Ozzapalooza

Database Design Document

Andrew Baran

CMPT 308

Marist College

Table of Contents:

Executive Summary:

Overview:

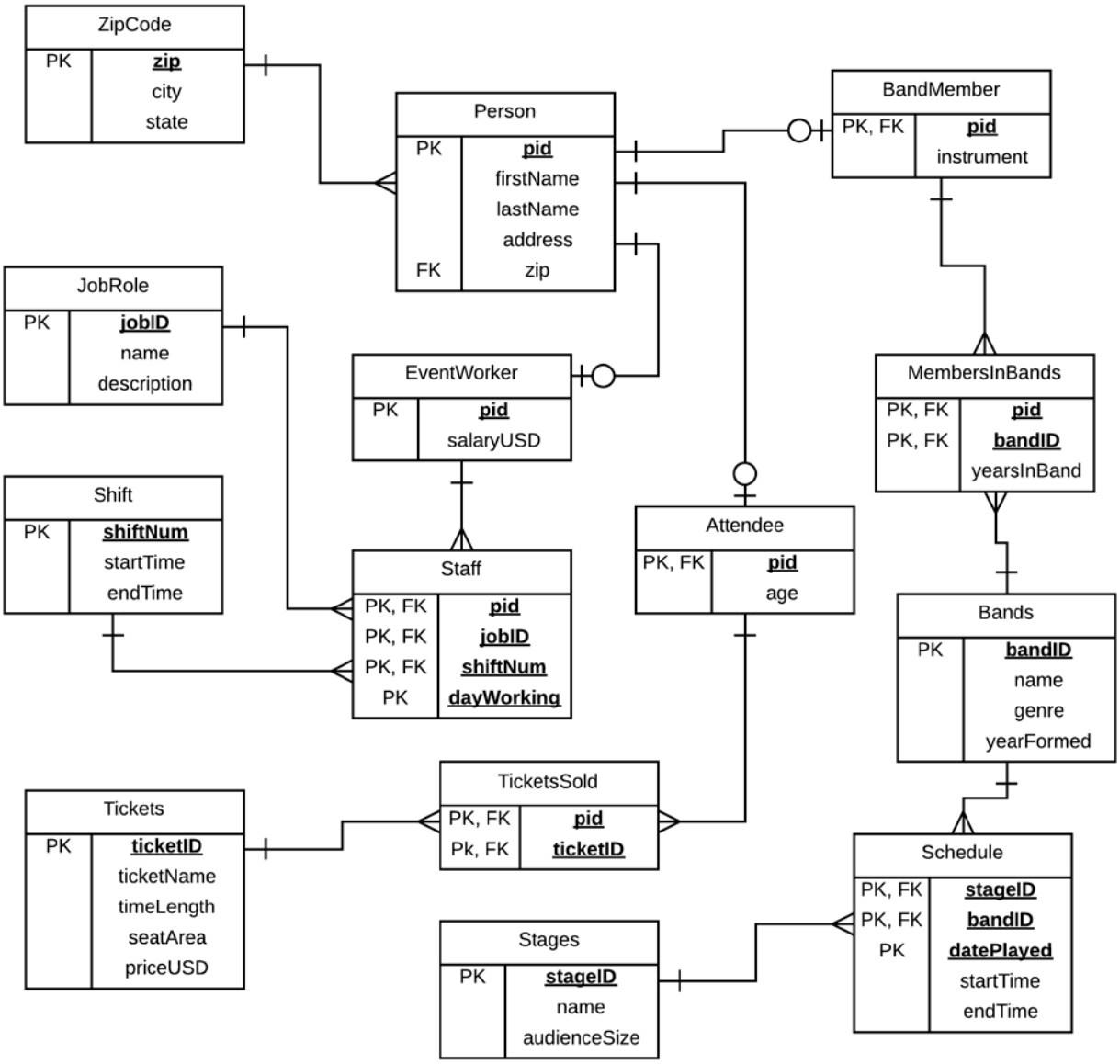
Ozzapalooza is one of the premiere music festivals in the United States. Located in and around the area of Phoenix, Arizona, this three-day long festival of music, energy, and fun brings in tens of thousands of individuals from all over the country. Many genres of music are heard, from classic rock to death metal, and many notable bands, like Rush and Slayer, perform each year to their fans.

