VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAVI-590014

A DMBS Mini-Project Report
On

"Amusement Park Management System"

Submitted in partial fulfillment of the requirements for the 5th semester of
Bachelor of Engineering in Computer Science and Engineering
of Visvesvaraya Technological University, Belgavi

Submitted by:

Pranav Srivatsa	1RN16CS078
Rahul Devajji	1RN16CS070

Under the guidance of:

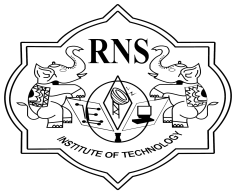
Mr. K Sunil Kumar	**Mrs. A N Ramyashree**	**Dr. H R Sashidhara**
Assitant Professor	**Assitant Professor**	**Prof. & Head**
Dept. of CSE	**Dept. of CSE**	**Dept. of CSE**

Department of Computer Science and Engineering
RNS Institute of Technology
Channasandra, Dr. Vishnuvardhan Road, Bengaluru-560 098

2017-2018

RNS Institute of Technology
Channasandra, Dr. Vishnuvaradana Road,
Bengaluru-560 098

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

Certified that the DBMS mini-project work entitled **"Amusement Park Management System"** has been successfully carried out by **Pranav Srivatsa** bearing USN **1RN16CS070** and **Rahul Devajji** bearing USN **1RN16CS078**, bonafide students of **RNS Institute of Technology** in partial fulfillment of the requirements for the **5th semester Bachelor of Engineering** in **Computer Science and Engineering** of **Visvesvaraya Technological University**, Belagavi, during the academic year 2018-2019. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated. The project report has been approved as it satisfies the mini-project requirements of DBMS lab of 5th semester BE in CSE.

Mr. Karanam Sunil Kumar	**Mrs.Manjula L**	**Dr. G T Raju**
Assitant Professor	**Assitant Professor**	**Prof. & Head**
Dept. of CSE	**Dept. of CSE**	**Dept. of CSE**

External Viva:

Name of the Examiners **Signature with date**

1.

2.

ABSTRACT

The Amusement Park Management System is an application that simulates the working of an amusement park. In this project we've tried to show the day to day activities in a park along with the Statistical analysis of the generated data.

The aim of this project is to try and simulate how a real-world park would process large amounts of data and produce information that is meaningful to the owner or/and to the administrator of the park.

We have used POSTGRESQL database to store the data generated, Python with the FLASK framework is used process the data from the database send it to the frontend where it is shown in tables and graphs.

ACKNOWLEDGEMENTS

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all. We would like to thank **Dr. H N Shivashankar**, Director, RNSIT, Bangalore, for his moral support towards completing our project. We are grateful to **Dr. M K Venkatesha**, Principal, RNSIT, Bangalore, for his support towards completing this mini project. We would like to thank **Dr. G T Raju**, Dean of Engg., Prof. and Head, Department of Computer Science and Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice. We deeply express my sincere gratitude to my guide **Mr. Karanam Sunil Kumar** and **Mrs. Manjula L.**, Asst Prof, Department of CSE, RNSIT, Bangalore, for their able guidance, regular source of encouragement and assistance throughout this project. We would like to thank all the teaching and non-teaching staff of department of Computer Science and Engineering, RNSIT, Bengaluru for their constant support and encouragement.

Date:
Place:

Pranav Srivatsa 1RN16CS070
Rahul Devajji 1RN16CS078

Contents

List of Figures

Chapter 1

Introduction

1.1 Database Technologies

The essential feature of database technology is that it provides an internal representation (model) of the external world of interest. Examples are the representation of a particular date/time/flight/aircraft in airline reservation or of item code/item description/quantity on hand/reorder level/reorder quantity in a stock control system. The technology involved is concerned primarily with maintaining the internal representation consistent with external reality; this involves the results of extensive Research and Development over the past 30 years in areas such as user requirements analysis, data modeling, process modeling, data integrity, concurrency, transactions, file organization, indexing, rollback and recovery, persistent programming, object-orientation, logic programming, deductive database systems, active database systems... and in all these (and other) areas there remains much to be done. The essential point is that database technology is a CORE TECHNOLOGY with links to:

- Information management / processing
- Data analysis / statistics
- Data visualization / presentation
- Multimedia and hypermedia
- Office and document systems
- Business processes, workflow, CSCW (computer-supported cooperative work)

Relational DBMS is the modern base technology for many business applications. It offers flexibility and easy-to-use tools at the expense of ultimate performance. More recently relational systems have started to extend their facilities in the directions of information retrieval, object-orientation and deductive/active systems leading to the so-called 'Extended Relational Systems'.

Information Retrieval Systems started with handling library catalogues and extended to full free-text utilizing inverted index technology with a lexicon or thesaurus. Modern systems utilize some KBS (knowledge-based systems) techniques to improve retrieval.

Object-Oriented DBMS started for engineering applications where objects are complex, have versions and need to be treated as a complete entity. OODBMSs share many of

the OOPL features such as identity, inheritance, late binding, overloading and overriding. OODBMSs have found favour in engineering and office systems but have not yet been successful in traditional application areas. Deductive / Active DBMS have emerged over the last 20 years and combine logic programming technology with database technology. This allows the database itself to react to external events to maintain dynamically its integrity with respect to the real world.

1.2 Characteristics of Database Approach

Traditionally, data was organized in file formats. DBMS was a new concept then, and all the research was done to make it overcome the deficiencies in traditional style of data management. A modern DBMS has the following characteristics:

- Real-world entity - A modern DBMS is more realistic and uses real-world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use students as an entity and their age as an attribute.
- Relation-based tables - DBMS allows entities and relations among them to form tables. A user can understand the architecture of a database just by looking at the table names.
- Isolation of data and application - A database system is entirely different than its data. A database is an active entity, whereas data is said to be passive, on which the database works and organizes. DBMS also stores metadata, which is data about data, to ease its own process.
- Less redundancy - DBMS follows the rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Normalization is a mathematically rich and scientific process that reduces data redundancy.
- Consistency - Consistency is a state where every relation in a database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state. A DBMS can provide greater consistency as compared to earlier forms of data storing applications like file-processing systems.
- Query Language - DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and as different filtering options as required to retrieve a set of data. Traditionally it was not possible where file-processing system was used.
- ACID Properties - DBMS follows the concepts of Atomicity, Consistency, Isolation, and Durability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- Multiuser and Concurrent Access - DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.

