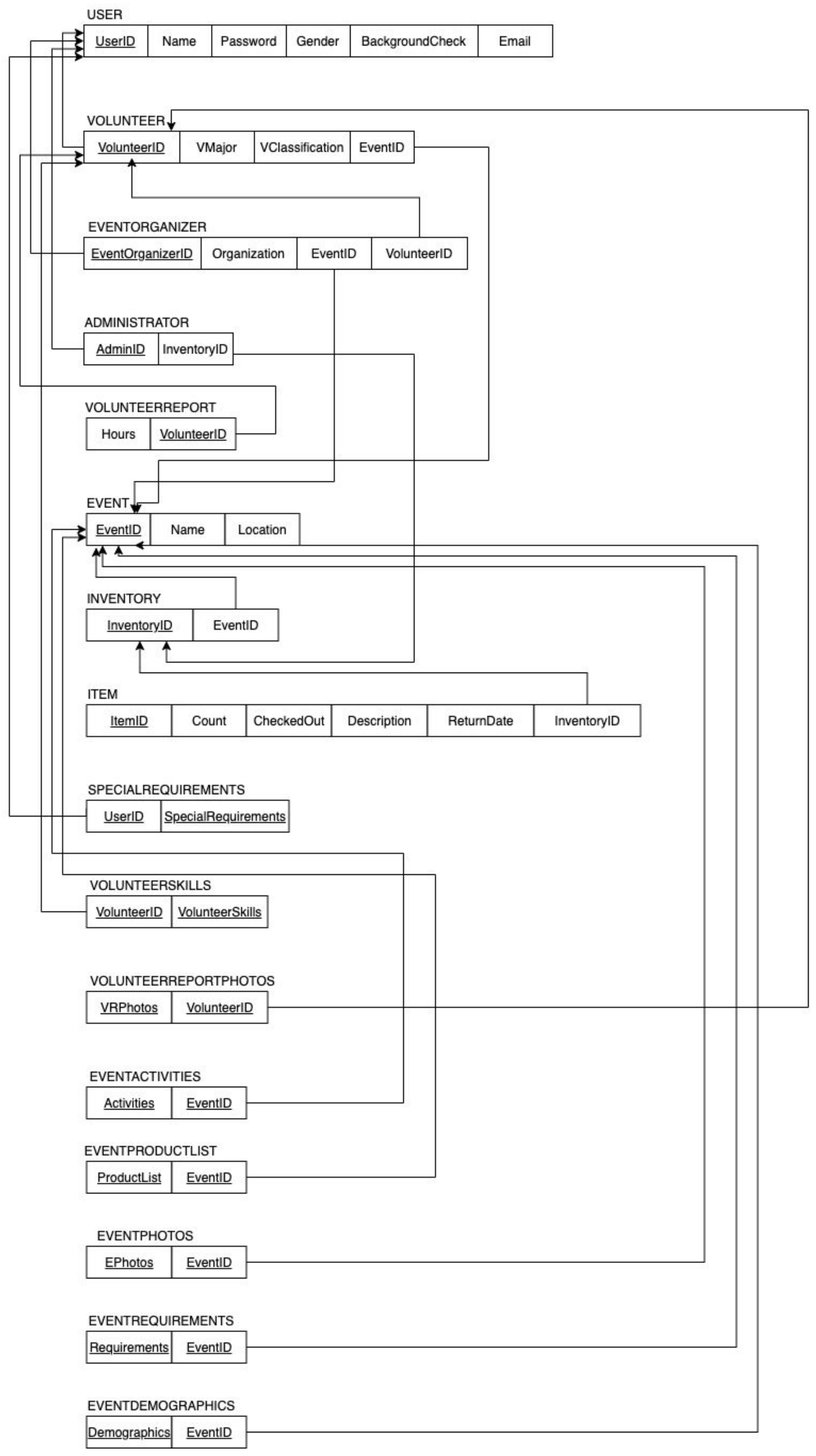
Assumptions:

We assume that users will know what roles they have whether they are admins, volunteers, or event coordinators. We assume event budgets are handled outside of the system. We assume background checks can be confirmed in the system but the background check itself won't be done through the system. We assume all volunteers are students from UTEP and sign in to the system using their UTEP credentials. We assume any special guests or volunteers will be handled by a case to case basis making administrators or other faculty responsible for their account.

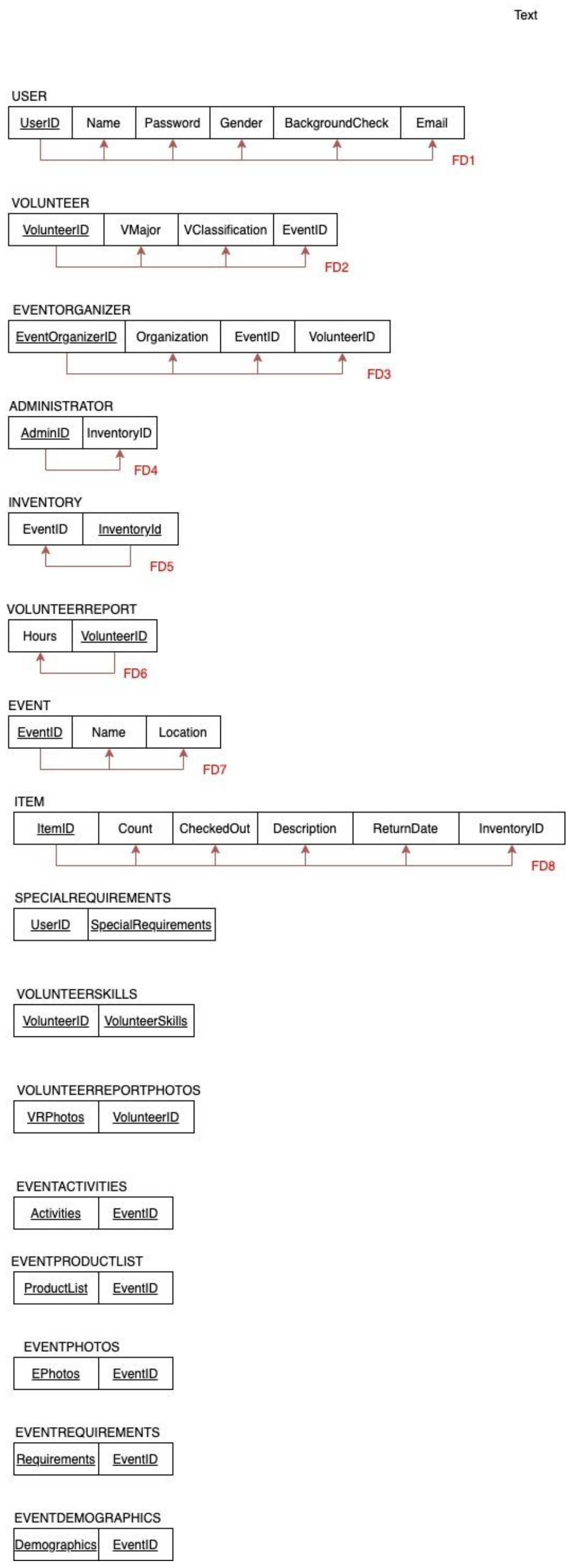
Entity-Relational Model



Relational Model



Normalized Schema



<u>USER</u>

- USER is in 1NF since all the attributes are atomic.
- USER is in 2NF since it is in 1NF and depends on the primary key UserID.
- USER is in 3NF since it is in 2NF and no non-prime attributes depend on another non-primary attribute.

<u>VOLUNTEER</u>

- VOLUNTEER is in 1NF since all the attributes are atomic.
- VOLUNTEER is in 2NF since it is in 1NF and depends on the primary key VolunteerID.
- VOLUNTEER is in 3NF since it is in 2NFand no non-prime attributes depend on another non-primary attribute.

<u>EVENTORGANIZER</u>

- EVENTORGANIZER is in 1NF since all the attributes are atomic.
- EVENTORGANIZER is in 2NF since it is in 1NF and depends on the primary key EventOrganizerID.
- EVENTORGANIZER is in 3NF since it is in 2NF and no non-prime attributes depend on another non-primary attribute

<u>ADMINISTRATOR</u>

- ADMINISTRATOR is in 1NF since all the attributes are atomic.
- ADMINISTRATOR is in 2NF since it is in 1NF and depends on the primary key AdminID.
- ADMINISTRATOR is in 3NF since it is in 2NF and no non-prime attributes depend on another non-primary attribute.

<u>INVENTORY</u>

- INVENTORY is in 1NF since all the attributes are atomic.
- INVENTORY is in 2NF since it is in 1NF and depends on the primary key InventoryID.
- INVENTORY is in 3NF since it is in 2NF and no non-prime attributes depend on another non-primary attribute.

<u>VOLUNTEERREPORT</u>

- VOLUNTEERREPORT is in 1NF since all the attributes are atomic.
- VOLUNTEERREPORT is in 2NF since it is in 1NF and depends on the primary key VolunteerID.
- VOLUNTEERREPORT is in 3NF since it is in 2NF and no non-prime attributes depend on another non-primary attribute.

<u>EVENT</u>

- EVENT is in 1NF since all the attributes are atomic.
- EVENT is in 2NF since it is in 1NF and depends on the primary key EventID.
- EVENT is in 3NF since it is in 2NF and no non-prime attributes depend on another non-primary attribute.

<u>ITEM</u>

- ITEM is in 1NF since all the attributes are atomic.
- ITEM is in 2NF since it is in 1NF and depends on the primary key ItemID.
- ITEM is in 3NF since it is in 2NF and no non-prime attributes depend on another non-primary attribute.