Objectives:

Our consulting company has been hired by the organizers of the Ozzapalooza to create an appropriate database system for them to keep better track of the information regarding their employees, bands, and the attendees of the festival. Our goal was to create a system that would be very easy to modify the information necessary to the festival, such as the scheduling the bands and employees. As well as creating the necessary structure for the database, we have also provided the organizers of the festival with a variety of stored procedures and triggers to ensure the reliability of the system and the data within it, as well as queries to help in creating the necessary reports and statistics of the festival.

The outline of our proposed Database system is within this document. Each page contains a separate entity of the database system, which in total makes up the schema of the database system. As well as each entity and its functional dependencies being shown, the views, stored procedures, triggers, and security permissions for the database schema are also shown. This database system was designed in PostgreSQL 9.3.2, and it has been tested thoroughly to ensure accuracy and reliability.

ER Diagram:


ZipCode
PK
zip
city
state
Person
PK
pid
firstName
lastName
address
FK
zip
BandMember
PK, FK
pid
instrument
JobRole
PK
jobID
name
description
EventWorker
PK
pid
salaryUSD
MembersInBands
PK, FK
pid
PK, FK
bandID
yearsInBand
Shift
PK
shiftNum
startTime
endTime
Staff
PK, FK
pid
PK, FK
jobID
PK, FK
shiftNum
PK
dayWorking
Attendee
PK, FK
pid
age
Bands
PK
bandID
name
genre
yearFormed
Tickets
PK
ticketID
ticketName
timeLength
seatArea
priceUSD
TicketsSold
PK, FK
pid
Pk, FK
ticketID
Schedule
PK, FK
stageID
PK, FK
bandID
PK
datePlayed
startTime
endTime
Stages
PK
stageID
name
audienceSize

Tables

Person Table

Purpose:

Table that stores the information about each person involved in the festival (attendees, event workers, and band members). Identifying information about the individual is stored in this table.

Functional Dependencies:

pid -> firstName, lastName, address, zip

```
-- Person
create table person
(
    pid     char(7) not null,

    firstName   text,
    lastName    text,
    address     text,

    zip     text references zipCode(zip),

    primary key(pid)
);
```

	pid character(7)	firstname text	lastname text	address text	zip text
1	p000001	Andrew	Apple	123 Fake St	06614
2	p000002	Bobby	Bacon	456 Fake St	12601
3	p000003	Carmen	Cactus	789 Fake St	06801
4	p000004	Dilbert	Dingbat	3399 North Rd	12601
5	p000005	Elie	Elephant	1 Bulldog Boulevard	06614
6	p000006	Frank	Foodie	283 Music St	12538
7	p000007	George	Giraffe	184 Costanza Lane	06614
8	p000008	Harry	Hamburger	128 Lane St	06701
9	p000009	Iago	Igloo	1485 Cherry Lane	06801
10	p000010	Jimmy	James	485 Peach St	06701
11	p000011	Tom	Araya	582 Metal Lane	06701
12	p000012	Kerry	King	293 Street Lane	06614
13	p000013	Jeff	Hanneman	123 Heaven Rd	06601
14	p000014	Dave	Lombardo	482 Fake St	12601
15	p000015	Geddy	Lee	48 Fake Rd	06614
16	p000016	Neil	Peart	123 Marist St	12601
17	p000017	Alex	Lifeson	456 Marist St	06701
18	p000018	Ozzy	Osbourne	482 Druggy Lane	12601

Tables

ZipCode Table

Purpose:

Table that stores the information of the zip codes of each person involved in the festival. Each zip code is associated with its corresponding city and state.