- Multiple views - DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.

- Security - Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

1.3 Applications of DBMS

Applications where we use Database Management Systems are:

- **Telecom:** There is a database to keeps track of the information regarding calls made, network usage, customer details etc. Without the database systems it is hard to maintain that huge amount of data that keeps updating every millisecond.

- **Industry:** Where it is a manufacturing unit, warehouse or distribution centre, each one needs a database to keep the records of ins and outs. For example distribution centre should keep a track of the product units that supplied into the centre as well as the products that got delivered out from the distribution centre on each day; this is where DBMS comes into picture.

- **Banking System:** For storing customer info, tracking day to day credit and debit transactions, generating bank statements etc. All this work has been done with the help of Database management systems.

- **Education Sector:** Database systems are frequently used in schools and colleges to store and retrieve the data regarding student details, staff details, course details, exam details, payroll data, attendance details, fees details etc. There is a hell lot amount of inter-related data that needs to be stored and retrieved in an efficient manner.

- **Online Shopping:** You must be aware of the online shopping websites such as Amazon, Flip kart etc. These sites store the product information, your addresses and preferences, credit details and provide you the relevant list of products based on your query. All this involves a Database management system.

1.4 Problem Description/Statement

Amusement parks get thousands of customers of varying age groups every single day. Each individual customer will have a unique taste and accordingly chooses an attraction in the park. This wealth of data generally goes unstored and unused. In this project we have tried to store the activities of each customer once he enters the park. This data can be crucial to know about the areas of improvement, which will benefit the owners by generating more revenue.

We have automated the tasks of adding customers to the park, since it is tedious to enter the details of each customer manually, not only is that task time-consuming, it is also impractical to do statistical analysis on minimal data. On the press of a button, customers are added into the park and are allocated to different attractions in the park for a pre-defined time period. In a way, we are simulating time-travel to generate the required data.

Our project is essentially the admin-dashboard of the amusement park. A user with valid credentials, on logging in, is redirected to the page where the real-time statistics of the park are displayed in the form of bar-graphs, line-graphs and tables.

Chapter 2

Requirement Analysis

2.1 Hardware Requirements

The Hardware requirements are very minimal and the program can be run on most of the machines.
Processor : Pentium4 processor
Processor Speed : 2.4 GHz
RAM : 1 GB
Storage Space : 40 GB
Monitor Resolution : 1024*768 or 1336*768 or 1280*1024

2.2 Software Requirements

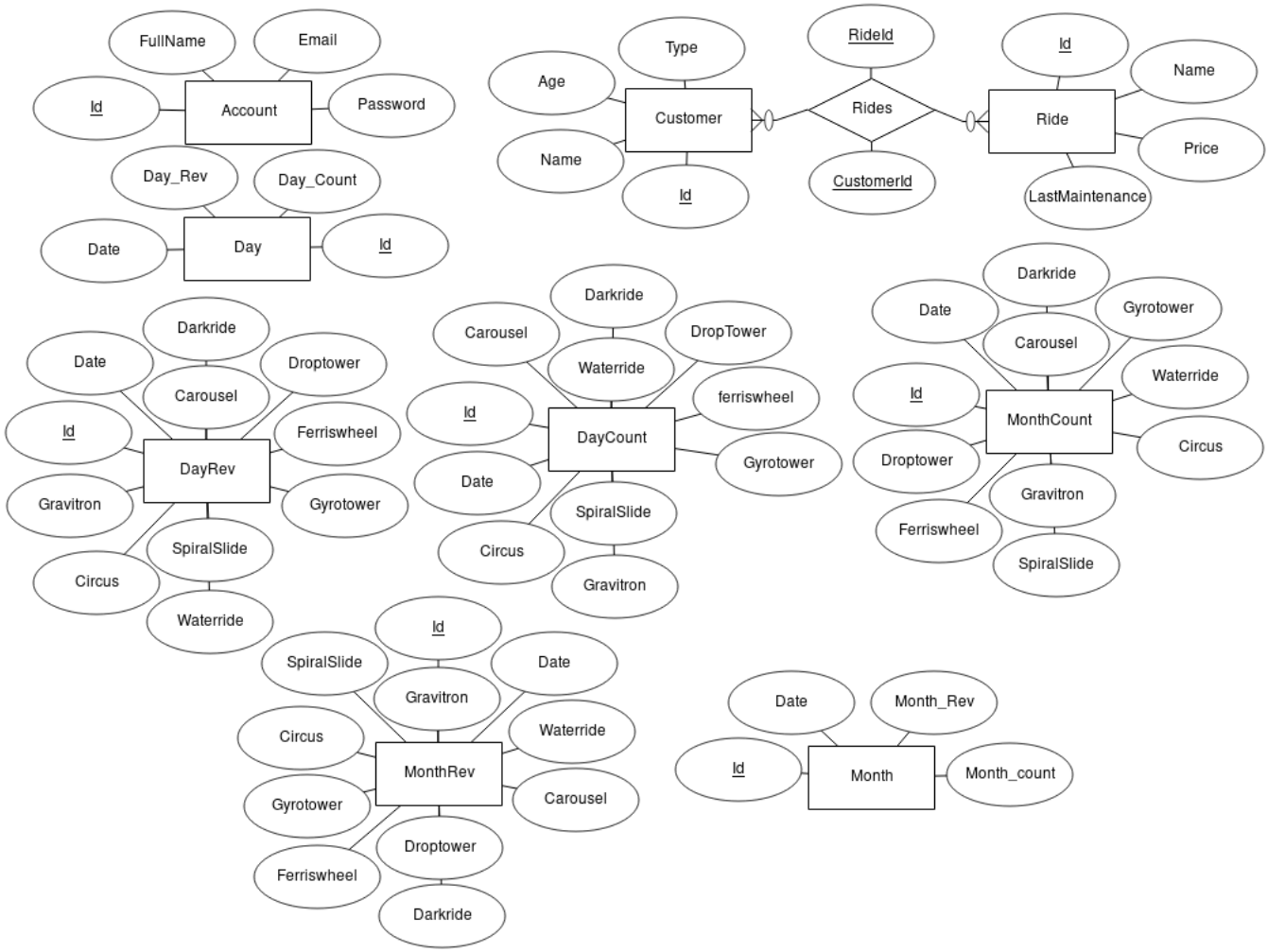
Operating System - Windows/Linux
Database - POSTGRESQL Frameworks - Python Flask
Browser supporting HTML5 and Javascript