<u>SPECIALREQUIREMENTS</u>

- SPECIALREQUIREMENTS is in 1NF since all the attributes are atomic.
- SPECIALREQUIREMENTS is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- SPECIALREQUIREMENTS is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

VOLUNTEERSKILLS

- VOLUNTEERSKILLS is in 1NF since all the attributes are atomic.
- VOLUNTEERSKILLS is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- VOLUNTEERSKILLS is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

VOLUNTEERREPORTPHOTOS

- VOLUNTEERREPORTPHOTOS is in 1NF since all the attributes are atomic.
- VOLUNTEERREPORTPHOTOS is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- VOLUNTEERREPORTPHOTOS is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

EVENTACTIVITIES

- EVENTACTIVITIES is in 1NF since all the attributes are atomic.
- EVENTACTIVITIES is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- EVENTACTIVITIES is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

EVENTPRODUCTLIST

- EVENTPRODUCTLIST is in 1NF since all the attributes are atomic.
- EVENTPRODUCTLIST is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- EVENTPRODUCTLIST is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

EVENTPHOTOS

- EVENTPHOTOS is in 1NF since all the attributes are atomic.
- EVENTPHOTOS is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- EVENTPHOTOS is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

EVENTREQUIREMENTS

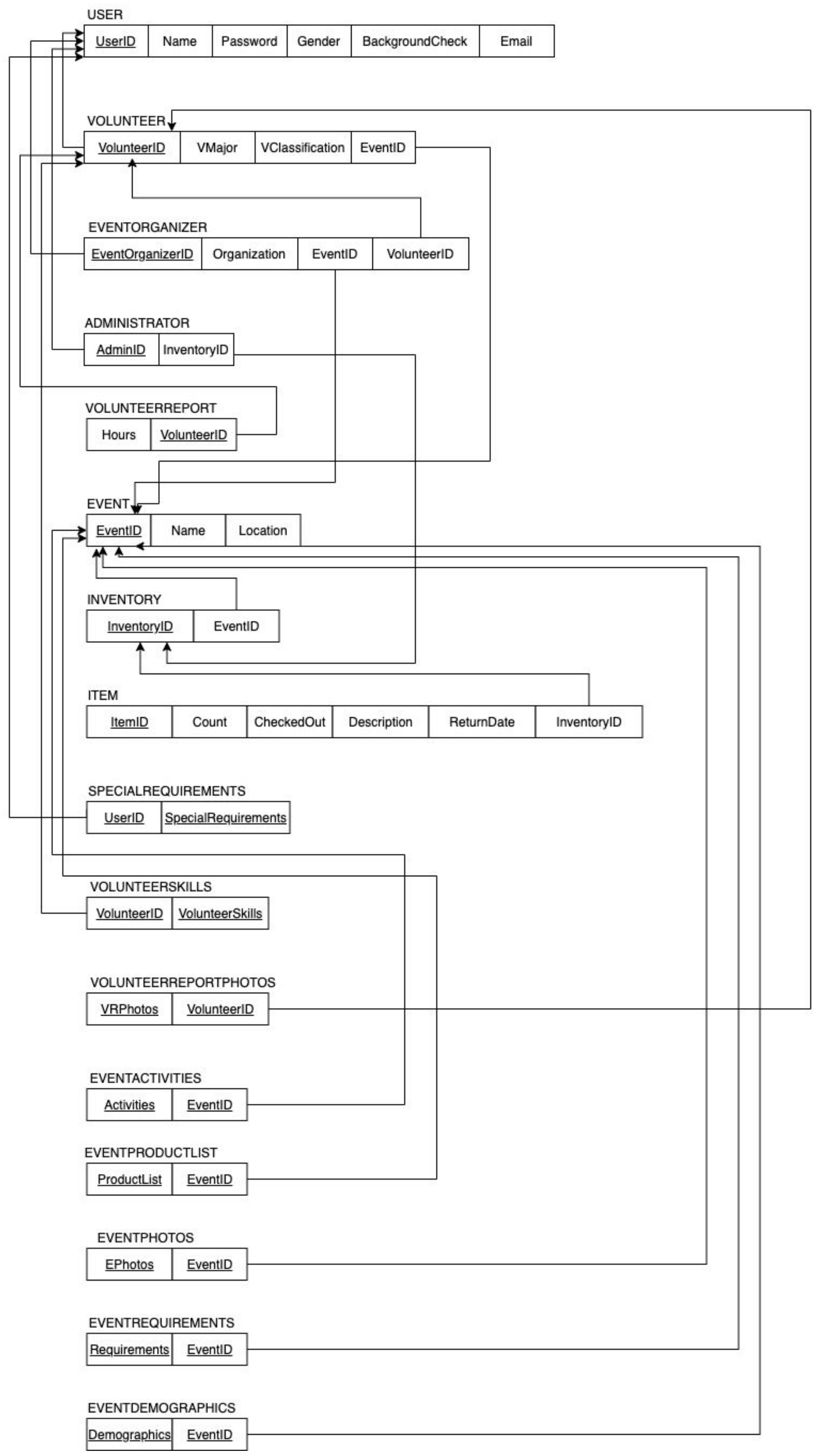
- EVENTREQUIREMENTS is in 1NF since all the attributes are atomic.
- EVENTREQUIREMENTS is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- EVENTREQUIREMENTS is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

EVENTDEMOGRAPHICS

- EVENTDEMOGRAPHICS is in 1NF since all attributes are atomic.

- EVENTDEMOGRAPHICS is in 2NF since it is in 1NF and it does not contain non-prime attributes that can violate the form.
- EVENTDEMOGRAPHICS is in 3NF since it is in 2NF and it does not contain non-prime attributes that can violate the form.

Final Normalized Schema



Database Schema

Following the structure indicated in our Relational Diagram, every table can be created with the following commands:

```
CREATE TABLE USER
( UserID INT NOT NULL, Name VARCHAR(15), Email VARCHAR(40), Password VARCHAR
(16), Gender VARCHAR (6) , BackgroundCheck (10), PRIMARY KEY(UserID));

CREATE TABLE VOLUNTEER
(VolunteerID INT NOT NULL, VMajor VARCHAR(20), VClassification VARCHAR(15), EventID
INT, PRIMARY KEY(VolunteerID) );
>ALTER TABLE VOLUNTEER ADD FOREIGN KEY (VolunteerID) REFERENCES
USER(UserID);

CREATE TABLE EVENTORGANIZER
(EventOrganizerID INT NOT NULL, Organization VARCHAR(20), EventID INT, VolunteerID INT,
PRIMARY KEY(EventOrganizerID) );
>ALTER TABLE EVENTORGANIZER ADD FOREIGN KEY(EventOrganizerID) REFERENCES
VOLUNTEER(VolunteerID);
>ALTER TABLE EVENTORGANIZER ADD FOREIGN KEY(EventID) REFERENCES
EVENT(EventID);

CREATE TABLE ADMINISTRATOR
(AdminID INT NOT NULL, AName VARCHAR(20), InventoryID INT, PRIMARY KEY(AdminID));
>ALTER TABLE ADMINISTRATOR ADD FOREIGN KEY (InventoryID) REFERENCES
INVENTORY(InventoryID);

CREATE TABLE VOLUNTEERREPORT
( Hours INT , VolunteerID INT NOT NULL, PRIMARY KEY(VolunteerID)  FOREIGN KEY
(VolunteerID) REFERENCES VOLUNTEER(UserID));
>ALTER TABLE VOLUNTEER REPORT ADD FOREIGN KEY (VolunteerID) REFERENCES
VOLUNTEER(VolunteerID);

CREATE TABLE EVENT
( EventID INT NOT NULL, Name VARCHAR(25), Location VARCHAR(25),PRIMARY
KEY(EventID));

CREATE TABLE INVENTORY
(InventoryID INT NOT NULL, EventID INT, PRIMARY KEY(InventoryID) );
```

```
>ALTER TABLE INVENTORY ADD FOREIGN KEY (EventID) REFERENCES EVENT
(EventID)
CREATE TABLE ITEM
(ItemID INT NOT NULL, Count INT, CheckOut VARCHAR(8), Description VARCHAR(30),
ReturnDate VARCHAR(8), InventoryID INT, PRIMARY KEY(ItemID);
>ALTER TABLE ITEM ADD FOREIGN KEY (InventoryID) REFERENCES
INVENTORY(InventoryID);

CREATE TABLE SPECIALREQUIRMENTS
(UserID INT NOT NULL, SpecialRequirements VARCHAR(50) NOT NULL, PRIMARY
KEY(UserID, SpecialRequirments));
>ALTER TABLE SPECIALREQUIREMENTS ADD FOREIGN KEY (UserID) REFERENCES
USER(UserID);

CREATE TABLE VOLUNTEERSKILLS
(VolunteerID INT NOT NULL, VolunteerSkills, PRIMARY KEY (VolunteerID) );
>ALTER TABLE VOLUNTEERSKILLS ADD FOREIGN KEY (VolunteerID) REFERENCES
VOLUNTEER(VolunteerID);

CREATE TABLE VOLUNTEERREPORTPHOTOS
(VRPhotoID INT NOT NULL, VRPhoto BLOB, VolunteerID INT NOT NULL, PRIMARY KEY
(VRPhoto, VolunteerID));
>ALTER TABLE VOLUNTEERREPORTPHOTOS ADD FOREIGN KEY (VolunteerID)
REFERENCES volunteer(VolunteerID);

CREATE TABLE EVENTACTIVITIES
(Activities VARCHAR(50) NOT NULL, EventID INT NOT NULL, PRIMARY KEY (Activities,
EventID));
>ALTER TABLE EVENTACTIVITIES ADD FOREIGN KEY (EventID) REFERENCES EVENT
(EventID);

CREATE TABLE EVENTPRODUCTLIST
(ProductList VARCHAR(30) NOT NULL, EventID INT NOT NULL, PRIMARY KEY (ProductList,
EventID));
>ALTER TABLE EVENTPRODUCTLIST ADD FOREIGN KEY (EventID) REFERENCES
EVENT (EventID);

CREATE TABLE EVENTPHOTOS
(EPhotoID int, EPhoto BLOB NOT NULL, EventID INT NOT NULL, PRIMARY KEY (EPhotoID,
EventID));
>ALTER TABLE EVENTPHOTOS ADD FOREIGN KEY (EventID) REFERENCES EVENT
(EventID);
```

```
CREATE TABLE EVENTREQUIREMENTS
(Requirements VARCHAR(40) NOT NULL, EventID INT NOT NULL, PRIMARY KEY
(Requirements, EventID));
>ALTER TABLE EVENTREQUIREMENTS ADD FOREIGN KEY (EventID) REFERENCES
EVENT (EventID);

CREATE TABLE EVENTDEMOGRAPHICS
(Demographics VARCHAR(15) NOT NULL, EventID INT NOT NULL, PRIMARY KEY
(Demographics, EventID) );
>ALTER TABLE EVENTDEMOGRAPHICS ADD FOREIGN KEY (EventID) REFERENCES
EVENT(EventID);
```

Database records

The following are examples of possible fields to be contained in our database and their respective commands.

