Ozzapalooza

Database Design Document

Andrew Baran

CMPT 308

Marist College

**Table of Contents:**

Executive Summary ………………………………………………………………………………3

ER Diagram ....................................................................................................................................4

Tables ……………………………………………………………………………………………..5

Person ……………………………………………………………………………………..5

ZipCode ...............................................................................................................................6

Attendee …………………………………………………………………………………..7

EventWorker ……………………………………………………………………………...8

BandMember………………………………………………………………………………9

Bands……………………………………………………………………………………..10

Stages…………………………………………………………………………………….11

JobRole…………………………………………………………………………………..12

Shift………………………………………………………………………………………13

Staff………………………………………………………………………………………14

Tickets……………………………………………………………………………………15

TicketsSold…………………………………………………………………………........16

MembersInBands…………………………………………………………………….......17

Schedule………………………………………………………………………………….18

Views…………………………………………………………………………………………….19

BandInformation…………………………………………………………………………19

CompleteSchedule…………………………………………………………………….....20

EmployeeSchedule……………………………………………………………………....21

CustomerData……………………………………………………………………………22

Queries…………………………………………………………………………………………...23

Sales Numbers…………………………………………………………………………...23

Attendee Location……………………………………………………………………….24

Stage Usage……………………………………………………………………………...25

Attendee Age…………………………………………………………………………….26

Stored Procedures and Triggers………………………………………………………………….27

Validating Salary………………………………………………………………………...27

Scheduling Conflict Manager……………………………………………………………28

Underage Attendee Verifier……………………………………………………………...29

Security Permissions……………………………………………………………………………..30

Implementation Notes……………………………………………………………………………33

Known Problems…………………………………………………………………………………34

Future Enhancements…………………………………………………………………………….35

**Executive Summary:**

**Overview:**

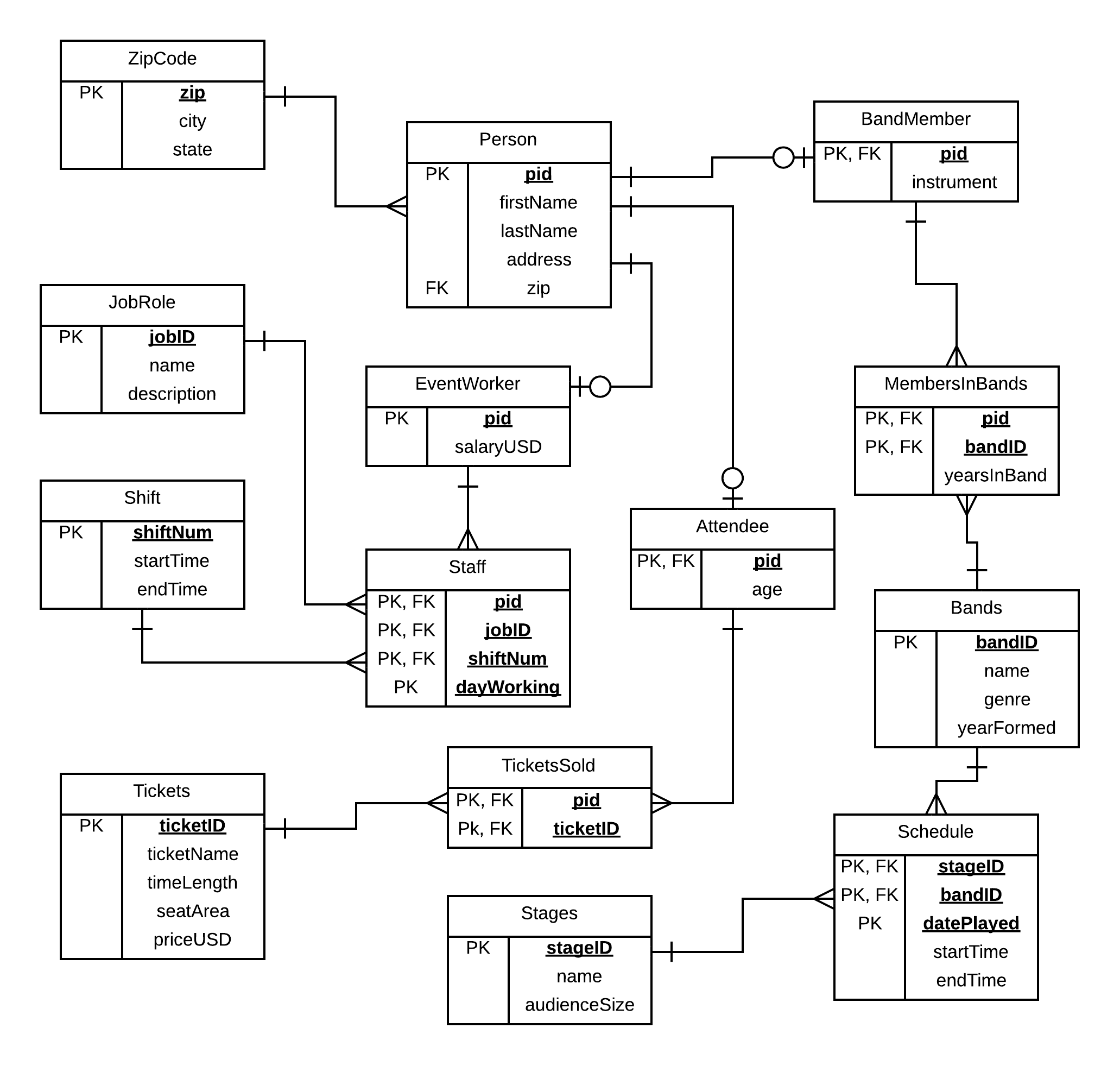
Ozzapalooza is one of the premiere music festivals in the United States. Located in and around the area of Phoenix, Arizona, this three-day long festival of music, energy, and fun brings in tens of thousands of individuals from all over the country. Many genres of music are heard, from classic rock to 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:**



