

IT214



Dhirubhai Ambani
Institute of Information and Communication Technology

Team ID: S7 T8

Entertainment Event booking System

Instructor: Prof. Minal Bhise

Mentor Name: Manoj Sir

ID	Name
201901004	Axit Dhola
201901031	Sanny Dhameliya
201901049	Bhadrayu Bhalodia
201901066	Kenil Bhingradiya

Table of Contents

Section 1: SRS	2
Section 2: Noun Analysis	23
Section 3: ER diagram	31
Section 4: Final ER-Diagram to Relational Model	35
Section 5: Normalization and Schema Refinement	38
Section 6: SQL: DDL Scripts, Insert statements, 40 SQL Queries	47
Section 7: Project Code with output screenshots	91

Section 1: SRS

Software Requirement Specification

INDEX

A. Understand & Complete the Description of the Case Study/Problem Domain.....	5
1.1 Purpose	5
1.2 Intended Audience and Reading Suggestions	5
1.3 Product Scope	5
1.4 Description	6
1.4.1 Product Perspective	6
1.4.2 Product Function	7
1.4.3 User Characteristics	8
B. Document the Requirements Collection/ Fact-Finding Phase	8
2.1 Background Reading	8
2.2 Interviews	8
2.2.1 Combined Requirements gathered from the Interviews	12
2.3 Questionnaires	13
2.4 Observations	18
C. Create Fact-Finding Chart	19
D. List Requirements	19
4.1 Non-Functional Requirements	19
4.2 Functional Requirements	20
E. User Class And Characteristics	21
F. Operating Environment	21
G. Product Function	22
H. Privileges	22
I. Assumptions	22
J. BusinesConstraints	23

[A] Understand & Complete the Description of the Case Study/Problem Domain

Database Name: - BOOK_SYS

1.1 Purpose: -

The primary purpose of this document is to manage a database of multiple events simultaneously, and in one place, so it is easy to track and manage for customers and managers in real-time.

There are many benefits of using booking systems for events, and online bookings are the future. Let the event booking system do all of the hard work.

The software will automatically update in real-time so that you can keep track of the number of presents and upcoming events, tickets that you have sold, and how many available slots are left; selection of tickets and payment can be done with ease.

In addition, developers can use this doc as a source and guide.

1.2 Intended Audience and Reading Suggestions:-

The main intention of this document is,

- **Database owner (Administrator):** They are the admin of the database, and they can add/remove events, managers, users.
- **Event managers:** These will be the primary users. They can add/remove customers, update any information of the customers, etc. In short, they are the manager of the database.
- **Event organizers:** They are the ones who organize new events or can add/edit old events. They can add or update event information in the database.
- **Customer:** They can only view information related to events, and if interested, they can book event tickets.

1.3 Product Scope:-

The database related product is helpful for,

- Provide facility for booking of all type of entertainment events
- Inquiry facility
- Provide instant ticket booking
- Manage user and maintain the database for all booking and transaction
- Reduction of traffic at offline booking centers
- Booking many tickets at the same time by different people

1.4 Description :

- The main motto of this database is to register different types of events and organize them easily using this platform. Also, for other users to search various events from one platform and book if they are interested. This database is also for managers to manage the registration process to manage and access easily.
- This platform brings event owners and customers nearer and able to contact them directly.
- Design must be able to handle large data uploading loads as well as keep data updating in real-time, so that will be very useful for users.
- In order to update the database information, which comes from sources like events organizers, movie theaters, which access the database and keep updating the number of users, seat availability.
- This platform is user-friendly, easy to learn, and reliable, and it also works on any multi-platforms, so this will work on all standard platforms like windows, android, iOS, Linux.
- The users can easily find available seats and reserve them. They can pay through various payment systems. Users can also cancel their bookings and also get a refund if canceled at the proper time.
- Using this database, event organizers easily find information about total confirmed tickets and details about all users who are willing to come to their events.
- Our database integrity is very powerful, so without any user concern, we will not store any private or financial data and not share it with others like the government or any other platforms.
- This module allows customers to make payments after the reservation. The customer will receive a ticket receipt by mail after the payment.
They Make payment to selected service, Generate billing receipt and Update credit card details and also View payment details when he/she wants.
- With the help of these databases, for each customer, we can get to know about positive and negative points with the help of personalized feedback reports.
- Event organizers can host various events on platforms and they can easily find information about customers, how many seats are remaining.
- This module allows the administrator of this site to manage the different aspects of ticket booking-related activities. The administrator of the site can add or modify the records of concerts, movies, dramas, plays, etc. they should be added, edit, view concerts records, movie details, and also View bookings, View cancellation reports, View payment reports, Add users and Change passwords.
- This module is for customers where customers can access the account module after login id and password. This module will display booked tickets, Billing receipts, ticket cancellation page, logout, etc. Users can Display booked receipts, Change passwords, and update profile information.