USER

UserID	Name	Email	Password	Gender	Background Check
1234	Lorenzo Torrez	LorenzoTorres@gmail.com	chaparro15	Male	Complete
1235	Shelby Diaz	ShelbyDiaz@Outlook.com	sunflowers333	Female	Complete
1236	Aileen Diaz	AileenDiaz@gmail.com	orwich959	Female	Complete
1237	Carlos Rosas	CarlosRosas@gmail.com	Patches915	Male	Complete
1240.	Fernanda Fiscal	ffiscal@utep.edu	fiscal599	Female	Complete
1241	Marjorie Maldonado	mmaldonado@utep.edu	maldonado699	Female	Complete
1242	Daniel Mejia	dmejia@utep.edu	mejia234	Male	Complete
1250	Olac Fuentes	ofuentes@utep.edu	fuentes12345	Male	Complete

1251	Julio Urenda	jurenda@utep.edu	urenda890	Male	Complete
1252	Steven Roach	sroach@utep.edu	roach567	Male	Complete

INSERT INTO USER VALUES('1234', 'Lorenzo Torrez', 'lorenzoTorrez@gmail.com', 'chaparro15', 'Male', 'Complete');
INSERT INTO USER (UserID, Email, Name , Password , Gender, BackgroundCheck) VALUES (‘1235’, ‘Shelby Diaz’ , ‘ ShelbyDiaz@Outlook.com‘, ‘sunflowers333’, ‘Female’,’Complete’);
INSERT INTO USER (UserID, Name , Email, Password , Gender, BackgroundCheck) VALUES (‘1236’, ‘Aileen Diaz’, ‘AileenDiaz@gmail.com ‘, ‘orwich959’, ‘Female’, Complete);
INSERT INTO USER (UserID, Name, Email , Password , Gender, BackgroundCheck) VALUES (‘1236’, ‘Carlos Rosas’, ‘CarlosRosas@gmail.com ‘, ‘Patches915’, ‘Male’, Complete);
INSERT INTO USER VALUES ('1240', 'Fernanda Fis', 'ffiscal@utep.edu', 'fiscal599' ,'Female','Complete');
INSERT INTO USER VALUES ('1241', 'Marjorie Maldo', 'mmaldonado@utep.edu', 'maldonado699' ,'Female','Complete');
INSERT INTO USER VALUES ('1242', 'Daniel Mejia', 'mmejia@utep.edu', 'mejia234' ,'Male','Complete');
INSERT INTO USER VALUES ('1250', 'Olac Fuentes', 'ofuentes@utep.edu', ' fuentes 12345', 'Male',   'Complete');
INSERT INTO USER VALUES ('1251', 'Julio Urenda' , jurenda@utep.edu', 'urenda890', 'Male', 'Complete');
INSERT INTO USER VALUES('1252','Steven Roach', 'sroach@utep.edu','roach567','Male','Complete');

VOLUNTEER

VolunteerID	VMajor	VClassification	EventID
1234	CS	Freshman	4321
1235	CS	Sophomore	4321
1236	CS	Freshman	4321

INSERT INTO VOLUNTEER VALUES ('1234','CS','Freshman', '4321');
INSERT INTO VOLUNTEER VALUES ('1235','CS','Sophomore', '4321');
INSERT INTO VOLUNTEER VALUES ('1236','CS','Freshman', '4321');

EVENTORGANIZER

EventOrganizerID	Organization	EventID
1240	UTEP	4321
1241	UTEP	4322
1242	UTEP	4323

```
INSERT INTO EVENTORGANIZER (OrganizerID, EOrganization, EventID)
VALUES (‘1240’, ‘UTEP’, ‘4321’);
INSERT INTO EVENTORGANIZER (OrganizerID, EOrganization, EventID)
VALUES (‘1241’, ‘UTEP’, ‘4322’);
INSERT INTO EVENTORGANIZER (OrganizerID, EOrganization, EventID)
VALUES (‘12342’, ‘UTEP’ , ‘4323’);
```

ADMINISTRATOR

AdminID	InventoryID
1250	15001
1251	15002
1252	15003

```
INSERT INTO ADMINISTRATOR (AdminID, InventoryID)
VALUES (‘ 1250 ‘ , ‘ 15001  ‘);
INSERT INTO ADMINISTRATOR (AdminID, InventoryID)
VALUES (‘ 1251 ‘ , ‘  15002 ‘);
INSERT INTO ADMINISTRATOR (AdminID, InventoryID)
VALUES (‘ 12352 ‘, ‘  15003 ‘);
```

VOLUNTEERREPORT

Hours	VolunteerID
20	1234
15	1235
16	1236

```
INSERT INTO VOLUNTEER REPORT (Hours, VolunteerID)
VALUES (‘ 20 ‘ , ‘  1234 ‘ ) ;
INSERT INTO VOLUNTEER REPORT (Hours, VolunteerID)
VALUES (‘ 15 ‘, ‘ 1235  ‘ ) ;
INSERT INTO VOLUNTEER REPORT (Hours, VolunteerID)
VALUES (‘ 16 ‘, ‘ 1236  ‘ ) ;
```

EVENT

EventID	Name	Location
4321	Minerpalooza	SunBowl
4322	TacoTuesday	El Paso Center
4323	MoviesOnTheLawn	Centennial Plaza

```
INSERT INTO  ( EventID, Name, Location)
VALUES ( ‘ 4321 ‘, ‘ Minerpalooza ‘, ‘ SunBowl ‘  ) ;
INSERT INTO  ( EventID, Name, Location)
VALUES ( ‘ 4322 ‘, ‘ TacoTuesday ‘, ‘ El Paso Center ‘  ) ;
INSERT INTO  ( EventID, Name, Location)
VALUES ( ‘ 4323 ‘, ‘ MoviesOnTheLawn ‘, ‘ Centennial Plaza ‘  ) ;
```

INVENTORY

InventoryID	EventID
15001	4321
15002	4322
15003	4323

```
INSERT INTO  ( InventoryID, EventID)
VALUES ( ‘ 15001 ‘, ‘  4321 ‘  ) ;
INSERT INTO  ( InventoryID, EventID)
VALUES ( ‘ 15002‘, ‘ 4322  ‘  ) ;
INSERT INTO  ( InventoryID, EventID)
VALUES ( ‘ 15003 ‘, ‘ 4323  ‘  ) ;
```

ITEM

ItemID	Count	CheckOut	Description	ReturnDate	InventoryID
789	5	none	Chair	none	15001
789	2	4/5/20	Table	4/7/20	15002
789	8	6/6/19	Umbrella	6/8/19	15003

```
INSERT INTO  ( Count, CheckedOut, Description, ReturnDate, InventoryID )
VALUES ( ‘ 5 ‘ ,‘none  ‘ , ‘ Chair ‘  , ‘ none ‘ , ‘ 15001 ‘  ) ;
INSERT INTO  ( Count, CheckedOut, Description, ReturnDate, InventoryID )
VALUES ( ‘2  ‘ ,‘ 4/5/20 ‘ , ‘ Table ‘  , ‘ 4/7/20 ‘ , ‘ 15002 ‘  ) ;
INSERT INTO  ( Count, CheckedOut, Description, ReturnDate, InventoryID )
VALUES ( ‘ 8 ‘ ,‘ 6/6/19 ‘ , ‘ Umbrella ‘  , ‘ 6/8/19 ‘ , ‘ 15003 ‘  ) ;
```

SPECIALREQUIREMENTS

UserID	SpecialRequirements
1234	None
1235	Glutten Free
1236	Ramp

```
INSERT INTO SPECIALREQUIREMENTS  (UserID, SpecialRequirements )
VALUES ( ‘ 1234 ‘ ,  ‘  None ‘ ) ;
INSERT INTO  SPECIALREQUIREMENTS (UserID, SpecialRequirements )
VALUES ( ‘ 1235 ‘ ,  ‘  Glutten Free ‘ ) ;
INSERT INTO  SPECIALREQUIREMENTS(UserID, SpecialRequirements )
VALUES ( ‘ 1236 ‘ ,  ‘ Ramp  ‘ ) ;
```

VOLUNTEERSKILLS

VolunteerID	VolunteerSkills
1234	Writing
1235	C Programming
1236	Java Programing

```
INSERT INTO  VOLUNTEERSKILLS (VolunteerID, VolunteerSKills )
VALUES ( ‘ 1234‘ , ‘ Writing ‘  ) ;
INSERT INTO  VOLUNTEERSKILLS (VolunteerID, VolunteerSKills )
VALUES ( ‘ 1235 ‘ , ‘ C Programming ‘  ) ;
INSERT INTO  VOLUNTEERSKILLS (VolunteerID, VolunteerSKills )
VALUES ( ‘ 1236 ‘ , ‘ Java Programing ‘  ) ;
```

VOLUNTEERREPORTPHOTOS

VRPhotoID	VRPhotos	VolunteerID
1101	BLOB	1234
1102	BLOB	1234
1103	BLOB	1235

```
INSERT INTO volunteerreportphotos VALUES('1101',
LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\p1.jpg'), '1234');
INSERT INTO volunteerreportphotos VALUES('1102',
LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\p2.jpg'), '1234');
INSERT INTO volunteerreportphotos VALUES('1103',
LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\p2.jpg'), '1235');
```

EVENTACTIVITIES

Activities	EventID
Screening Movie	4323
Handing out Flyers	4323
Cleaning lawn	4323

INSERT INTO EVENTACTIVITIES(Activities, EventID)
VALUES (‘ Screening Movie‘ , ‘4323 ‘) ;
INSERT INTO EVENTACTIVITIES (Activities, EventID)
VALUES (‘ Handing out Flyers ‘ , ‘ 4323‘) ;
INSERT INTO EVENTACTIVITIES (Activities, EventID)
VALUES (‘ Cleaning Lawn ‘ , ‘ 4323‘) ;

EVENTPRODUCTLIST

ProductList	EventID
Camera	4323
IPad	4323
Chair	4323

INSERT INTO EVENTPRODUCTLIST (ProductList, EventID)
VALUES (‘ Camera ‘ , ‘ 4323 ‘) ;
INSERT INTO EVENTPRODUCTLIST (ProductList, EventID)
VALUES (‘IPad ‘ , ‘ 4323 ‘) ;
INSERT INTO EVENTPRODUCTLIST (ProductList, EventID)
VALUES (‘ Chair ‘ , ‘ 4323 ‘) ;

EVENTPHOTOS

EPhotoID	EPhotos	EventID
0001	blob	4321
0002	blob	4321
0003	blob	4321

INSERT INTO eventphotos VALUES ('0001',
LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\images.jpg'),
'4321');

INSERT INTO eventphotos values ('0002',
LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\ue-banner.jpg'),
'4321');
INSERT INTO eventphotos values ('0003',
LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\Purple-Heart-UTEP-celebrate1.jpg'), '4321');

EVENTREQUIREMENTS

Requirements	EventID
Registration	4323
Mobile Phone	4323
TextBook	4323

INSERT INTO EVENTREQUIREMENTS VALUES ('Registration','4323');
INSERT INTO EVENTREQUIREMENTS VALUES ('Mobile Phone','4323');
INSERT INTO EVENTREQUIREMENTS VALUES ('TextBook','4323');

EVENTDEMOGRAPHICS

Demographics	EventID
5xFreshmen	4323
8xMarried	4322
2xOver21	4321

INSERT INTO EVENTDEMOGRAPHICS VALUES(‘5xFreshmen’,’4323’);
INSERT INTO EVENTDEMOGRAPHICS VALUES(‘8xMarried’ , ‘ 4322’);
INSERT INTO EVENTDEMOGRAPHICS VALUES(‘2xOver21’, ‘4321’);

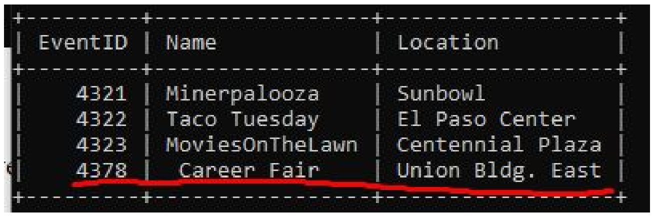
SQL Queries

The following list goes by functional requirements (non-functional requirements are referenced to functional requirements). The commands that meet the functionality are shown under every requirement.