Functional Dependencies:

zip -> city, state

```
-- ZipCode
create table zipCode
(
    zip     text not null,

    city    text,
    state   char(2),

    primary key(zip)
);
```

	zip text	city text	state character(2)
1	06701	Waterbury	CT
2	06601	Bridgeport	CT
3	06614	Stratford	CT
4	06801	Bethel	CT
5	12601	Poughkeepsie	NY
6	12538	Hyde Park	NY
7	12524	Fishkill	NY

Tables

Attendee Table

Purpose:

Table that stores the information of each person (customer) attending the festival, along with their age. This table is an entity subtype of the table Person.

Functional Dependencies:

pid -> age

```
-- Attendee
create table attendee
(
    pid     char(7) not null references person(pid),

    age     integer,

    primary key(pid)
);
```

	pid character(7)	age integer
1	p000006	18
2	p000007	25
3	p000008	32
4	p000009	44
5	p000010	59

Tables

EventWorker Table

Purpose:

Table that stores the information about each employee that works at the festival, along with that employee's salary in USD. This table is an entity subtype of the table Person.

Functional Dependencies:

pid -> salaryUSD

```
-- EventWorker
create table eventWorker
(
    pid     char(7) not null references person(pid),

    salaryUSD   real,

    primary key(pid)
);
```

	pid character(7)	salaryusd real
1	p000001	15
2	p000002	20
3	p000003	25
4	p000004	30
5	p000005	40

Tables

BandMember Table

Purpose:

Table that stores the information about each band member, which includes the instrument they play. This table is an entity subtype of the Person table.

Functional Dependencies:

pid -> instrument

```
-- BandMember
create table bandMember
(
    pid     char(7) not null references person(pid),

    instrument      text,

    primary key(pid)
);
```

	pid character(7)	instrument text
1	p000011	Vocals
2	p000012	Guitar
3	p000013	Guitar
4	p000014	Drums
5	p000015	Vocals
6	p000016	Drums
7	p000017	Guitar
8	p000018	Vocals

Tables

Bands Table

Purpose:

Table that stores the information of each band, including their name, genre, and other information about their history.

Functional Dependencies:

bandID -> name, genre, yearFormed

```
-- Bands
create table bands
(
    bandID  char(5) not null,

    name    text,
    genre   text,
    yearFormed  integer,

    primary key(bandID)
);
```

	bandid character(5)	name text	genre text	yearformed integer
1	b0001	Slayer	Thrash Metal	1981
2	b0002	Rush	Progressive Rock	1968
3	b0003	Black Sabbeth	Metal	1968

Tables

Stages Table

Purpose:

Table that stores the information about each stage that is at the festival, which includes the capacity of each audience for that stage.

Functional Dependencies:

stageID -> name, audienceSize

```
-- Stages
create table stages
(
    stageID     char(4) not null,

    name    text,
    audienceSize    integer,

    primary key(stageID)
);
```

	stageid character(4)	name text	audiencesize integer
1	s001	Main Stage 1	10000
2	s002	Main Stage 2	10000
3	s003	Side Stage	5000
4	s004	Indoor Stage	3000

Tables

JobRole Table

Purpose:

Table that stores the information about each job that an event worker at the festival can perform, including a description of that job

Functional Dependencies:

jobID -> name, description

```
-- JobRole
create table jobRole
(
    jobID   char(4) not null,

    name    text,
    description text,

    primary key(jobID)
);
```

	jobid character(4)	name text	description text
1	j001	Security	Protects people
2	j002	Food vendor	Sells food to attendees
3	j003	Merchandise vendor	Sells shirts and the like to attendees
4	j004	Ticket seller	Hands out tickets
5	j005	Stage crew	Helps set up bands on stage
6	j006	Groundskeeper	Keeps festival grounds clean

Tables

Shift Table

Purpose:

Table that stores the information about each shift that an event worker can work during, including the time frame of each shift.

Functional Dependencies:

shiftNum -> startTime, endTime

```
-- Shift
create table shift
(
    shiftNum    integer not null,

    startTime   time,
    endTime     time,

    primary key(shiftNum)
);
```

	shiftnum integer	starttime time without time zone	endtime time without time zone
1	1	09:00:00	14:00:00
2	2	14:00:00	19:00:00
3	3	19:00:00	00:00:00

Tables

Staff Table

Purpose:

Table that stores the information about each employee, when they are working and what job they are working on for a given day.