**Tables**

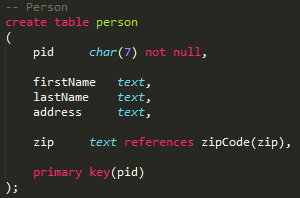
**Person Table**

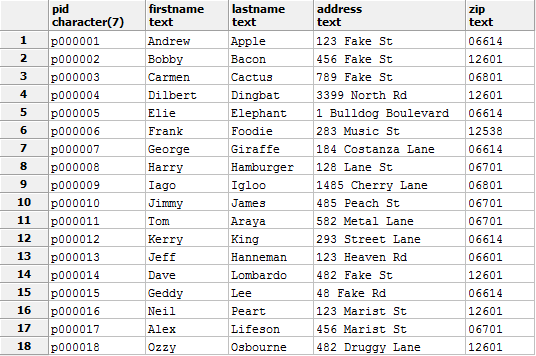
Purpose:

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





**Tables**

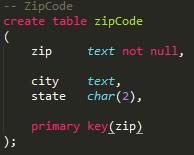
**ZipCode Table**

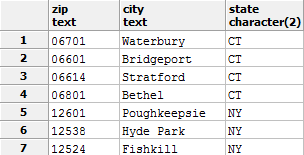
Purpose:

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





**Tables**

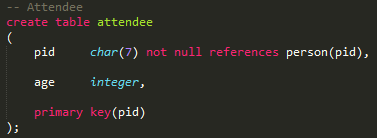
**Attendee Table**

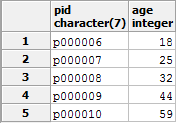
Purpose:

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





**Tables**

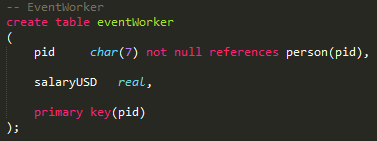
**EventWorker Table**

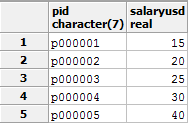
Purpose:

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





**Tables**

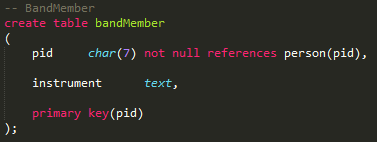
**BandMember Table**

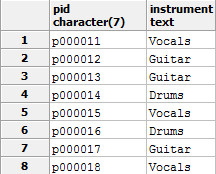
Purpose:

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





**Tables**

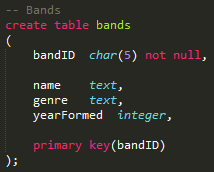
**Bands Table**

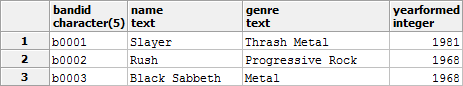
Purpose:

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

Functional Dependencies:

bandID -> name, genre, yearFormed





**Tables**

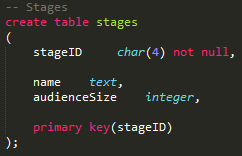
**Stages Table**

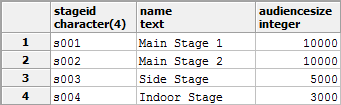
Purpose:

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





**Tables**

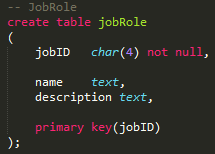
**JobRole Table**

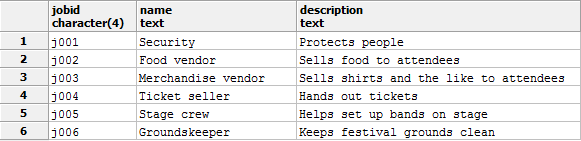
Purpose:

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





**Tables**

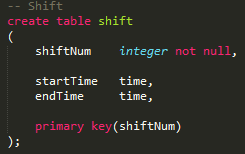
**Shift Table**

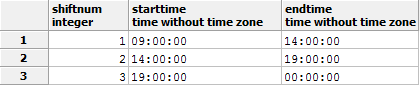
Purpose:

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





**Tables**

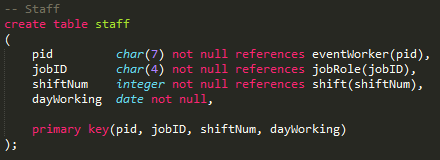
**Staff Table**

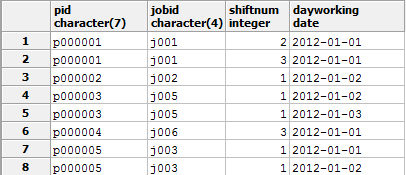
Purpose:

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 ->





**Tables**

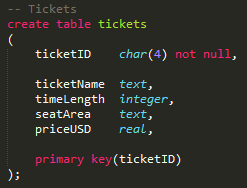
**Tickets Table**

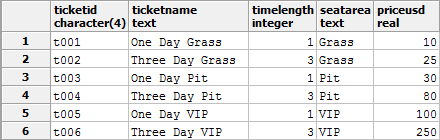
Purpose:

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

Functional Dependencies:

ticketID -> ticketName, timeLength, seatArea, priceUSD





**Tables**

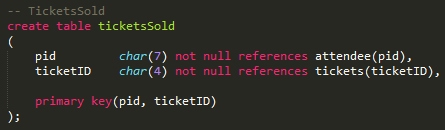
**TicketsSold Table**

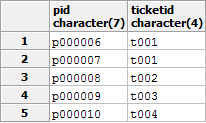
Purpose:

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

Functional Dependencies:

pid + ticketID ->





**Tables**

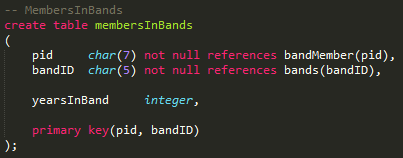
**MembersInBands Table**

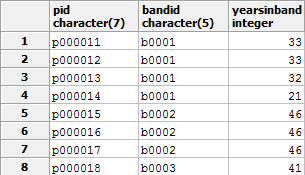
Purpose:

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





**Tables**

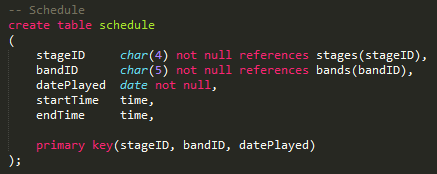
**Schedule Table**

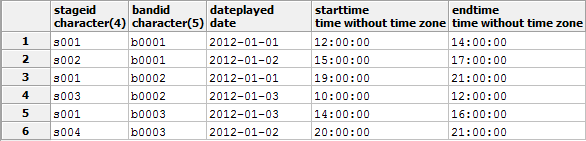
Purpose:

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



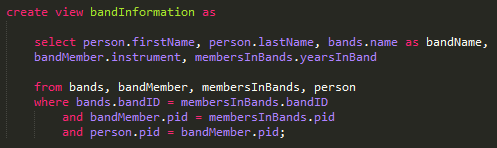


**Views**

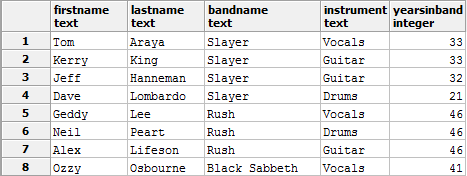
**Band Information View**

Purpose:

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



Results:

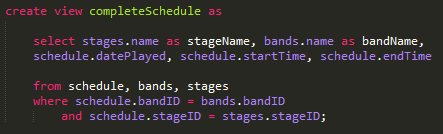


**Views**

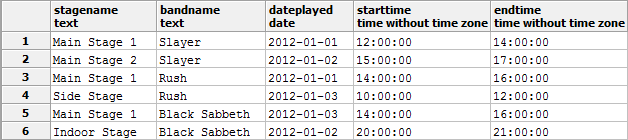
**Complete Schedule View**

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.



Results:

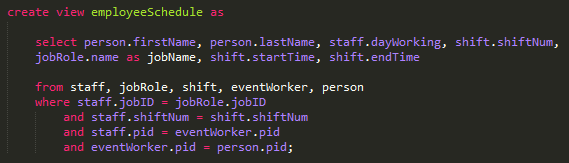


**Views**

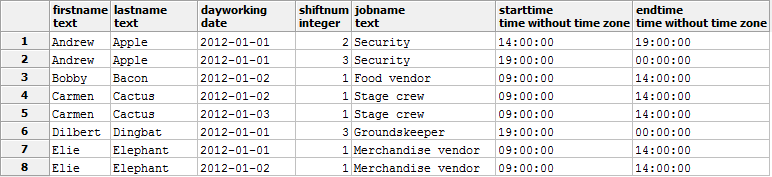
**Employee Schedule View**

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.



Results:

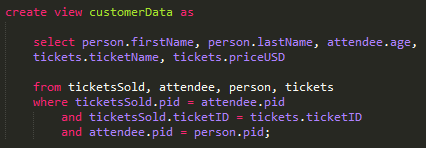


**Views**

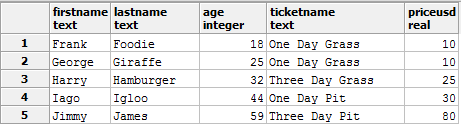
**Customer Data View**

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.