1.4.1 Product Perspective:-

- Product Perspective BOOK_SYS Online entertainment-related booking Reservation System is aimed to provide a Simple, Quick, and efficient Way to Book an entertainment-related Ticket.
- This System Should be User friendly, smooth, Fast, secure, reliable, and hassle-free to provide the user a pleasant Experience by Booking a Ticket.
- BOOK_SYS is intended to be a Stand-alone Online Portal and should be independent of any Other Software. It should be compatible With any Browser and Should be Platform Independent. SRS for BOOK_SYS System.

1.4.2 Product Functions:-

1. User Management
 - New User Registration
 - Existing User Login/Logout
 - Forgot the Password Recovery mechanism
 - Recovery Via Phone
 - Recovery Via Mail.
2. Enquiries
 - Seat Availability
 - Current event Information
 - Ticket Related Enquiries
3. Booking Facilities.
 - Booking a Ticket.
 - Booking a general Ticket.
 - Booking History.
 - Ticket Reprint Facility.
 - Booking Status/Current Status
 - Providing customer Details.
 - Personal Details.
 - ID details.
 - Type of seats
4. Payment Options.
 - Credit Cards.
 - Debit Cards.
 - Mobile/Net Banking.
 - Cash.
5. Cancellation and Refund Facilities.
 - Cancel a ticket.
 - Print Cancelled Ticket Facility.
 - Cancellation Status.
 - Canceled Ticket History.
 - Refunds Status.
 - Refunds History.
6. Event types
 - Event name
 - Event total cost
 - Event ticket price and availability
 - Event Date

1.4.3 USER CHARACTERISTICS

- The User Shall be familiar With the Procedure and Provisions to Book a ticket.
- The User shall have an average knowledge of Computer and Internet.
- The User should know the methods of transactions and Payment.

[B]. Document the Requirements Collection/ Fact-Finding Phase

2.1 Background readings

1. In this section, we collected data from various event booking sites and articles like
 - a. www.bookmyshow.com
 - b. <https://pdfcoffee.com/srs-on-online-booking-system-pdf-free.html>
 - c. www.zoominfo.com
 - d. <https://www.accelevents.com/platform/>
 - e. <https://play.google.com/store/apps/details?id=com.bt.bms> (ratings and reviews)
2. We also collected customer reviews from different booking sites and also generated a google form.
3. For changes in anything then the website needs accurate and real-time data for Example
 - Available number of events
 - Number of customers region-wise.
 - Customers are fully vaccinated or not.
 - Food availability region-wise.
 - An available number of employees.
4. Some tips to improve situations
 - India's population is very high so Needs some fast servers that can handle many requests at a time.
 - There has to be total data of events with no chances of missing any kind of event-related data. It is very important for customer satisfaction.
 - Lack of understanding between customer and event organizer.
 - Better communication between customer and help-center or organizer.

2.2 Interviews: -

Interview 1

- Role Play interview plan
- System: Entertainment booking System
- Interviewee: Dhruvil Gorasiya(Role Play)
 - Designation: Administrator of Event
- Interviewer:
 - Axit Dhola (Student of DAIICT)
 - Sanny Dhameliya (Student of DAIICT)
 - Bhadrayu Bhalodia (Student of DAIICT)
 - Kenil Bhingradya (Student of DAIICT)

- Date: 04/10/2021
- Time: 16:30
- Duration: 30 minutes
- Place: Google Meet
- Purpose of Interview:
 - Preliminary meeting to identify problems and requirements regarding decisions of the Manager about Event.
- Agenda:
 - Problems with the lack of Management
 - Current Situation about booking System
 - Initial ideas
 - Follow-up actions

Interview Summary 1

- System: Entertainment booking System
- Interviewee: Dhruvil Gorasiya (Role Play)
 - Designation: Administrator of Event
- Interviewer:
 - Axit Dhola (Student of DAIICT)
 - Sanny Dhameliya (Student of DAIICT)
 - Bhadrayu Bhalodia (Student of DAIICT)
 - Kenil Bhingradya (Student of DAIICT)
- Date: 04/10/2021
- Time: 16:30
- Duration: 30 minutes
- Place: Google Meet
- Summary of Interview
 - Preliminary meeting to identify problems and requirements regarding decisions of the Manager about Booking Management.
 - Also, users watch trailers of currently showing and upcoming movies, read reviews, synopsis, the cast of currently showing and forthcoming movie/ theatre/ concerts, and view and provide theatre/ venue review, user ratings, etc...
 - In the time of Covid-19, all things move on to online platforms and all the things like education, work, and all this stuff convert into online mode. Also, people promoted online work without the crowd of people.
 - The problem of lack of understanding between event managers and customers which now is solved by enjoying their event without buying tickets to spend several hours in a queue of people.