Functional Dependencies:

pid + jobID + shiftNum + daysWorking ->

```
-- Staff
create table staff
(
    pid         char(7) not null references eventWorker(pid),
    jobID       char(4) not null references jobRole(jobID),
    shiftNum    integer not null references shift(shiftNum),
    dayWorking  date not null,

    primary key(pid, jobID, shiftNum, dayWorking)
);
```

	pid character(7)	jobid character(4)	shiftnum integer	dayworking date
1	p000001	j001	2	2012-01-01
2	p000001	j001	3	2012-01-01
3	p000002	j002	1	2012-01-02
4	p000003	j005	1	2012-01-02
5	p000003	j005	1	2012-01-03
6	p000004	j006	3	2012-01-01
7	p000005	j003	1	2012-01-01
8	p000005	j003	1	2012-01-02

Tables

Tickets Table

Purpose:

Table that stores the information about each type of ticket that is available for sale.

Functional Dependencies:

ticketID -> ticketName, timeLength, seatArea, priceUSD

```
-- Tickets
create table tickets
(
    ticketID    char(4) not null,

    ticketName  text,
    timeLength  integer,
    seatArea    text,
    priceUSD    real,

    primary key(ticketID)
);
```

	ticketid character(4)	ticketname text	timelength integer	seatarea text	priceusd real
1	t001	One Day Grass	1	Grass	10
2	t002	Three Day Grass	3	Grass	25
3	t003	One Day Pit	1	Pit	30
4	t004	Three Day Pit	3	Pit	80
5	t005	One Day VIP	1	VIP	100
6	t006	Three Day VIP	3	VIP	250

Tables

TicketsSold Table

Purpose:

Table that stores the information about which attendee bought which type of ticket for the festival.

Functional Dependencies:

pid + ticketID ->

```
-- TicketsSold
create table ticketsSold
(
    pid         char(7) not null references attendee(pid),
    ticketID    char(4) not null references tickets(ticketID),

    primary key(pid, ticketID)
);
```

	pid character(7)	ticketid character(4)
1	p000006	t001
2	p000007	t001
3	p000008	t002
4	p000009	t003
5	p000010	t004

Tables

MembersInBands Table

Purpose:

Table that stores the information about which band members are in which band, as well as how long they have been in that band.

Functional Dependencies:

pid + bandID -> yearsInBand

```
-- MembersInBands
create table membersInBands
(
    pid     char(7) not null references bandMember(pid),
    bandID  char(5) not null references bands(bandID),

    yearsInBand     integer,

    primary key(pid, bandID)
);
```

	pid character(7)	bandid character(5)	yearsinband integer
1	p000011	b0001	33
2	p000012	b0001	33
3	p000013	b0001	32
4	p000014	b0001	21
5	p000015	b0002	46
6	p000016	b0002	46
7	p000017	b0002	46
8	p000018	b0003	41

Tables

Schedule Table

Purpose:

Table that stores the information about which band plays on which stage at a given day, as well as the start and end time of that bands performance.

Functional Dependencies:

stageID + bandID + datePlayed -> startTime, endTime

```
-- Schedule
create table schedule
(
    stageID     char(4) not null references stages(stageID),
    bandID      char(5) not null references bands(bandID),
    datePlayed  date not null,
    startTime   time,
    endTime     time,

    primary key(stageID, bandID, datePlayed)
);
```

	stageid character(4)	bandid character(5)	dateplayed date	starttime time without time zone	endtime time without time zone
1	s001	b0001	2012-01-01	12:00:00	14:00:00
2	s002	b0001	2012-01-02	15:00:00	17:00:00
3	s001	b0002	2012-01-01	19:00:00	21:00:00
4	s003	b0002	2012-01-03	10:00:00	12:00:00
5	s001	b0003	2012-01-03	14:00:00	16:00:00
6	s004	b0003	2012-01-02	20:00:00	21:00:00