Chapter 3

Database Design

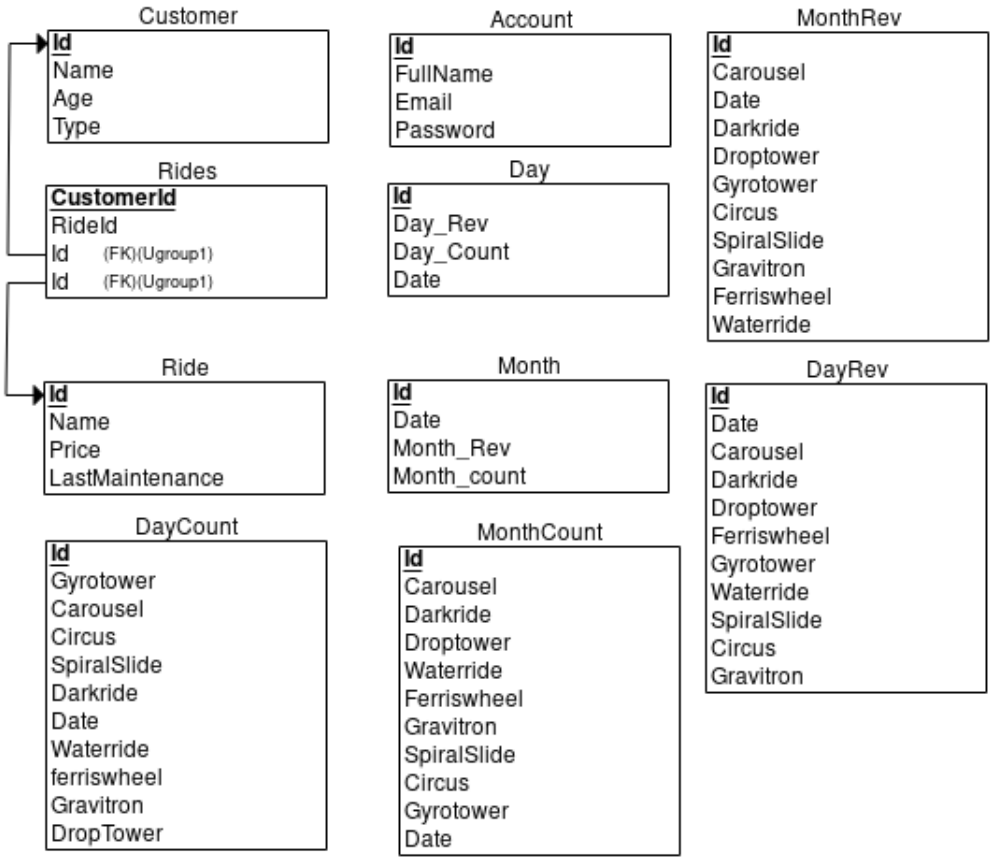
3.1 ER Schema

Figure 3.1: ERD



3.2 Relational Schema

Figure 3.2: Relational Schema



Chapter 4

Description of Tools and Technologies

4.1 PostgreSQL

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. PostgreSQL has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open source community behind the software to consistently deliver performant and innovative solutions. PostgreSQL runs on all major operating systems, has been ACID-compliant since 2001, and has powerful add-ons such as the popular PostGIS geospatial database extender. It is no surprise that PostgreSQL has become the open source relational database of choice for many people and organisations.

4.2 Python Flask

Flask is a microframework for Python, written in Python based on Werkzeug, Jinja 2 and good intentions. It is classified as a microframework because it does not require particular tools or libraries.It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.Flask is commonly used with MongoDB, which gives it more control over databases and history. Applications that use the Flask framework include Pinterest,LinkedIn,and the community web page for Flask itself.

4.3 Flask-SQLAlchemy

SQLAlchemy provides ”a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language”. SQLAlchemy’s philosophy is that relational databases behave less like object collections as the scale gets larger and performance starts being a

concern, while object collections behave less like tables and rows as more abstraction is designed into them. For this reason it has adopted the data mapper pattern (similar to Hibernate for Java) rather than the active record pattern used by a number of other object-relational mappers.However, optional plugins allow users to develop using declarative syntax.

Chapter 5

SQL Database Connectivity

5.1 Opening a Connection

For the common case of having one Flask application all you have to do is to create your Flask application, load the configuration of choice and then create the SQLAlchemy object by passing it the application. Once created, that object then contains all the functions and helpers from both sqlalchemy and sqlalchemy.orm. Furthermore it provides a class called Model that is a declarative base which can be used to declare models:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
 ↪  'database_system://username:password@IPAddress:port/database_name'
db = SQLAlchemy(app)
```

5.2 Closing a Connection

The application context keeps track of the application-level data during a request, CLI command, or other activity. Rather than passing the application around to each function, the `current_app` and g proxies are accessed instead. The application context is created and destroyed as necessary. When a Flask application begins handling a request, it pushes an application context and a request context. When the request ends it pops the request context then the application context. Typically, an application context will have the same lifetime as a request. The application will call functions registered with `teardown_appcontext()` when the application context is popped.

```
@app.teardown_appcontext
def shutdown_session(exception=None):
    db.session.remove()
```

5.3 Executing a Query

The below statement query the relation named ”Customer” and return all the tuples in the form of a list. this object can be used display the appropriate data.

```
customer = Customer.query.all()
```

5.4 Close the Connection

You need to close the session after each request or application context shutdown.

```
@app.teardown_appcontext
def shutdown_session(exception=None):
    db.session.remove()
```

Chapter 6

Implementation

6.1 Code For Major Functionalities

6.1.1 Main Application

```
from flask import Flask, url_for, render_template, g, request,
 ↪  redirect, Markup, json
import os
from flask_sqlalchemy import SQLAlchemy
from models import *
import populateDB
import graph
import pusher

app = Flask(__name__)
pusher_client = pusher.Pusher(
        app_id='643906',
        key='1a2e5fd5d91f28433d49',
        secret='f20f9cbc8cd1cb953e67',
        cluster='ap2',
        ssl=True)
#app.config['SQLALCHEMY_DATABASE_URI'] =
 ↪  'postgres://dbbkiputbafcju:3477b7be42046136fa9d2dec76b7b397933f13
 ↪  14dcbf136a64e1d1288185663a@ec2-54-83-29-34.compute-1.amazonaws.co
 ↪  m:5432/d78tp1vprns7ma?sslmode=require'
app.config['SQLALCHEMY_DATABASE_URI'] =
 ↪  'postgresql://postgres:root@localhost:5432/apms'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
db.init_app(app)