1. **The system shall keep a documentation record of events.**
   Events are recorded by adding them to the events list by the administrators with:

INSERT INTO EVENT VALUES('4378' , ' Career Fair' , 'Union Bldg. East');

Result:



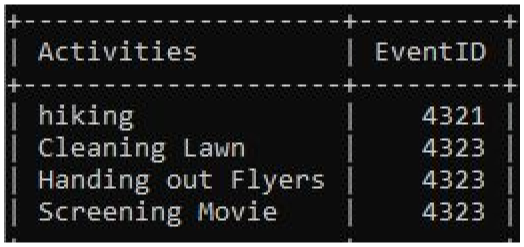
```
+---------+-----------------+------------------+
| EventID | Name            | Location         |
+---------+-----------------+------------------+
|    4321 | Minerpalooza    | Sunbowl          |
|    4322 | Taco Tuesday    | El Paso Center   |
|    4323 | MoviesOnTheLawn | Centennial Plaza |
|    4378 |  Career Fair    | Union Bldg. East |
+---------+-----------------+------------------+
```

2. **The system shall be able to organize and describe individual events.**
   Every event has an ID that can be used to reference other details during the event advertisement such as SPECIALREQUIREMENTS, EVENTACTIVITIES, EVENTPRODUCTLIST, EVENTPHOTOS, EVENTREQUIREMENTS, EVENTDEMOGRAPHICS.

The admin can create these details by using the commands:

INSERT INTO EVENTACTIVITIES VALUES ('hiking', '4321');

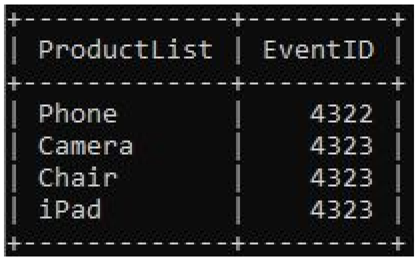
Result:



```
+--------------------+---------+
| Activities         | EventID |
+--------------------+---------+
| hiking             |    4321 |
| Cleaning Lawn      |    4323 |
| Handing out Flyers |    4323 |
| Screening Movie    |    4323 |
```

INSERT INTO EVENTPRODUCTLIST VALUES ('Phone','4322');

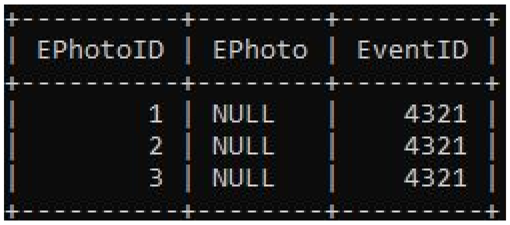
Result:



```
+-------------+---------+
| ProductList | EventID |
+-------------+---------+
| Phone       |    4322 |
| Camera      |    4323 |
| Chair       |    4323 |
| iPad        |    4323 |
+-------------+---------+
```

INSERT INTO eventphotos VALUES ('0001', LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\images.jpg'), '4321');

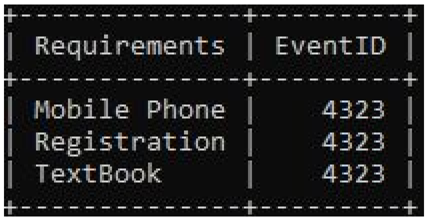
Result:



```
+----------+--------+---------+
| EPhotoID | EPhoto | EventID |
+----------+--------+---------+
|        1 | NULL   |    4321 |
|        2 | NULL   |    4321 |
|        3 | NULL   |    4321 |
+----------+--------+---------+
```

INSERT INTO EVENTREQUIREMENTS VALUES ('Registration','4323');

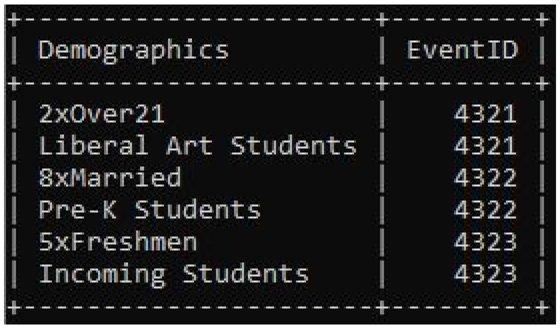
Result:



```
+--------------+---------+
| Requirements | EventID |
+--------------+---------+
| Mobile Phone |    4323 |
| Registration |    4323 |
| TextBook     |    4323 |
+--------------+---------+
```

INSERT INTO EVENTDEMOGRAPHICS VALUES('5xFreshmen','4323');

Result:



```
+------------------------+---------+
| Demographics           | EventID |
+------------------------+---------+
| 2xOver21               |    4321 |
| Liberal Art Students   |    4321 |
| 8xMarried              |    4322 |
| Pre-K Students         |    4322 |
| 5xFreshmen             |    4323 |
| Incoming Students      |    4323 |
+------------------------+---------+
```

And to update these details, the corresponding commands are:

UPDATE EVENTACTIVITIES SET Activities = '*newActivity*' WHERE Activities = '*oldActivity*' ;
Result: old activity is replaced with new activity

UPDATE EVENTPRODUCTLIST SET ProductList = '*newProductItem*' WHERE ProductList = '*oldProductItem*';
Result: old product item is replaced with new product item

UPDATE EVENTPHOTOS SET EPhotos = ' *image* ' WHERE EPhotos = '*oldImage*' *;*
Result: old photo is replaced with new photo

UPDATE EVENTREQUIREMENTS SET Requirement = ' *newEventRequirement* ' WHERE Requirement = '*oldEventRequirement*'*;*
Result: Old event requirement is replaced with new requirement

UPDATE EVENTDEMOGRAPHICS SET Demographics = '*newDemographic*' WHERE Demographics = '*oldDemographic*' ;
Result: Old demographic datum is replaced with new demographic datum.

3. **The system shall allow admins to manage, edit, and delete the events.**

Admins can modify both the list of events and the details in the description of the events by using the commands in requirements 1 and 2. Additionally, individual instances for every event can be deleted using these commands:

DELETE FROM EVENT WHERE EventID = '*anEventID*';
Result: the event is deleted from the main list

DELETE FROM EVENTACTIVITIES WHERE Activities = '*anActivity*' ;
Result: the activity is deleted in the events activities list

DELETE FROM EVENTPRODUCTLIST WHERE ProductList = '*aProductItem*';
Result: the product item is deleted in the event product list

DELETE FROM EVENTPHOTOS WHERE EPhotos = '*anImage*' *;*
Result: the photo is deleted in the event photos list

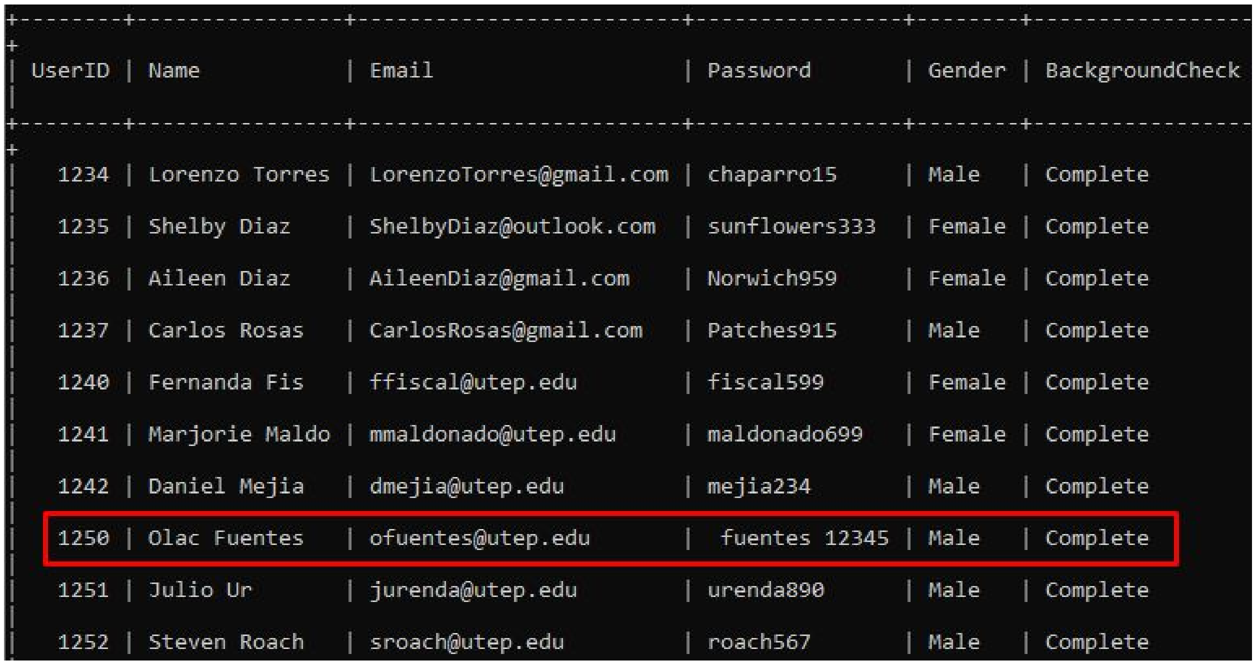
DELETE FROM EVENTREQUIREMENTS WHERE Requirement = '*anEventRequirement*'*;*
Result: An event requirement is deleted from the event requirements list

DELETE FROM EVENTDEMOGRAPHICS WHERE Demographics = *'aDemographicDatum'* ;
Result: the demographic datum is deleted from the list.