Views

Band Information View

Purpose:

View that allows a person to see a succinct summary of a band, its members, and other important information.

```
create view bandInformation as

    select person.firstName, person.lastName, bands.name as bandName,
    bandMember.instrument, membersInBands.yearsInBand

    from bands, bandMember, membersInBands, person
    where bands.bandID = membersInBands.bandID
        and bandMember.pid = membersInBands.pid
        and person.pid = bandMember.pid;
```

Results:

	firstname text	lastname text	bandname text	instrument text	yearsinband integer
1	Tom	Araya	Slayer	Vocals	33
2	Kerry	King	Slayer	Guitar	33
3	Jeff	Hanneman	Slayer	Guitar	32
4	Dave	Lombardo	Slayer	Drums	21
5	Geddy	Lee	Rush	Vocals	46
6	Neil	Peart	Rush	Drums	46
7	Alex	Lifeson	Rush	Guitar	46
8	Ozzy	Osbourne	Black Sabbeth	Vocals	41

Views

Complete Schedule View

Purpose:

View that allows a person to see the complete schedule of the festival, including which band is on which stage at a certain time.

```
create view completeSchedule as

    select stages.name as stageName, bands.name as bandName,
    schedule.datePlayed, schedule.startTime, schedule.endTime

    from schedule, bands, stages
    where schedule.bandID = bands.bandID
        and schedule.stageID = stages.stageID;
```

Results:

	stagename text	bandname text	dateplayed date	starttime time without time zone	endtime time without time zone
1	Main Stage 1	Slayer	2012-01-01	12:00:00	14:00:00
2	Main Stage 2	Slayer	2012-01-02	15:00:00	17:00:00
3	Main Stage 1	Rush	2012-01-01	14:00:00	16:00:00
4	Side Stage	Rush	2012-01-03	10:00:00	12:00:00
5	Main Stage 1	Black Sabbeth	2012-01-03	14:00:00	16:00:00
6	Indoor Stage	Black Sabbeth	2012-01-02	20:00:00	21:00:00

Views

Employee Schedule View

Purpose:

View that allows a person to see the complete employee schedule, including which job an event worker is working on and at what time.

```
create view employeeSchedule as

    select person.firstName, person.lastName, staff.dayWorking, shift.shiftNum,
    jobRole.name as jobName, shift.startTime, shift.endTime

    from staff, jobRole, shift, eventWorker, person
    where staff.jobID = jobRole.jobID
        and staff.shiftNum = shift.shiftNum
        and staff.pid = eventWorker.pid
        and eventWorker.pid = person.pid;
```

Results:

	firstname text	lastname text	dayworking date	shiftnum integer	jobname text	starttime time without time zone	endtime time without time zone
1	Andrew	Apple	2012-01-01	2	Security	14:00:00	19:00:00
2	Andrew	Apple	2012-01-01	3	Security	19:00:00	00:00:00
3	Bobby	Bacon	2012-01-02	1	Food vendor	09:00:00	14:00:00
4	Carmen	Cactus	2012-01-02	1	Stage crew	09:00:00	14:00:00
5	Carmen	Cactus	2012-01-03	1	Stage crew	09:00:00	14:00:00
6	Dilbert	Dingbat	2012-01-01	3	Groundskeeper	19:00:00	00:00:00
7	Elie	Elephant	2012-01-01	1	Merchandise vendor	09:00:00	14:00:00
8	Elie	Elephant	2012-01-02	1	Merchandise vendor	09:00:00	14:00:00

Views

Customer Data View

Purpose:

View that allows a person to see the information about each attendee (customer) that is attending the festival, as well as the ticket that they bought.

```
create view customerData as

    select person.firstName, person.lastName, attendee.age,
    tickets.ticketName, tickets.priceUSD

    from ticketsSold, attendee, person, tickets
    where ticketsSold.pid = attendee.pid
        and ticketsSold.ticketID = tickets.ticketID
        and attendee.pid = person.pid;
```