@app.route("/", methods=['GET', 'POST'])
def login():
  error = None
  if request.method == 'POST':
```

```
    inputEmail = request.form["InputEmail"]
    actualPassword = Account.query.filter_by(email=inputEmail).first()
    if request.form['InputPassword'] != str(actualPassword):
      error = "Invalid Credentials."
      return render_template('auth/login.html', error=error)
    else:
      return redirect('/dashboard')
  else:
    return render_template("auth/login.html")

@app.route('/register', methods=['GET', 'POST'])
def register():
  if request.method == "POST":
    inputCredentials = request.form.to_dict()
    user = Account(fullname=inputCredentials["InputName"],
    ↪  email=inputCredentials["InputEmail"],
    ↪  password=inputCredentials["InputPassword"])
    db.session.add(user)
    db.session.commit()
    return redirect(url_for("index"))
  else:
    return render_template("auth/register.html")

@app.route("/dashboard", methods=['GET', 'POST'])
def index():
  daystats = day.query.all()
  dayRideRev = dayrev.query.all()
  dayRideCnt = daycount.query.all()
  customerrides = CustomerRidesLink.query.all()
  if request.method == "POST":
      if request.form['pop'] == 'popcust':
          populateDB.populateCustomer()
      if request.form['pop'] == 'startpark':
          populateDB.populateCustomerRides()
      if request.form['pop'] == 'stoppark':
        populateDB.change()
  return render_template("dashboard.html",dayStats=daystats,dayriderev
  ↪  =dayRideRev,dayridecnt=dayRideCnt,customerRides=customerrides,age
  ↪  Ranges=ageList,Days=days,Hours=hours,dates=datems,dayRevenue=dayr
  ↪  evenue,dayCount=daycunt,rideDates=ridedates,CR=cr,DRR=drr,DTR=dtr
  ↪  ,FWR=fwr,GTR=gtr,RCR=rcr,WRR=wrr,SSR=ssr,CIR=cir,GR=gr,CC=cc,DRC=
  ↪  drc,DTC=dtc,FWC=fwc,GTC=gtc,RCC=rcc,WRC=wrc,SSC=ssc,CIC=cic,GC=gc)

@app.route("/graphs",methods=['GET'])
def getGraph():
```

```
  return render_template("graphs.html",ageRanges=ageList,Days=days,Hou
  ↪ rs=hours,dates=datems,dayRevenue=dayrevenue,dayCount=daycunt,ride
  ↪ Dates=ridedates,CR=cr,DRR=drr,DTR=dtr,FWR=fwr,GTR=gtr,RCR=rcr,WRR
  ↪ =wrr,SSR=ssr,CIR=cir,GR=gr,CC=cc,DRC=drc,DTC=dtc,FWC=fwc,GTC=gtc,
  ↪ RCC=rcc,WRC=wrc,SSC=ssc,CIC=cic,GC=gc)

@app.route("/backend", methods=['POST', 'GET'])
def backend():
  if request.method == "POST":
    customer = request.form["customer"]
    ride = request.form["ride"]
    time_in = datetime.strptime(request.form['time_in'], '%d-%m-%Y
    ↪ %H:%M')
    new_customerride =
    ↪ CustomerRidesLink(customerId=customer,rideId=ride,time=time_in)
    db.session.add(new_customerride)
    db.session.commit()
    data = {
            "id": new_customerride.id,
            "customer": customer,
            "ride": ride,
            "time_in": request.form['time_in']}

    pusher_client.trigger('table', 'new-record', {'data': data })
    return redirect("/dashboard", code=302)
  else:
    crs = db.session.query(CustomerRidesLink,Customer,Ride).filter_by(c
    ↪ ustomerId=Customer.id,rideId=Ride.id).all()
    return render_template('dashboard.html', customers=crs)

@app.route("/edit/<int:id>", methods=['POST', 'GET'])
def update_record(id):
  if request.method == "POST":
    customer = request.form["customer"]
    ride = request.form["ride"]
    time_in = datetime.strptime(request.form['time_in'], '%d-%m-%Y
    ↪ %H:%M')
    update_customer = db.session.query(CustomerRidesLink,Customer,Ride)
    ↪ .filter_by(id=id,customerId=Customer.id,rideId=Ride.id).first()
    update_customer.CustomerRidesLink.customerId = customer
    update_customer.CustomerRidesLink.rideId = ride
    update_customer.CustomerRidesLink.time = time_in
    db.session.commit()
    data = {
            "customer": customer,
            "ride": ride,
            "time_in": request.form['time_in']}
```

```
    pusher_client.trigger('table', 'update-record', {'data': data })
    return redirect("/dashboard", code=302)
  else:
    new_customer = db.session.query(CustomerRidesLink,Customer,Ride).fi
    ↪  lter_by(id=id,customerId=Customer.id,rideId=Ride.id).first()
    print(new_customer)
    new_customer.CustomerRidesLink.time =
    ↪  new_customer.CustomerRidesLink.time.strftime("%d-%m-%Y %H:%M")
    return render_template('update.html', data=new_customer)

@app.route("/delete/<int:id>", methods=['POST','GET'])
def delete_record(id):
    customer =
    ↪  db.session.query(CustomerRidesLink).filter_by(id=id).first()
    db.session.delete(customer)
    db.session.commit()
    return redirect("/dashboard")