Results:

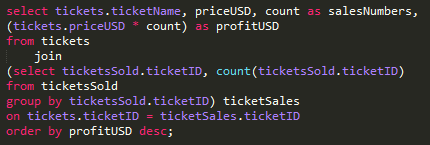


**Queries and Reports**

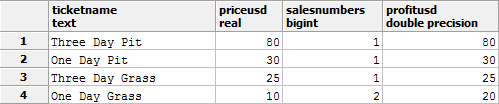
**Sales Numbers Report**

Purpose:

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



Results:

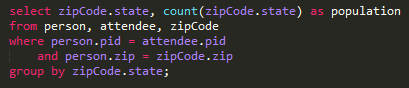


**Queries and Reports**

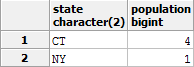
**Attendee Location Report**

Purpose:

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



Results:

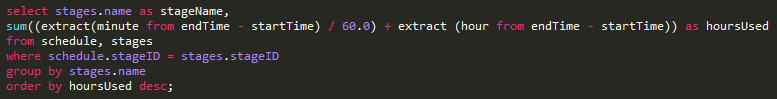


**Queries and Reports**

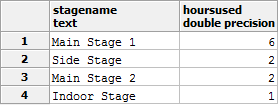
**Stage Usage Report**

Purpose:

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



Results:



**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.

C:\Users\Marist User\Desktop\Database-Management\Design Project\Pictures\queryAverageAgeTable.png

Results:

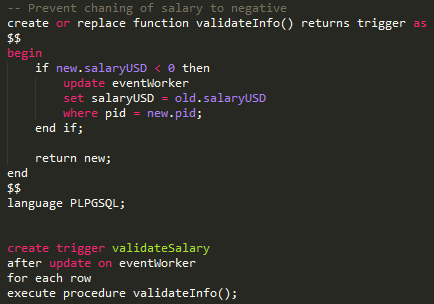
C:\Users\Marist User\Desktop\Database-Management\Design Project\Pictures\queryAverageAge.png

**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.

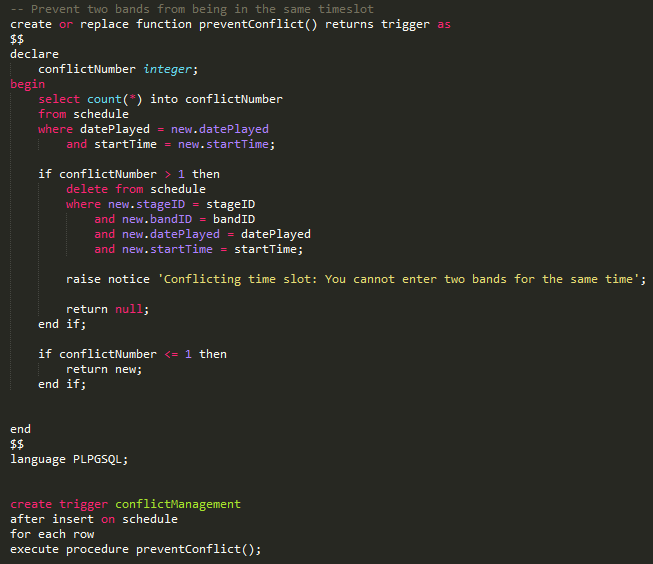


**Stored Procedures and Triggers**

**Scheduling Conflict Manager**

Purpose:

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

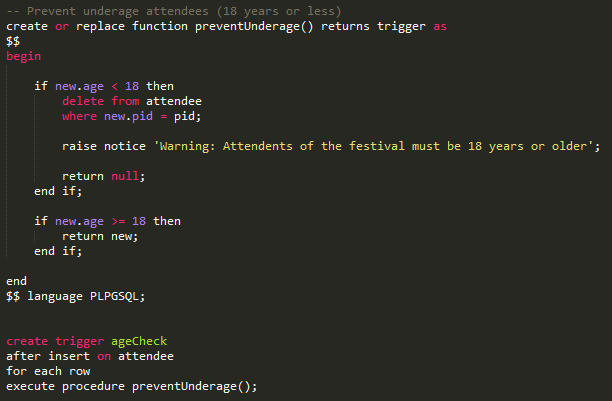


**Stored Procedures and Triggers**

**Underage Attendee Verifier**

Purpose:

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



**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.