Results:

	firstname text	lastname text	age integer	ticketname text	priceusd real
1	Frank	Foodie	18	One Day Grass	10
2	George	Giraffe	25	One Day Grass	10
3	Harry	Hamburger	32	Three Day Grass	25
4	Iago	Igloo	44	One Day Pit	30
5	Jimmy	James	59	Three Day Pit	80

Queries and Reports

Sales Numbers Report

Purpose:

Query that allows management to see the sales numbers of the festival, as well as number and type of each ticket sold.

```
select tickets.ticketName, priceUSD, count as salesNumbers,
(tickets.priceUSD * count) as profitUSD
from tickets
    join
(select ticketsSold.ticketID, count(ticketsSold.ticketID)
from ticketsSold
group by ticketsSold.ticketID) ticketSales
on tickets.ticketID = ticketSales.ticketID
order by profitUSD desc;
```

Results:

	ticketname text	priceusd real	salesnumbers bigint	profitusd double precision
1	Three Day Pit	80	1	80
2	One Day Pit	30	1	30
3	Three Day Grass	25	1	25
4	One Day Grass	10	2	20

Queries and Reports

Attendee Location Report

Purpose:

Query that allows management to see the number of attendees from each state. This is useful for future advertising campaigns and analytic research.

```
select zipCode.state, count(zipCode.state) as population
from person, attendee, zipCode
where person.pid = attendee.pid
    and person.zip = zipCode.zip
group by zipCode.state;
```

Results:

	state character(2)	population bigint
1	CT	4
2	NY	1

Queries and Reports

Stage Usage Report

Purpose:

Query that produces a report that shows the number of hours that each stage is being used throughout the festival.

```
select stages.name as stageName,
sum((extract(minute from endTime - startTime) / 60.0) + extract (hour from endTime - startTime)) as hoursUsed
from schedule, stages
where schedule.stageID = stages.stageID
group by stages.name
order by hoursUsed desc;
```

Results:

	stagename text	hoursused double precision
1	Main Stage 1	6
2	Side Stage	2
3	Main Stage 2	2
4	Indoor Stage	1

Queries and Reports

Attendee Age Report

Purpose:

Query that returns the average age of the attendees of the festival. This is a useful report in determining who the target population that needs advertising the most.

```
select round(avg(attendee.age), 2) as averageAge
from attendee;
```

Results:

	averageage numeric
1	35.60

Stored Procedures and Triggers

Validating Salary

Purpose:

Prevents the changing of an employee's salary to a negative value, and the original value is kept the same.

```
-- Prevent chaning of salary to negative
create or replace function validateInfo() returns trigger as
$$
begin
    if new.salaryUSD < 0 then
        update eventWorker
        set salaryUSD = old.salaryUSD
        where pid = new.pid;
    end if;

    return new;
end
$$
language PLPGSQL;


create trigger validateSalary
after update on eventWorker
for each row
execute procedure validateInfo();
```

Stored Procedures and Triggers

Scheduling Conflict Manager

Purpose:

Prevent the scheduling of two bands on the same stage at the same time on the same day.

```
-- Prevent two bands from being in the same timeslot
create or replace function preventConflict() returns trigger as
$$
declare
    conflictNumber integer;
begin
    select count(*) into conflictNumber
    from schedule
    where datePlayed = new.datePlayed
        and startTime = new.startTime;

    if conflictNumber > 1 then
        delete from schedule
        where new.stageID = stageID
            and new.bandID = bandID
            and new.datePlayed = datePlayed
            and new.startTime = startTime;

        raise notice 'Conflicting time slot: You cannot enter two bands for the same time';

        return null;
    end if;

    if conflictNumber <= 1 then
        return new;
    end if;


end
$$
language PLPGSQL;


create trigger conflictManagement
after insert on schedule
for each row
execute procedure preventConflict();
```

Stored Procedures and Triggers

Underage Attendee Verifier

Purpose:

Prevent an underage attendee (below the age of 18 years of age) from purchasing a ticket.

```
-- Prevent underage attendees (18 years or less)
create or replace function preventUnderage() returns trigger as
$$
begin

    if new.age < 18 then
        delete from attendee
        where new.pid = pid;

        raise notice 'Warning: Attendents of the festival must be 18 years or older';

        return null;
    end if;

    if new.age >= 18 then
        return new;
    end if;

end
$$ language PLPGSQL;


create trigger ageCheck
after insert on attendee
for each row
execute procedure preventUnderage();
```

Security Permissions

Overview:

There are five types of people that would need access to the database, and they are: attendees, ticket office employees, security personnel, management, and the database manager. Below are the permissions that are granted to each of these five types of people:

Attendees:

An attendee of the festival only needs access to each bands information and the schedule for when each band is on stage. This information is summed up in two of the provided views.

```
drop role if exists attendee;
create role attendee;

grant select on completeSchedule to attendee;
grant select on bandInformation to attendee;
```

Security Permissions

Ticket Office Employees:

Employees of the ticket office need access to ticket information, as well as have the ability to sell tickets to attendees.

```
drop role if exists ticketOffice;
create role ticketOffice;

grant select on tickets to ticketOffice;
grant select, insert, update, delete on ticketsSold to ticketOffice;
grant select, insert, update, delete on attendee to ticketOffice;
```

Security:

Security officials need access to the list of attendees and their related information, as they need the ability to kick an individual out if they bring the ruckus. They also need access to their schedule, which tells them where they are needed and when.

```
drop role if exists securityOfficial;
create role securityOfficial;

grant select, delete on person to securityOfficial;
grant select, delete on attendee to securityOfficial;
grant select on employeeSchedule to securityOfficial;
```

Security Permissions

Management:

The fine, smart, always-right folks over at management need access to all the tables regarding employee information, as well as the tables that define the schedule and ticket prices.

```
drop role if exists management;
create role management;

grant select, insert, update, delete on jobRole to management;
grant select, insert, update, delete on staff to management;
grant select, insert, update, delete on schedule to management;
grant select, insert, update, delete on eventWorker to management;
grant select, insert, update, delete on tickets to management;
grant select on employeeSchedule to management;
grant select on completeSchedule to management;
```

Database Manager:

The database manager gets permission to do whatever the hell he wants with the database because he designed and toiled over it.

```
drop role if exists databaseManager;
create role databaseManager;

grant all privileges on all tables in schema public to databaseManager;
```

Implementation Notes

This database design has been implemented in PostgreSQL 9.3.2. The testing of the design has been thorough and will be error free by the time the system is implemented for the festival. To interface with this database system, it is recommended that a web front end be created in order to simplify the process of adding new information to the database. By creating a website that can accurately sell tickets and automatically add attendees and their related information to the system, it would lessen the workload of the employees and provide a reliable system to work with this database system. Our company also specializes in creating web front ends that are fully compatible with the database systems that we design.

Known Problems

- Although bands cannot be scheduled for the same starting time on the same stage, bands can still be scheduled to play between the start time and end time of another band on the same stage.

- As of now, the database manager is the only person that can add stages and bands to the database.

- A zip code is associated with each person at the festival, but some people may not have a zip code (from outside of USA) or the zip code length is not enough space (9 digit zip code vs. 5 digit zip code).

- The total number of people allowed in the festival is 1,000,000. If the festival were to experience major expansion, the length of the pid field would need to be changed all around the database.

- If a band goes over the ending time that they were provided for their performance on stage, there is no way to move up every other band to compensate.

Future Enhancements

- Add the ability to specify the quantity of tickets that are available. This would allow you to sell the proper amount of tickets while still staying within the maximum capacity of each stage.
    - This would require a new trigger that checks before an insert into ticketsSold, which checks whether the ticket that is being sold is available.

- Make the scheduling conflict manager trigger more flexible, as you should not be allowed to schedule a band to the same stage when another band is already on the stage at that time.
    - As of now, it only checks if the start times conflict when scheduling bands.

- Create more elaborate queries for creating reports, such as the total pay due to each employee and a report that shows the free time between stage performances for each stage.

- Create the needed trigger to push back every band in the schedule to compensate for if a band goes over their time limit.