@app.teardown_appcontext
def shutdown_session(exception=None):
    db.session.remove()

ageList = graph.getAgeRanges()
datems,dayrevenue,daycunt = graph.getDayStats()
days = len(datems)
ridedates,cr,drr,dtr,fwr,gtr,rcr,wrr,ssr,cir,gr =
↪  graph.getDayRideRevenue()
hours = len(ridedates)
cc,drc,dtc,fwc,gtc,rcc,wrc,ssc,cic,gc = graph.getDayRideCount()

if __name__ == '__main__':
  app.run(host="localhost",port=5010, debug=True)
```

6.1.2 Models

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_script import Manager
from flask_migrate import Migrate, MigrateCommand
from datetime import datetime
from alembic import op
import populateDB


app = Flask(__name__)
```

```
app.config['SQLALCHEMY_DATABASE_URI'] =
↪  'postgresql://postgres:root@localhost:5432/apms'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
db.init_app(app)
migrate = Migrate(app, db)

manager = Manager(app)
manager.add_command('db', MigrateCommand)

class Account(db.Model):
    __tablename__ = 'account'
    id = db.Column(db.Integer, primary_key=True)
    fullname = db.Column(db.String(128), nullable=False)
    email = db.Column(db.String(128), unique=True, nullable=False)
    password = db.Column(db.String(128), nullable=False)

    def __repr__(self):
        return self.password

class Customer(db.Model):
    __tablename__ = 'customer'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128), nullable=False)
    age = db.Column(db.Integer, nullable=False)
    type = db.Column(db.String(128), nullable=False)
    rides = db.relationship('CustomerRidesLink')

class Ride(db.Model):
    __tablename__ = 'ride'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128), unique=True, nullable=False)
    price = db.Column(db.Integer, nullable=False)
    maintenance_cost = db.Column(db.Integer, nullable=False)

class CustomerRidesLink(db.Model):
    __tablename__ = 'customerrides'
    id = db.Column(db.Integer, primary_key=True)
    customerId = db.Column(db.Integer, db.ForeignKey('customer.id'))
    rideId = db.Column(db.Integer, db.ForeignKey('ride.id'))
    customer = db.relationship('Customer')
    ride = db.relationship('Ride')
    time = db.Column(db.DateTime, nullable=False,
    ↪  default=datetime.utcnow)

class day(db.Model):
    __tablename__ = 'daydetails'
```

```
    id = db.Column(db.Integer, primary_key=True)
    time = db.Column(db.DateTime, nullable=False,
    ↪  default=datetime.utcnow)
    day_rev = db.Column(db.Integer)
    day_count = db.Column(db.Integer)

class dayrev(db.Model):
    __tablename__ = 'dayrev'
    id = db.Column(db.Integer, primary_key=True)
    time = db.Column(db.DateTime, default=datetime.utcnow, nullable =
    ↪  False)
    Carousel = db.Column(db.Integer)
    Darkride = db.Column(db.Integer)
    Droptower = db.Column(db.Integer)
    Ferriswheel = db.Column(db.Integer)
    Gyrotower = db.Column(db.Integer)
    Rollercoaster = db.Column(db.Integer)
    Waterride = db.Column(db.Integer)
    SpiralSlide = db.Column(db.Integer)
    Circus = db.Column(db.Integer)
    Gravitron = db.Column(db.Integer)

class daycount(db.Model):
    __tablename__ = 'daycount'
    id = db.Column(db.Integer, primary_key=True)
    time = db.Column(db.DateTime, default=datetime.utcnow, nullable =
    ↪  False)
    Carousel = db.Column(db.Integer)
    Darkride = db.Column(db.Integer)
    Droptower = db.Column(db.Integer)
    Ferriswheel = db.Column(db.Integer)
    Gyrotower = db.Column(db.Integer)
    Rollercoaster = db.Column(db.Integer)
    Waterride = db.Column(db.Integer)
    SpiralSlide = db.Column(db.Integer)
    Circus = db.Column(db.Integer)
    Gravitron = db.Column(db.Integer)

class month(db.Model):
    __tablename__ = 'monthdetails'
    id = db.Column(db.Integer, primary_key=True)
    time = db.Column(db.DateTime, nullable=False,
    ↪  default=datetime.utcnow)
    month_rev = db.Column(db.Integer)
    month_count = db.Column(db.Integer)

class monthrev(db.Model):
```

```
    __tablename__ = 'monthrev'
    id = db.Column(db.Integer, primary_key=True)
    time = db.Column(db.DateTime, default=datetime.utcnow, nullable =
    ↪  False)
    Carousel = db.Column(db.Integer)
    Darkride = db.Column(db.Integer)
    Droptower = db.Column(db.Integer)
    Ferriswheel = db.Column(db.Integer)
    Gyrotower = db.Column(db.Integer)
    Rollercoaster = db.Column(db.Integer)
    Waterride = db.Column(db.Integer)
    SpiralSlide = db.Column(db.Integer)
    Circus = db.Column(db.Integer)
    Gravitron = db.Column(db.Integer)

class monthcount(db.Model):
    __tablename__ = 'monthcount'
    id = db.Column(db.Integer, primary_key=True)
    time = db.Column(db.DateTime, default=datetime.utcnow, nullable =
    ↪  False)
    Carousel = db.Column(db.Integer)
    Darkride = db.Column(db.Integer)
    Droptower = db.Column(db.Integer)
    Ferriswheel = db.Column(db.Integer)
    Gyrotower = db.Column(db.Integer)
    Rollercoaster = db.Column(db.Integer)
    Waterride = db.Column(db.Integer)
    SpiralSlide = db.Column(db.Integer)
    Circus = db.Column(db.Integer)
    Gravitron = db.Column(db.Integer)

@manager.command
def seed():
    populateDB.populateRide()

if __name__=='__main__':
    manager.run()
```

6.1.3 Populate Database

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from faker import Faker
import models
import random
from datetime import datetime, timedelta
```

```
import time

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
↪  'postgresql://postgres:root@localhost:5432/apms'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
db.init_app(app)

fake = Faker()

def change():
    global a
    a = 0

def populateCustomerRides():
    global a
    global tim
    a = 1
    customerList = models.Customer.query.all()
    rideList = models.Ride.query.all()

    while True:
        if a is 0:
            break
        customer = random.choice(customerList)
        ride = random.choice(rideList)
        newTime = tim + timedelta(0,1800) # days, seconds, then other
        ↪  fields.
        if newTime.hour == 18:
            tim =
            ↪  datetime(newTime.year,newTime.month,newTime.day+1,10,00)
            time.sleep(5)
            if newTime.day == 28:
                time.sleep(3)
                if newTime.month < 12:
                    tim = datetime(newTime.year,newTime.month+1,1,10,00)
                if newTime.month == 12:
                    tim = datetime(newTime.year+1,1,1,10,00)
        else:
            tim = newTime
        customerride = models.CustomerRidesLink(customerId=customer.id,
        ↪  rideId=ride.id,time=newTime)
        db.session.add(customerride)
        db.session.commit()
        time.sleep(3)
```