4. **The system shall allow admins to sign in and manage events**

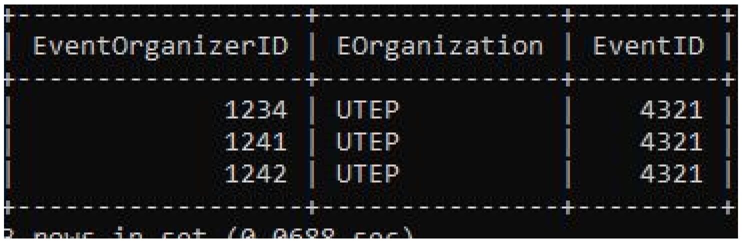
Admins can be created by first signing them up as users whose information is stored using this command:
INSERT INTO USER VALUES ('1250', 'Olac Fuentes', 'ofuentes@utep.edu', ' fuentes 12345', 'Male', 'Complete');

Result:



```
+--------+---------------+-----------------------+---------------+--------+-----------------
+
| UserID | Name          | Email                 | Password      | Gender | BackgroundCheck
|
+--------+---------------+-----------------------+---------------+--------+-----------------
+
|   1234 | Lorenzo Torres | LorenzoTorres@gmail.com | chaparro15   | Male   | Complete
|
|   1235 | Shelby Diaz   | ShelbyDiaz@outlook.com | sunflowers333 | Female | Complete
|
|   1236 | Aileen Diaz   | AileenDiaz@gmail.com  | Norwich959    | Female | Complete
|
|   1237 | Carlos Rosas  | CarlosRosas@gmail.com | Patches915    | Male   | Complete
|
|   1240 | Fernanda Fis  | ffiscal@utep.edu      | fiscal599     | Female | Complete
|
|   1241 | Marjorie Maldo | mmaldonado@utep.edu  | maldonado699  | Female | Complete
|
|   1242 | Daniel Mejia  | dmejia@utep.edu       | mejia234      | Male   | Complete
|
|   1250 | Olac Fuentes  | ofuentes@utep.edu     |  fuentes 12345 | Male  | Complete
|
|   1251 | Julio Ur      | jurenda@utep.edu      | urenda890     | Male   | Complete
|
|   1252 | Steven Roach  | sroach@utep.edu       | roach567      | Male   | Complete
```

With a user account, an administrator or events, also called an 'Event Organizer' can then be created with the command:
INSERT INTO EVENTORGANIZER (AdminID, InventoryID)
VALUES (' 1250 ' , ' 15001 ');
Result:

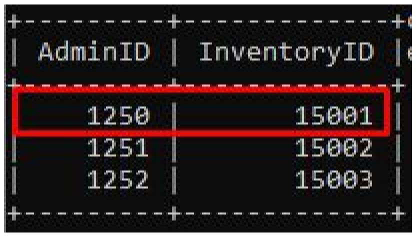


```
+------------------+---------------+---------+
| EventOrganizerID | EOrganization | EventID |
+------------------+---------------+---------+
|             1234 | UTEP          |    4321 |
|             1241 | UTEP          |    4321 |
|             1242 | UTEP          |    4321 |
+------------------+---------------+---------+
3 rows in set (0.0688 sec)
```

If the admin is in charge of any set of inventory, they can be added to the list:

INSERT INTO ADMINISTRATOR (AdminID, InventoryID)
VALUES (‘ 1250 ‘ , ‘ 15001 ‘);

Result:



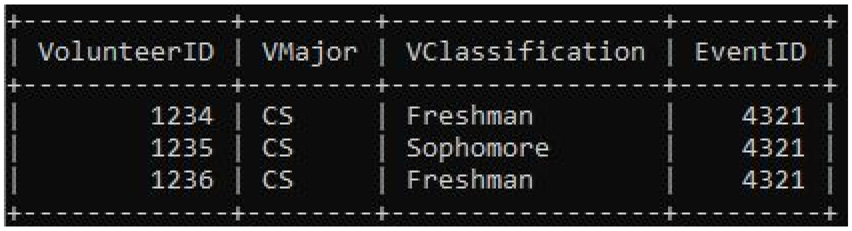
```
+---------+-------------+
| AdminID | InventoryID |
+---------+-------------+
|    1250 |       15001 |
|    1251 |       15002 |
|    1252 |       15003 |
+---------+-------------+
```

5. **The system shall allow students to sign up as members for specific events.**

Similar to requirement 4, students must create a user account, and then their volunteer information is stored using:

INSERT INTO VOLUNTEER VALUES ('1234','CS','Freshman', '4321');

Result:



```
+-------------+--------+-----------------+---------+
| VolunteerID | VMajor | VClassification | EventID |
+-------------+--------+-----------------+---------+
|        1234 | CS     | Freshman        |    4321 |
|        1235 | CS     | Sophomore       |    4321 |
|        1236 | CS     | Freshman        |    4321 |
+-------------+--------+-----------------+---------+
```

6. **The system shall be able to capture optional or required information of the event members like classification, ID numbers(same as volunteerID), Major, etc.**

   Refer to requirement 5 which includes the fields of classification, ID, and major as well.

7. **The system shall be able to annotate the requirements for the event attendees such as writing material, equipment, or other types of required equipment.**

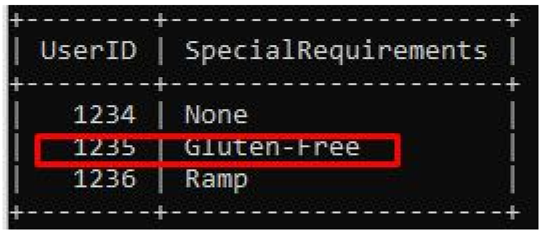
Refer to requirements 2 and 3 which indicate the administrator can add and edit these details through the tables of Event Product list and Event Requirements.

8. **The system shall be able to capture special requirements from the attendees (e.g. wheelchair, gluten-free, etc).**

   Participants can indicate their personal requirements

INSERT INTO (UserID, SpecialRequirements)
VALUES (‘ 1235 ‘ , ‘ Glutten Free ‘) ;

Result:

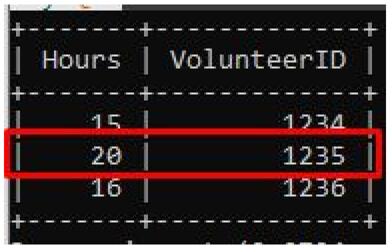


```
+--------+----------------------+
| UserID | SpecialRequirements  |
+--------+----------------------+
|   1234 | None                 |
|   1235 | Gluten-Free          |
|   1236 | Ramp                 |
+--------+----------------------+
```

9. **The system shall allow the volunteers to report their attendance and participation hours.**
   The interface allows volunteers to submit a form to report their participation hours. The form contains the volunteerID, and number of hours and is sent to the event organizer. Then the organizer can approve those hours, see requirement 10.
10. **The system shall allow the admins to approve the member attendance reports based on their participation hours.**
    Upon receiving individual form request from volunteers, the event organizer can approve the hours reported by the volunteer and then query the following command:

```
INSERT INTO VOLUNTEER REPORT (Hours, VolunteerID)
VALUES (‘ 20 ‘ , ‘  1234 ‘ ) ;
```

Result:



```
+-------+-------------+
| Hours | VolunteerID |
+-------+-------------+
|    15 |        1234 |
|    20 |        1235 |
|    16 |        1236 |
+-------+-------------+
```

11. **The system shall keep track of all hours reported by the members.**
    Accomplished by requirement 10.

12. **The system shall keep a record of activities realized at the event or any worthwhile incident.**
    This functionality is also accomplished by enabling volunteers to report to the administrator who can employ the queries in requirement 3 to annotate activities or special notes.

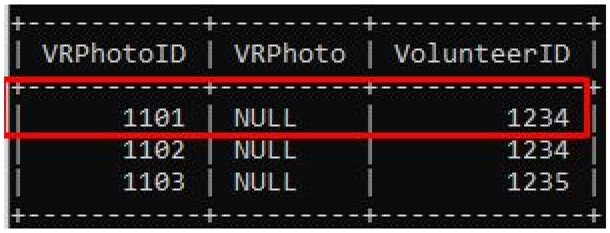
13. **The system shall be able to create and distribute a log of pictures taken during the event**
    In addition to the event pictures allocated to every event, see requirement 3, and requirement 14

14. The system shall allow the event members to upload pictures to their respective events

Volunteers can also upload their own pictures with the query:

INSERT INTO volunteerreportphotos VALUES('1101', LOAD_FILE('C:\\Users\\leo\\Documents\\Assignments\\databases\\imgsampler\\p1.jpg'), '1234');

Result:



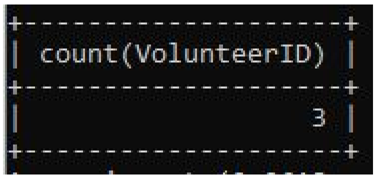
```
+-----------+---------+-------------+
| VRPhotoID | VRPhoto | VolunteerID |
+-----------+---------+-------------+
|      1101 | NULL    |        1234 |
|      1102 | NULL    |        1234 |
|      1103 | NULL    |        1235 |
+-----------+---------+-------------+
```

15. The system shall keep track of the total number of attendees for every event and be able to deliver reports of attendance.

The number of attendees is extracted from the volunteers allocated to an event, which can be used to return an attendance count. Number of attendees can be obtained by executing the following command:

SELECT COUNT(VolunteerID) FROM VOLUNTEER WHERE EventID = *'relevantEventID'*;

Result:



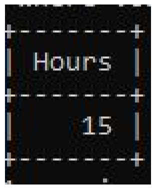
```
+--------------------+
| count(VolunteerID) |
+--------------------+
|                  3 |
+--------------------+
```

16. The system must keep track of where the impact is being made the most, and what areas are being visited the most.

Impact can be measured by the total participation (requirement 15), time spent (volunteer hours), as well as the number of activities dedicated to a given event. Volunteer hours and number of activities per event can be obtained using:

SELECT Hours FROM VOLUNTEERREPORT WHERE VolunteerID='1234';

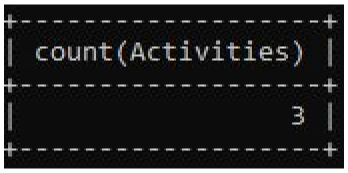
Result:



```
+-------+
| Hours |
+-------+
|    15 |
+-------+
```

SELECT COUNT(Activities DISTINCT) FROM EVENTACTIVITIES WHERE EventID = '4323';

Result:



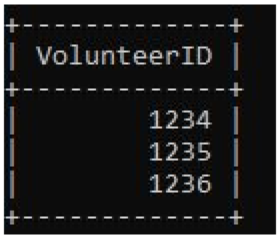
```
+-------------------+
| count(Activities) |
+-------------------+
|                 3 |
+-------------------+
```

17. The system will advertise to students about future events, for students or schools being outreached too.

When events are added ahead of time, the application can generate an invitation for volunteers signed up for that event.