```
def populateRide():
    ride_list =
    ↪  ['Carousel','Darkride','Droptower','Ferriswheel','Gyrotower','R
    ↪  ollercoaster','Waterride','SpiralSlide','Circus','Gravitron']
    price_list = [200, 300, 200, 250, 200, 350, 300, 150, 200, 250]
    x = 0
    for ride in ride_list:
        name = ride
        price = price_list[x]
        x += 1
        maintenance_cost = (random.randint(20,30)) * 100
        rideDetails = models.Ride(name=name,price=price,maintenance_cos
        ↪  t=maintenance_cost)
        db.session.add(rideDetails)
        db.session.commit()

def populateCustomer():
    count = 100
    type_list = ['normal','student']
    age10 = list(range(5,10))
    age20 = list(range(11,19))
    age30 = list(range(20,30))
    age50 = list(range(31,50))
    age70 = list(range(51,70))
    age100 = list(range(71,100))
    wr = age10*70+age20*20+age30*5+age50*3+age70*1+age100*1
    while count > 0:
        name = fake.name()
        age = random.choice(wr)
        type = 'normal'
        if age <= 10:
            type = 'child'
        elif age >= 60:
            type = 'senior'
        else:
            type = random.choice(type_list)
        customer = models.Customer(name=name,age=age,type=type)
        db.session.add(customer)
        db.session.commit()
        count -= 1

a = 1
tim = datetime(2018,12,26,10,0)
```

6.2 Procedures

6.2.1 Consolidate day statistics

```
CREATE OR REPLACE FUNCTION daydcfill ()
RETURNS TRIGGER AS $$
DECLARE
    totalcount INTEGER := 0;
    Carouselcount INTEGER := 0 ;
    Darkridecount INTEGER := 0 ;
    Droptowercount INTEGER := 0 ;
    Ferriswheelcount INTEGER := 0 ;
    Gyrotowercount INTEGER := 0 ;
    Rollercoastercount INTEGER := 0 ;
    Waterridecount INTEGER := 0 ;
    SpiralSlidecount INTEGER := 0 ;
    Circuscount INTEGER := 0 ;
    Gravitroncount INTEGER := 0 ;

BEGIN
    Carouselcount := new."Carousel";
    Darkridecount := new."Darkride";
    Droptowercount := new."Droptower";
    Ferriswheelcount := new."Ferriswheel";
    Gyrotowercount := new."Gyrotower";
    Rollercoastercount := new."Rollercoaster";
    Waterridecount := new."Waterride";
    SpiralSlidecount := new."SpiralSlide";
    Circuscount := new."Circus";
    Gravitroncount := new."Gravitron";
    totalcount := Carouselcount + Darkridecount + Droptowercount +
    ↪  Ferriswheelcount + Gyrotowercount + Rollercoastercount +
    ↪  Waterridecount + SpiralSlidecount + Circuscount +
    ↪  Gravitroncount;
    insert into daydetails("time", "day_rev", "day_count") values
    ↪  (new.time, 0, totalcount);
    RETURN new;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER update_daydccount
    AFTER INSERT ON daycount
    FOR EACH ROW
    EXECUTE PROCEDURE daydcfill();

CREATE OR REPLACE FUNCTION daydrfill ()
RETURNS TRIGGER AS $$
```

```
DECLARE
    totalprice INTEGER := 0;
    Carouselprice INTEGER := 0 ;
    Darkrideprice INTEGER := 0 ;
    Droptowerprice INTEGER := 0 ;
    Ferriswheelprice INTEGER := 0 ;
    Gyrotowerprice INTEGER := 0 ;
    Rollercoasterprice INTEGER := 0 ;
    Waterrideprice INTEGER := 0 ;
    SpiralSlideprice INTEGER := 0 ;
    Circusprice INTEGER := 0 ;
    Gravitronprice INTEGER := 0 ;

BEGIN
    Carouselprice := new."Carousel";
    Darkrideprice := new."Darkride";
    Droptowerprice := new."Droptower";
    Ferriswheelprice := new."Ferriswheel";
    Gyrotowerprice := new."Gyrotower";
    Rollercoasterprice := new."Rollercoaster";
    Waterrideprice:= new."Waterride";
    SpiralSlideprice := new."SpiralSlide";
    Circusprice := new."Circus";
    Gravitronprice := new."Gravitron";
    totalprice := Carouselprice + Darkrideprice + Droptowerprice +
    ↪  Ferriswheelprice + Gyrotowerprice + Rollercoasterprice +
    ↪  Waterrideprice + SpiralSlideprice + Circusprice +
    ↪  Gravitronprice;
    update daydetails set day_rev = totalprice where id = new.id;
    RETURN new;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER update_daydrcount
    AFTER INSERT ON dayrev
    FOR EACH ROW
    EXECUTE PROCEDURE daydrfill();

CREATE OR REPLACE FUNCTION dayrevfill ()
RETURNS TRIGGER AS $$
DECLARE
    Carouselprice INTEGER := 0 ;
    Darkrideprice INTEGER := 0 ;
    Droptowerprice INTEGER := 0 ;
    Ferriswheelprice INTEGER := 0 ;
    Gyrotowerprice INTEGER := 0 ;
    Rollercoasterprice INTEGER := 0 ;
```

```
    Waterrideprice INTEGER := 0 ;
    SpiralSlideprice INTEGER := 0 ;
    Circusprice INTEGER := 0 ;
    Gravitronprice INTEGER := 0 ;
    Carouselprice1 INTEGER := 0 ;
    Darkrideprice1 INTEGER := 0 ;
    Droptowerprice1 INTEGER := 0 ;
    Ferriswheelprice1 INTEGER := 0 ;
    Gyrotowerprice1 INTEGER := 0 ;
    Rollercoasterprice1 INTEGER := 0 ;
    Waterrideprice1 INTEGER := 0 ;
    SpiralSlideprice1 INTEGER := 0 ;
    Circusprice1 INTEGER := 0 ;
    Gravitronprice1 INTEGER := 0 ;
BEGIN
    Carouselprice := count(*) from customerrides where "rideId" = 1;
    Carouselprice1 := r.price from ride r where id = 1;
    Carouselprice := Carouselprice * Carouselprice1;

    Darkrideprice := count(*) from customerrides where "rideId" = 2;
    Darkrideprice1 := r.price from ride r where id = 2;
    Darkrideprice := Darkrideprice * Darkrideprice1;

    Droptowerprice := count(*) from customerrides where "rideId" = 3;
    Droptowerprice1 := r.price from ride r where id = 3;
    Droptowerprice := Droptowerprice * Droptowerprice1;

    Ferriswheelprice := count(*) from customerrides where "rideId" = 4;
    Ferriswheelprice1 := r.price from ride r where id = 4;
    Ferriswheelprice := Ferriswheelprice * Ferriswheelprice1;

    Gyrotowerprice := count(*) from customerrides where "rideId" = 5;
    Gyrotowerprice1 := r.price from ride r where id = 5;
    Gyrotowerprice := Gyrotowerprice * Gyrotowerprice1;

    Rollercoasterprice := count(*) from customerrides where "rideId" =
    ↪  6;
    Rollercoasterprice1 := r.price from ride r where id = 6;
    Rollercoasterprice := Rollercoasterprice * Rollercoasterprice1;

    Waterrideprice := count(*) from customerrides where "rideId" = 7;
    Waterrideprice1 := r.price from ride r where id = 7;
    Waterrideprice := Waterrideprice * Waterrideprice1;

    SpiralSlideprice := count(*) from customerrides where "rideId" = 8;
    SpiralSlideprice1 := r.price from ride r where id = 8;
    SpiralSlideprice := SpiralSlideprice * SpiralSlideprice1;
```

```
    Circusprice := count(*) from customerrides where "rideId" = 9;
    Circusprice1 := r.price from ride r where id = 9;
    Circusprice := Circusprice * Circusprice1;

    Gravitronprice := count(*) from customerrides where "rideId" = 10;
    Gravitronprice1 := r.price from ride r where id = 10;
    Gravitronprice := Gravitronprice * Gravitronprice1;

    insert into dayrev("time", "Carousel", "Darkride", "Droptower",
     ↪  "Ferriswheel", "Gyrotower", "Rollercoaster", "Waterride",
     ↪  "SpiralSlide", "Circus", "Gravitron") values (new.time,
     ↪  Carouselprice, Darkrideprice, Droptowerprice, Ferriswheelprice,
     ↪  Gyrotowerprice, Rollercoasterprice, Waterrideprice,
     ↪  SpiralSlideprice, Circusprice, Gravitronprice);
    delete from customerrides;
    alter sequence customerrides_id_seq restart with 1;
    RETURN new;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER update_dayrev
    AFTER INSERT ON customerrides
    FOR EACH ROW
    WHEN ( extract (hour from new.time ) = 18)
    EXECUTE PROCEDURE dayrevfill();

    CREATE OR REPLACE FUNCTION daycountfill ()
    RETURNS TRIGGER AS $$
    DECLARE
        Carouselcount INTEGER := 0 ;
        Darkridecount INTEGER := 0 ;
        Droptowercount INTEGER := 0 ;
        Ferriswheelcount INTEGER := 0 ;
        Gyrotowercount INTEGER := 0 ;
        Rollercoastercount INTEGER := 0 ;
        Waterridecount INTEGER := 0 ;
        SpiralSlidecount INTEGER := 0 ;
        Circuscount INTEGER := 0 ;
        Gravitroncount INTEGER := 0 ;

    BEGIN
        Carouselcount := count(*) from customerrides where "rideId" =
         ↪  1;
        Darkridecount := count(*) from customerrides where "rideId" =
         ↪  2;
```

```
        Droptowercount := count(*) from customerrides where "rideId" =
        ↪  3;
        Ferriswheelcount := count(*) from customerrides where "rideId"
        ↪  = 4;
        Gyrotowercount := count(*) from customerrides where "rideId" =
        ↪  5;
        Rollercoastercount := count(*) from customerrides where
        ↪  "rideId" = 6;
        Waterridecount := count(*) from customerrides where "rideId" =
        ↪  7;
        SpiralSlidecount := count(*) from customerrides where "rideId"
        ↪  = 8;
        Circuscount := count(*) from customerrides where "rideId" = 9;
        Gravitroncount := count(*) from customerrides where "rideId" =
        ↪  10;

        insert into daycount("time", "Carousel", "Darkride",
        ↪  "Droptower", "Ferriswheel", "Gyrotower", "Rollercoaster",
        ↪  "Waterride", "SpiralSlide", "Circus", "Gravitron") values
        ↪  (new.time, Carouselcount, Darkridecount, Droptowercount,
        ↪  Ferriswheelcount, Gyrotowercount, Rollercoastercount,
        ↪  Waterridecount, SpiralSlidecount, Circuscount,
        ↪  Gravitroncount);
        RETURN new;
    END;$$
    LANGUAGE plpgsql;

    CREATE TRIGGER update_daycount
        AFTER INSERT ON customerrides
        FOR EACH ROW
        WHEN ( extract (hour from new.time ) = 18)
        EXECUTE PROCEDURE daycountfill();
```

6.2.2 Consolidate month statistics

```
CREATE OR REPLACE FUNCTION monthcountfill ()
RETURNS TRIGGER AS $$
DECLARE
    Carouselcount INTEGER := 0 ;
    Darkridecount INTEGER := 0 ;
    Droptowercount INTEGER := 0 ;
    Ferriswheelcount INTEGER := 0 ;
    Gyrotowercount INTEGER := 0 ;
    Rollercoastercount INTEGER := 0 ;
    Waterridecount INTEGER := 0 ;
    SpiralSlidecount INTEGER := 0 ;
```

```
    Circuscount INTEGER := 0 ;
    Gravitroncount INTEGER := 0 ;

BEGIN
    Carouselcount := sum("Carousel") from daycount;
    Darkridecount := sum("Darkride") from daycount;
    Droptowercount := sum("Droptower") from daycount;
    Ferriswheelcount := sum("Ferriswheel") from daycount;
    Gyrotowercount := sum("Gyrotower") from daycount;
    Rollercoastercount := sum("Rollercoaster") from daycount;
    Waterridecount := sum("Waterride") from daycount;
    SpiralSlidecount := sum("SpiralSlide") from daycount;
    Circuscount := sum("Circus") from daycount;
    Gravitroncount := sum("Gravitron") from daycount;

    insert into monthcount("time", "Carousel", "Darkride",
    ↪  "Droptower", "Ferriswheel", "Gyrotower", "Rollercoaster",
    ↪  "Waterride", "SpiralSlide", "Circus", "Gravitron") values
    ↪  (new.time, Carouselcount, Darkridecount, Droptowercount,
    ↪  Ferriswheelcount, Gyrotowercount, Rollercoastercount,
    ↪  Waterridecount, SpiralSlidecount, Circuscount, Gravitroncount);
    RETURN new;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER update_monthcount
    AFTER INSERT ON daycount
    FOR EACH ROW
    WHEN ( extract (day from new.time ) = 28)
    EXECUTE PROCEDURE monthcountfill();