SELECT DISTINCT VolunteerID FROM VOLUNTEER WHERE EventID= *'someFutureEvent';*

Result:



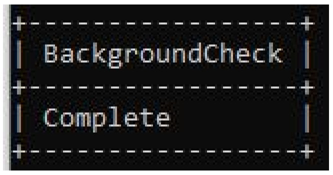
```
+-------------+
| VolunteerID |
+-------------+
|        1234 |
|        1235 |
|        1236 |
+-------------+
```

18. The system will also notify the users whether they require a background check

The application can check the status of the Background Check field in USER and let the Event Organizer determine if the volunteer will need a background check. The background check status can be obtained by:

SELECT BackgroundCheck FROM USER WHERE UserID = '*1234'* ;

Result:



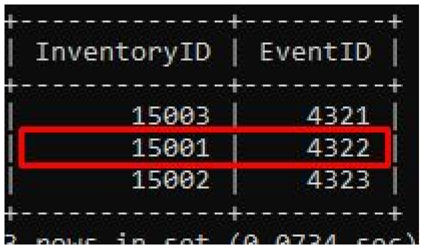
```
+-----------------+
| BackgroundCheck |
+-----------------+
| Complete        |
+-----------------+
```

19. The system shall provide the option to allocate an inventory with items and their description

Items can be allocated to an event using the commands from requirement 3. In addition, the inventory administrator can add and manage inventory items with the following queries:

INSERT INTO (InventoryID, EventID) VALUES (' 15001 ', ' 4321 ') ;

Result:

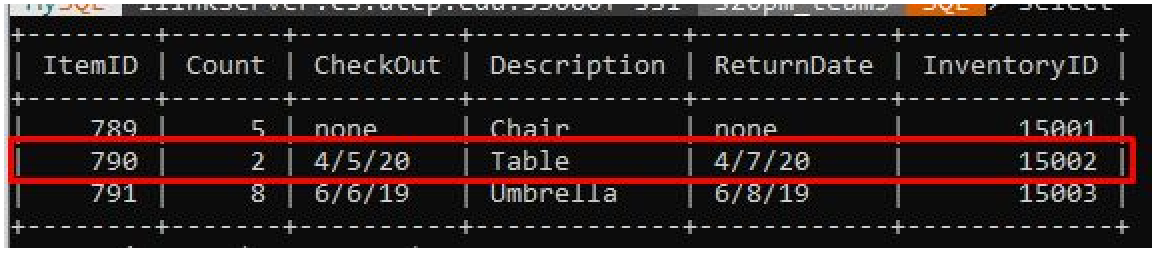


```
+-------------+---------+
| InventoryID | EventID |
+-------------+---------+
|       15003 |    4321 |
|       15001 |    4322 |
|       15002 |    4323 |
+-------------+---------+
3 rows in set (0.0734 sec)
```

UPDATE INVENTORY SET ItemID= ‘ *someInventory’* WHERE ItemID = ‘*newInventory’*;
Result: old inventory set is replaced with new one.

DELETE INVENTORY WHERE ItemID = ‘*someInventory’*
Result: the specified inventory is deleted.

Similarly, inventory item can be created, updated, and deleted:
INSERT INTO (Count, CheckedOut, Description, ReturnDate, InventoryID)
VALUES (‘2 ‘ ,‘ 4/5/20 ‘ , ‘ Table ‘ , ‘ 4/7/20 ‘ , ‘ 15002 ‘) ;
Result:



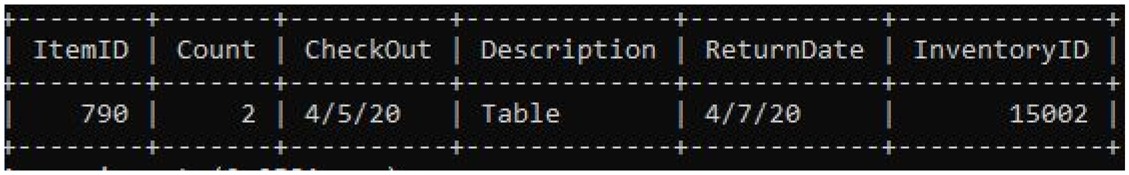
```
+--------+-------+----------+-------------+------------+-------------+
| ItemID | Count | CheckOut | Description | ReturnDate | InventoryID |
+--------+-------+----------+-------------+------------+-------------+
|    789 |     5 | none     | Chair       | none       |       15001 |
|    790 |     2 | 4/5/20   | Table       | 4/7/20     |       15002 |
|    791 |     8 | 6/6/19   | Umbrella    | 6/8/19     |       15003 |
+--------+-------+----------+-------------+------------+-------------+
```

UPDATE ITEM SET Count = *‘NewCount’* , CheckedOut = *‘dateIfCheckedOut’*, Description = *‘descriptionIfNewDescription’*, ReturnDate = ‘*returnDateIfReturningItem’* WHERE ItemID = *‘itemID’* ;
Result: Changes the item data per request.

DELETE ITEM WHERE ItemID = *‘itemID’* ;
Result: Deletes specified item.

20. **The system shall be able to verify the flow of items, whether the items have been checked in or out from the inventory as well as any important description (e.g. damage or returning date).**
Inventory items contain the current status of the item and a description of their current condition. It can be retrieved from the inventory list by using:

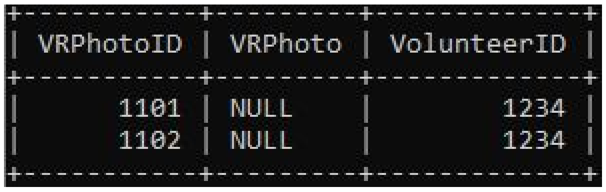
SELECT * FROM ITEM WHERE ItemID ='790';
Result:



```
+--------+-------+----------+-------------+------------+-------------+
| ItemID | Count | CheckOut | Description | ReturnDate | InventoryID |
+--------+-------+----------+-------------+------------+-------------+
|    790 |     2 | 4/5/20   | Table       | 4/7/20     |       15002 |
+--------+-------+----------+-------------+------------+-------------+
```

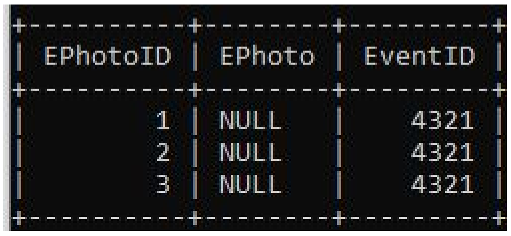
21. The system shall have a report function executed by admins which lets them obtain a report by event.

In addition, an actual report filled through the application by printing the total number of attendees (requirements 15, 16, 20), their respective hours, photos from the event, activities, products or equipment utilized. For pictures retrieval we use the following query:
SELECT * FROM VOLUNTEERREPORTPHOTOS WHERE VolunteerID = '*1234';*
Result:



```
+-----------+---------+-------------+
| VRPhotoID | VRPhoto | VolunteerID |
+-----------+---------+-------------+
|      1101 | NULL    |        1234 |
|      1102 | NULL    |        1234 |
+-----------+---------+-------------+
```

SELECT * FROM EVENTPHOTOS WHERE EventID = '*4321';*
Result:



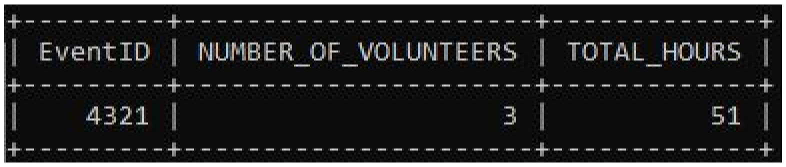
```
+----------+--------+---------+
| EPhotoID | EPhoto | EventID |
+----------+--------+---------+
|        1 | NULL   |    4321 |
|        2 | NULL   |    4321 |
|        3 | NULL   |    4321 |
+----------+--------+---------+
```

Views

1. A view of the number of volunteers and their accrued hours that displays by distinct event

```
CREATE VIEW event_totals AS
SELECT EventID, SUM(Hours) AS COMPLETED_HOURS,
COUNT(VOLUNTEER.VolunteerID) AS NUMBER_OF_VOLUNTEERS FROM
VOLUNTEER INNER JOIN VOLUNTEERREPORT ON
VOLUNTEER.VolunteerID = VOLUNTEERREPORT.VolunteerID
GROUP BY EventID;
```

Result:

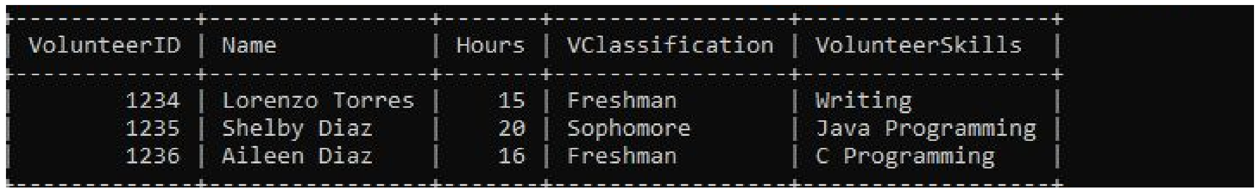


```
+---------+----------------------+-------------+
| EventID | NUMBER_OF_VOLUNTEERS | TOTAL_HOURS |
+---------+----------------------+-------------+
|    4321 |                    3 |          51 |
+---------+----------------------+-------------+
```

2. A view that shows a list of volunteers in a specific event and their information.

```
CREATE VIEW ReportOf4321 AS
SELECT VOLUNTEER.VolunteerID, USER.Name,VOLUNTEERREPORT.Hours,
VOLUNTEER.VClassification, VOLUNTEERSKILLS.VolunteerSkills FROM
( ( (USER INNER JOIN VOLUNTEER ON USER.UserID = VOLUNTEER.VolunteerID )
INNER JOIN VOLUNTEERREPORT ON USER.UserID = VOLUNTEERREPORT.VolunteerID)
INNER JOIN VOLUNTEERSKILLS ON USER.UserID = VOLUNTEERSKILLS.VolunteerID)
WHERE VOLUNTEER.EventID = ' 4321';
```

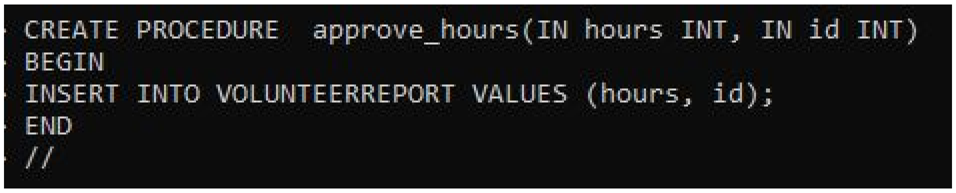
Result:



```
+-------------+----------------+-------+-----------------+------------------+
| VolunteerID | Name           | Hours | VClassification | VolunteerSkills  |
+-------------+----------------+-------+-----------------+------------------+
|        1234 | Lorenzo Torres |    15 | Freshman        | Writing          |
|        1235 | Shelby Diaz    |    20 | Sophomore       | Java Programming |
|        1236 | Aileen Diaz    |    16 | Freshman        | C Programming    |
+-------------+----------------+-------+-----------------+------------------+
```

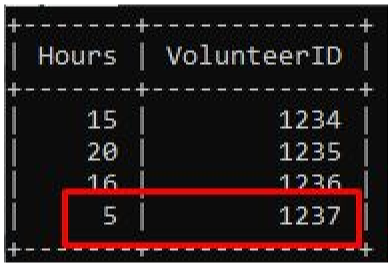
Procedures, Triggers

1. This procedure can be used to approve hours by the administrator



```
CREATE PROCEDURE  approve_hours(IN hours INT, IN id INT)
BEGIN
INSERT INTO VOLUNTEERREPORT VALUES (hours, id);
END
//
```

Result:

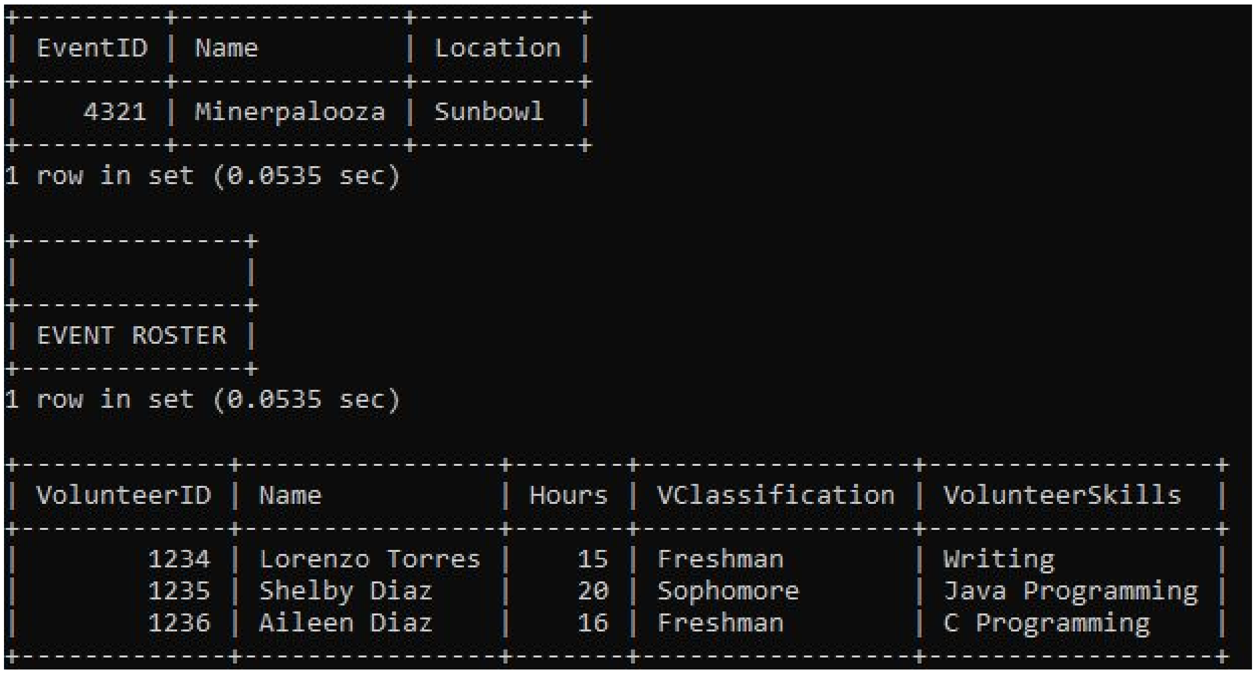


```
+-------+-------------+
| Hours | VolunteerID |
+-------+-------------+
|    15 |        1234 |
|    20 |        1235 |
|    16 |        1236 |
|     5 |        1237 |
+-------+-------------+
```

2. For the last view that was created with the same model of the view for rosters but with an input parameter and the tittle of the event

```
CREATE PROCEDURE GETREPORTBYT (IN eventn INT)
BEGIN SELECT * FROM EVENT WHERE EventID = eventn;
SELECT  'EVENT ROSTER' AS '';
SELECT VOLUNTEER.VolunteerID, USER>Name, VOLUNTEERREPORT.Hours,
VOLUNTEER.VClassification, VOLUNTEERSKILLS.VolunteerSkills FROM
(((USER INNER JOIN VOLUNTEER ON USER.UserID = VOLUNTEER.VolunteerID)
INNER JOIN VOLUNTEERREPORT ON USER.UserID=VOLUNTEERREPORT.VolunteerID)
INNER JOIN VOLUNTEERSKILLS ON USER.UserID = VOLUNTEERSKILLS.VolunteerID)
WHERE VOLUNTEER.EventID = eventn;
END//
```

Result:

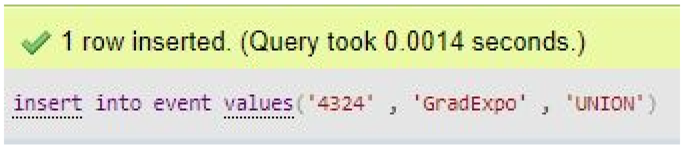


```
+---------+-------------+----------+
| EventID | Name        | Location |
+---------+-------------+----------+
|    4321 | Minerpalooza | Sunbowl  |
+---------+-------------+----------+
1 row in set (0.0535 sec)

+--------------+
|              |
+--------------+
| EVENT ROSTER |
+--------------+
1 row in set (0.0535 sec)

+-------------+----------------+-------+-----------------+------------------+
| VolunteerID | Name           | Hours | VClassification | VolunteerSkills  |
+-------------+----------------+-------+-----------------+------------------+
|        1234 | Lorenzo Torres |    15 | Freshman        | Writing          |
|        1235 | Shelby Diaz    |    20 | Sophomore       | Java Programming |
|        1236 | Aileen Diaz    |    16 | Freshman        | C Programming    |
+-------------+----------------+-------+-----------------+------------------+
```

1. A trigger that changes event names to all capitals

```
CREATE TRIGGER tr_insert_event
BEFORE INSERT ON EVENT
FOR EACH ROW
SET NEW.Name = UPPER(NEW.Name);
```

Result

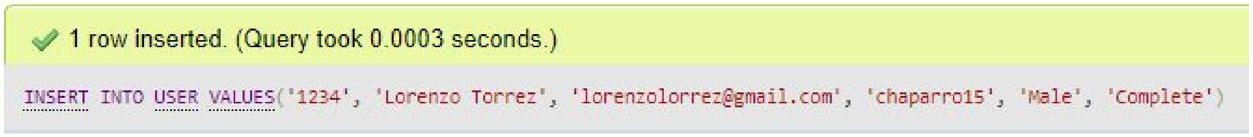


```
1 row inserted. (Query took 0.0014 seconds.)
insert into event values('4324' , 'GradExpo' , 'UNION')
```

2. Conversely, another trigger that converts all letters from

```
CREATE TRIGGER tr2_insert_event
BEFORE INSERT ON USER
FOR EACH ROW
SET NEW.Email = LOWER(NEW.Email);
```

Result:



```
1 row inserted. (Query took 0.0003 seconds.)
INSERT INTO USER VALUES('1234', 'Lorenzo Torrez', 'lorenzolorrez@gmail.com', 'chaparro15', 'Male', 'Complete')
```

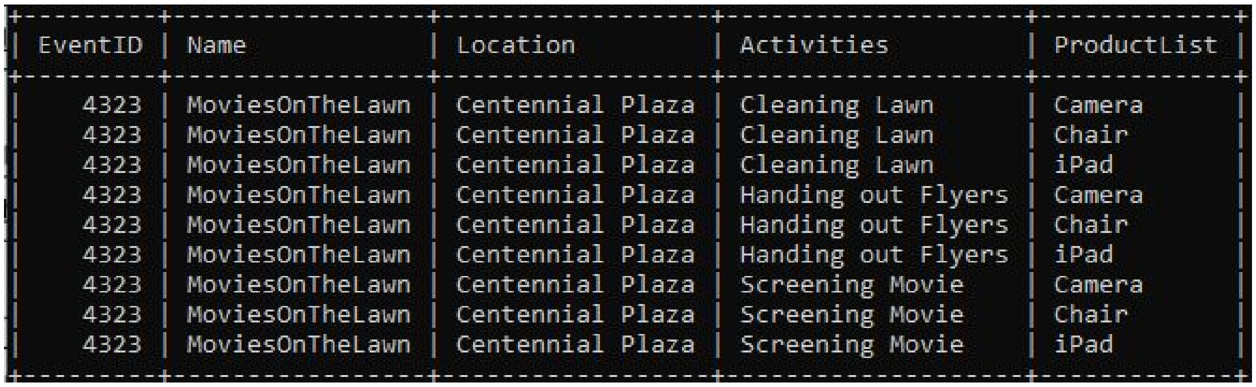
Reports:

The requirements involving some kind of report are presented below with their respective list number, queries, and results.

2. The system shall be able to organize and describe individual events.
3 The system shall allow admins to manage, edit, and delete the events.

```
SELECT EVENT.EventID, EVENT.Name, EVENT.Location, EVENTACTIVITIES.Activities,
EVENTPRODUCTLIST.ProductList  FROM
( (EVENT INNER JOIN EVENTACTIVITIES ON EVENT.EventID =
EVENTACTIVITIES.EventID)
INNER JOIN EVENTPRODUCTLIST ON EVENT.EventID =
EVENTPRODUCTLIST.EventID) ;
```

Result:

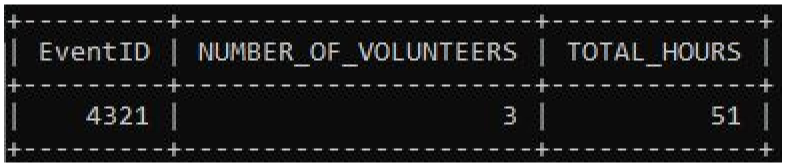


```
+---------+-----------------+-----------------+------------------+-------------+
| EventID | Name            | Location        | Activities       | ProductList |
+---------+-----------------+-----------------+------------------+-------------+
|    4323 | MoviesOnTheLawn | Centennial Plaza | Cleaning Lawn    | Camera      |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Cleaning Lawn    | Chair       |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Cleaning Lawn    | iPad        |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Handing out Flyers | Camera    |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Handing out Flyers | Chair     |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Handing out Flyers | iPad      |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Screening Movie  | Camera      |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Screening Movie  | Chair       |
|    4323 | MoviesOnTheLawn | Centennial Plaza | Screening Movie  | iPad        |
+---------+-----------------+-----------------+------------------+-------------+
```

11. The system shall keep track of all hours reported by the members.
As done in views

```
CREATE VIEW event_totals AS
SELECT EventID, SUM(Hours) AS COMPLETED_HOURS,
COUNT(VOLUNTEER.VolunteerID) AS NUMBER_OF_VOLUNTEERS FROM
VOLUNTEER INNER JOIN VOLUNTEERREPORT ON
VOLUNTEER.VolunteerID = VOLUNTEERREPORT.VolunteerID
GROUP BY EventID;
```

Result:

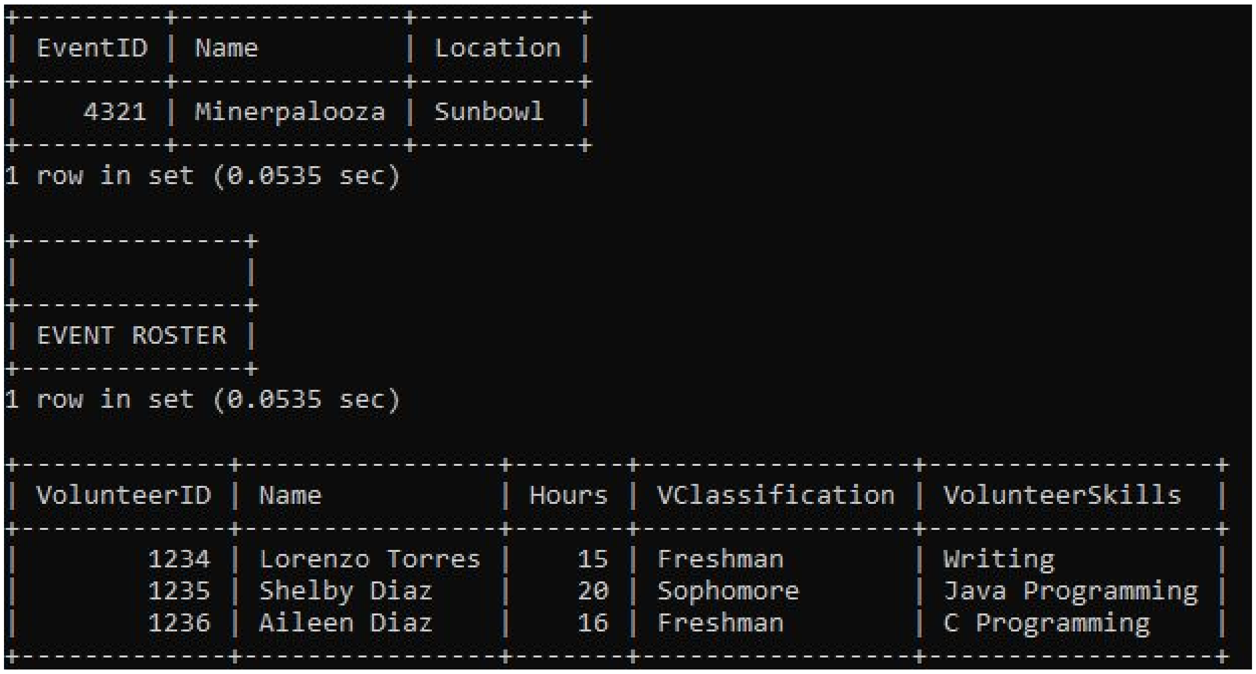


```
+---------+----------------------+-------------+
| EventID | NUMBER_OF_VOLUNTEERS | TOTAL_HOURS |
+---------+----------------------+-------------+
|    4321 |                    3 |          51 |
+---------+----------------------+-------------+
```

Which can also be broken down by individual event with as in procedures:

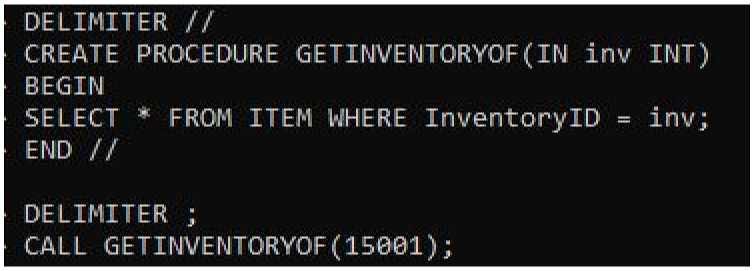
```
CREATE PROCEDURE GETREPORTBYT (IN eventn INT)
BEGIN SELECT * FROM EVENT WHERE EventID = eventn;
SELECT  'EVENT ROSTER' AS '';
SELECT VOLUNTEER.VolunteerID, USER>Name, VOLUNTEERREPORT.Hours,
VOLUNTEER.VClassification, VOLUNTEERSKILLS.VolunteerSkills FROM
(((USER INNER JOIN VOLUNTEER ON USER.UserID = VOLUNTEER.VolunteerID)
INNER JOIN VOLUNTEERREPORT ON USER.UserID=VOLUNTEERREPORT.VolunteerID)
INNER JOIN VOLUNTEERSKILLS ON USER.UserID = VOLUNTEERSKILLS.VolunteerID)
WHERE VOLUNTEER.EventID = eventn;
END//
```

Result:



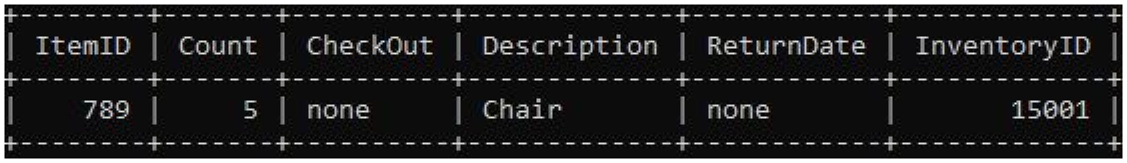
```
+---------+--------------+----------+
| EventID | Name         | Location |
+---------+--------------+----------+
|    4321 | Minerpalooza | Sunbowl  |
+---------+--------------+----------+
1 row in set (0.0535 sec)

+--------------+
|              |
+--------------+
| EVENT ROSTER |
+--------------+
1 row in set (0.0535 sec)

+-------------+----------------+-------+-----------------+------------------+
| VolunteerID | Name           | Hours | VClassification | VolunteerSkills  |
+-------------+----------------+-------+-----------------+------------------+
|        1234 | Lorenzo Torres |    15 | Freshman        | Writing          |
|        1235 | Shelby Diaz    |    20 | Sophomore       | Java Programming |
|        1236 | Aileen Diaz    |    16 | Freshman        | C Programming    |
+-------------+----------------+-------+-----------------+------------------+
```

19. The system shall provide the option to allocate an inventory with items and their description
Select items
Items would be reviewed with the following procedure:



```
DELIMITER //
CREATE PROCEDURE GETINVENTORYOF(IN inv INT)
BEGIN
SELECT * FROM ITEM WHERE InventoryID = inv;
END //

DELIMITER ;
CALL GETINVENTORYOF(15001);
```

Result:

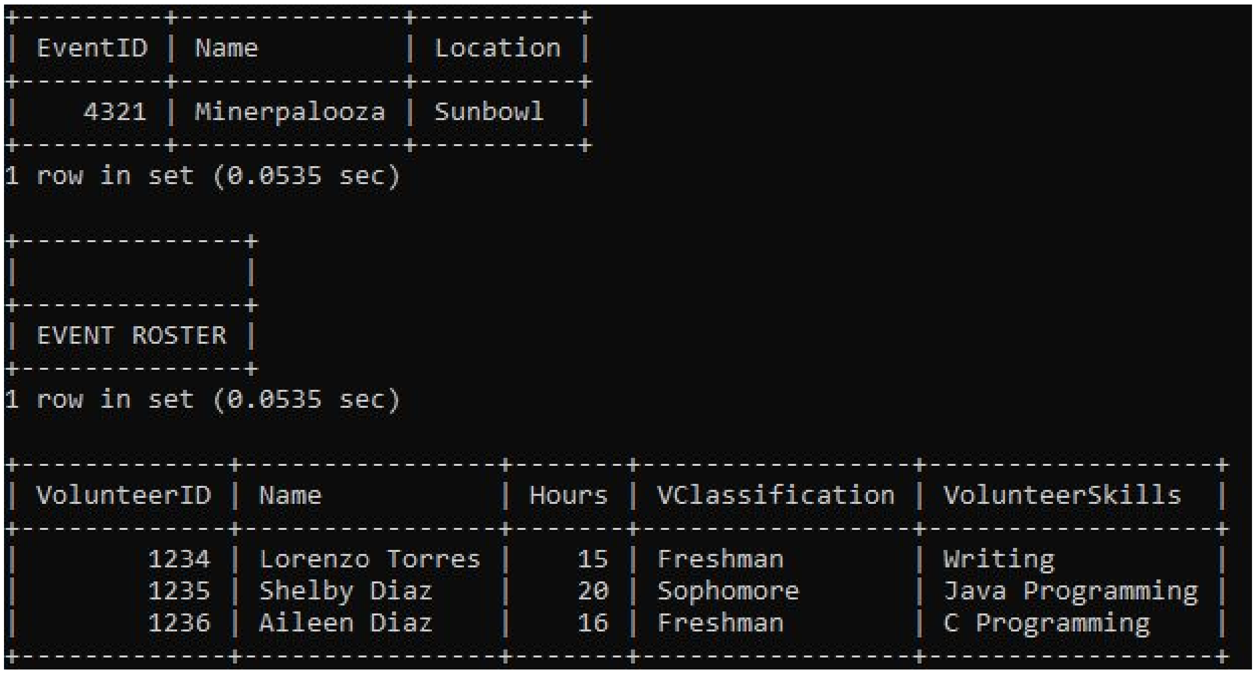


```
+--------+-------+----------+-------------+------------+-------------+
| ItemID | Count | CheckOut | Description | ReturnDate | InventoryID |
+--------+-------+----------+-------------+------------+-------------+
|    789 |     5 | none     | Chair       | none       |       15001 |
+--------+-------+----------+-------------+------------+-------------+
```

21. The system shall have a report function executed by admins which lets them obtain a report by event.

```
CREATE PROCEDURE GETREPORTBYT (IN eventn INT)
BEGIN
SELECT * FROM EVENT WHERE EventID = eventn;
SELECT  'EVENT ROSTER' AS '';
SELECT VOLUNTEER.VolunteerID, USER>Name, VOLUNTEERREPORT.Hours,
VOLUNTEER.VClassification, VOLUNTEERSKILLS.VolunteerSkills FROM
(((USER INNER JOIN VOLUNTEER ON USER.UserID = VOLUNTEER.VolunteerID)
INNER JOIN VOLUNTEERREPORT ON USER.UserID=VOLUNTEERREPORT.VolunteerID)
INNER JOIN VOLUNTEERSKILLS ON USER.UserID = VOLUNTEERSKILLS.VolunteerID)
WHERE VOLUNTEER.EventID = eventn;
END//
```

Result:



```
+----------+--------------+----------+
| EventID  | Name         | Location |
+----------+--------------+----------+
|     4321 | Minerpalooza | Sunbowl  |
+----------+--------------+----------+
1 row in set (0.0535 sec)

+--------------+
|              |
+--------------+
| EVENT ROSTER |
+--------------+
1 row in set (0.0535 sec)

+-------------+----------------+-------+-----------------+------------------+
| VolunteerID | Name           | Hours | VClassification | VolunteerSkills  |
+-------------+----------------+-------+-----------------+------------------+
|        1234 | Lorenzo Torres |    15 | Freshman        | Writing          |
|        1235 | Shelby Diaz    |    20 | Sophomore       | Java Programming |
|        1236 | Aileen Diaz    |    16 | Freshman        | C Programming    |
+-------------+----------------+-------+-----------------+------------------+
```

References

Textbook: Elmasri, R., & Navathe, S. (2011). *Database systems* (Vol. 9). Boston, MA: Pearson Education.

Appendix A: Attribution Info

Emmanuel Menier: Updated relational model, normalized schema.

Leonardo Orea-Amador: Expanded the queries from our last assignment to incorporate views, procedures, and triggers, and report queries.

Ivan Perez: Updated document to include changes recommended in feedback. Updated slides in the presentation and contributed to the expansion of procedures, triggers, and reports.

Jaime Salas: Added an admin and volunteer sign in option for the login page, Created an admin dashboard that retrieves, Demographics, Inventory and Volunteer Reports. Created a Volunteer Dashboard with an image uploader option on the sidebar, updated the main page.