CREATE OR REPLACE FUNCTION monthrevfill ()
RETURNS TRIGGER AS $$
DECLARE
    Carouselcount INTEGER := 0 ;
    Darkridecount INTEGER := 0 ;
    Droptowercount INTEGER := 0 ;
    Ferriswheelcount INTEGER := 0 ;
    Gyrotowercount INTEGER := 0 ;
    Rollercoastercount INTEGER := 0 ;
    Waterridecount INTEGER := 0 ;
    SpiralSlidecount INTEGER := 0 ;
    Circuscount INTEGER := 0 ;
    Gravitroncount INTEGER := 0 ;

BEGIN
    Carouselcount := sum("Carousel") from dayrev;
```

```
    Darkridecount := sum("Darkride") from dayrev;
    Droptowercount := sum("Droptower") from dayrev;
    Ferriswheelcount := sum("Ferriswheel") from dayrev;
    Gyrotowercount := sum("Gyrotower") from dayrev;
    Rollercoastercount := sum("Rollercoaster") from dayrev;
    Waterridecount := sum("Waterride") from dayrev;
    SpiralSlidecount := sum("SpiralSlide") from dayrev;
    Circuscount := sum("Circus") from dayrev;
    Gravitroncount := sum("Gravitron") from dayrev;

    insert into monthrev("time", "Carousel", "Darkride", "Droptower",
    ↪  "Ferriswheel", "Gyrotower", "Rollercoaster", "Waterride",
    ↪  "SpiralSlide", "Circus", "Gravitron") values (new.time,
    ↪  Carouselcount, Darkridecount, Droptowercount, Ferriswheelcount,
    ↪  Gyrotowercount, Rollercoastercount, Waterridecount,
    ↪  SpiralSlidecount, Circuscount, Gravitroncount);
    delete from dayrev;
    delete from daydetails;
    delete from daycount;
    alter sequence dayrev_id_seq restart with 1;
    alter sequence daydetails_id_seq restart with 1;
    alter sequence daycount_id_seq restart with 1;
    RETURN new;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER update_monthrev
    AFTER INSERT ON dayrev
    FOR EACH ROW
    WHEN ( extract (day from new.time ) = 28)
    EXECUTE PROCEDURE monthrevfill();

CREATE OR REPLACE FUNCTION monthmcfill ()
RETURNS TRIGGER AS $$
DECLARE
    totalcount INTEGER := 0;
    Carouselcount INTEGER := 0 ;
    Darkridecount INTEGER := 0 ;
    Droptowercount INTEGER := 0 ;
    Ferriswheelcount INTEGER := 0 ;
    Gyrotowercount INTEGER := 0 ;
    Rollercoastercount INTEGER := 0 ;
    Waterridecount INTEGER := 0 ;
    SpiralSlidecount INTEGER := 0 ;
    Circuscount INTEGER := 0 ;
    Gravitroncount INTEGER := 0 ;
```

```
BEGIN
    Carouselcount := new."Carousel";
    Darkridecount := new."Darkride";
    Droptowercount := new."Droptower";
    Ferriswheelcount := new."Ferriswheel";
    Gyrotowercount := new."Gyrotower";
    Rollercoastercount := new."Rollercoaster";
    Waterridecount := new."Waterride";
    SpiralSlidecount := new."SpiralSlide";
    Circuscount := new."Circus";
    Gravitroncount := new."Gravitron";
    totalcount := Carouselcount + Darkridecount + Droptowercount +
    ↪  Ferriswheelcount + Gyrotowercount + Rollercoastercount +
    ↪  Waterridecount + SpiralSlidecount + Circuscount +
    ↪  Gravitroncount;
    insert into monthdetails("time", "month_rev", "month_count")
    ↪  values (new.time, 0, totalcount);
    RETURN new;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER update_monthmccount
    AFTER INSERT ON monthcount
    FOR EACH ROW
    EXECUTE PROCEDURE monthmcfill();

CREATE OR REPLACE FUNCTION monthmrfill ()
RETURNS TRIGGER AS $$
DECLARE
    totalprice INTEGER := 0;
    Carouselprice INTEGER := 0 ;
    Darkrideprice INTEGER := 0 ;
    Droptowerprice INTEGER := 0 ;
    Ferriswheelprice INTEGER := 0 ;
    Gyrotowerprice INTEGER := 0 ;
    Rollercoasterprice INTEGER := 0 ;
    Waterrideprice INTEGER := 0 ;
    SpiralSlideprice INTEGER := 0 ;
    Circusprice INTEGER := 0 ;
    Gravitronprice INTEGER := 0 ;

BEGIN
    Carouselprice := new."Carousel";
    Darkrideprice := new."Darkride";
    Droptowerprice := new."Droptower";
    Ferriswheelprice := new."Ferriswheel";
    Gyrotowerprice := new."Gyrotower";
```

```
    Rollercoasterprice := new."Rollercoaster";
    Waterrideprice:= new."Waterride";
    SpiralSlideprice := new."SpiralSlide";
    Circusprice := new."Circus";
    Gravitronprice := new."Gravitron";
    totalprice := Carouselprice + Darkrideprice + Droptowerprice +
     ↪  Ferriswheelprice + Gyrotowerprice + Rollercoasterprice +
     ↪  Waterrideprice + SpiralSlideprice + Circusprice +
     ↪  Gravitronprice;
    update monthdetails set month_rev = totalprice where id = new.id;
    RETURN new;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER update_monthmrcount
    AFTER INSERT ON monthrev
    FOR EACH ROW
    EXECUTE PROCEDURE monthmrfill();
```

Chapter 7

Snapshots

Chapter 8

Conclusion and Future Enhancements

This project is a rather rudimentary implementation of a larger system. We have been successfull in automating the core objective of this project.

Enhancements that we would like to explore in the future:

- Implementing regression analysis to predict future trends of revenue.
- Implement threads to enable visitor inflow and statistical analysis.
- Enhance the graph plotting algorithm to access the database directly.

References

[1] Fundamentals of database systems by (Elmasri Navathe, 2000)

[2] Flask-SQLAlchemy: `http://flask-sqlalchemy.pocoo.org/2.3/`

[3] Postgresql: `http://www.postgresqltutorial.com/`

[4] JavaScript charting: `https://www.chartjs.org/`, `https://www.amcharts.com/`