Interview 2

- Role Play interview plan
- System: Entertainment booking System
- Interviewee: Yash Sakaria (Role Play)
 - Designation: Event Organizer
- Interviewer:
 - Axit Dhola (Student of DAIICT)
 - Sanny Dhameliya (Student of DAIICT)

- Bhadrayu Bhalodia (Student of DAIICT)
 - Kenil Bhingradiya (Student of DAIICT)
- Date: 05/10/2021
- Time: 17:30
- Duration: 30 minutes
- Place: Google Meet
- Purpose of Interview:
 - Preliminary meeting to identify problems and requirements regarding decisions of the Manager about Event.
- Agenda:
 - Problems with the lack of Management
 - Current Situation about booking System
 - Initial ideas
 - Follow-up actions

Interview Summary 2

- System: Entertainment booking System
- Interviewee: Yash Sakaria (Role Play)
 - Designation: Event organizer
- Interviewer:
 - Axit Dhola (Student of DAIICT)
 - Sanny Dhameliya (Student of DAIICT)
 - Bhadrayu Bhalodia (Student of DAIICT)
 - Kenil Bhingradiya (Student of DAIICT)
- Date: 05/10/2021
- Time: 17:30
- Duration: 30 minutes
- Place: Google Meet
- Summary of Interview
 - Preliminary meeting to identify problems and requirements regarding decisions of the Manager about Booking Management.
 - The users should be able to select movies/concerts/play at its nearby place so that they can select appropriate theatre and can also select show time, select ticket price, and set and carry out the efficient purchase of a ticket.
 - Make sure that the customer always gets the acknowledgment about his/her ticket for this event successfully booked.
 - Event organizer provides offers and rewards to the customers i.e. send alerts to the customers at an instance of time. So, customers can use the offers when they wish to do so.

Interview 3

- Role Play interview plan
- System: Entertainment booking System
- Interviewee: Vishabh maniya(Role Play)
 - Designation: Customer
- Interviewer:
 - Axit Dhola (Student of DAIICT)

- Sanny Dhameliya (Student of DAIICT)
 - Bhadrayu Bhalodia (Student of DAIICT)
 - Kenil Bhingradiya (Student of DAIICT)
- Date: 06/10/2021
- Time: 18:30
- Duration: 30 minutes
- Place: Google Meet
- Purpose of Interview:
 - Preliminary meeting to identify problems and requirements regarding decisions of the Manager about Event.
- Agenda:
 - Problems with the lack of Management
 - Current Situation about booking System
 - Initial ideas
 - Follow-up actions

Interview Summary 3

- System: Entertainment booking System
- Interviewee: Vishabh Maniya(Role Play)
 - Designation: Customer
- Interviewer:
 - Axit Dhola (Student of DAIICT)
 - Sanny Dhameliya (Student of DAIICT)
 - Bhadrayu Bhalodia (Student of DAIICT)
 - Kenil Bhingradiya (Student of DAIICT)
- Date: 06/10/2021
- Time: 18:30
- Duration: 30 minutes
- Place: Google Meet
- Summary of Interview
 - Additionally, users should be able to buy snacks at the desired time and receive alerts for forthcoming movies/concerts/theatre shows.
 - Also, users watch trailers of currently showing and upcoming movies, Read reviews, synopsis, the cast of currently showing and forthcoming movie/ theatre/ concerts, and View and provide theatre/ venue review, user ratings, etc...
 - Allow users to sort movies/ plays according to their genre and also provide movie/ play/ concert reviews, user ratings. See theatres/ venues on a map and get directions from the current location.
 - Further discussions are needed when more information is available.

2.2.1 Combined Requirements gathered from the Interviews

- As per all interviews, there has to be a good UI for the user interface with the platform.
- There is good privacy practice and robust data management and security.
- There should be multiple platform support so users can have accessibility from all types of devices and also be accessible on low-speed internet.
- There should be enough cloud storage and continuous flow to data read and write in the database.
- There should be real-time updates for the database without fail.
- There should be good communication between the customer and platform administrator for any interface difficulties(bugs).

2.3 Questionnaires:-



Your tickets are just one click away...
Still Waiting for tickets?

Online Events Registration portal

movies, concert and many more...

 201901049@daiict.ac.in (not shared) 

* Required

What is your preferred mode for register events ? *

Online
 Offline

What is your age group ? *

<18
 18-30
 >30

How often do you use online event registrations platforms every month? *

0-5
 6-10
 10-15
 >15

What is your preferred events for online registrations? *

Movies
 Concert
 Live promotion events
 Filmfare
 Live drama events
 New product release events
 Other: _____

Which platforms for event registration *

ZoomInfo
 Elevate
 PayTm
 Amazon Pay
 BookMyShow
 Other: _____

Satisfaction for UI of your preferred platforms, *

1 2 3 4 5
bad ○ ○ ○ ○ ○ excellent

Which mode of payment do you use ? *

UPI
 Debit/Credit Card
 Internet Banking
 PayTm/Amazon Pay Wallet
 Other: _____

Satisfaction level for event registration platforms, *

1 2 3 4 5
bad ○ ○ ○ ○ ○ excellent

Suggestion for events registration platforms

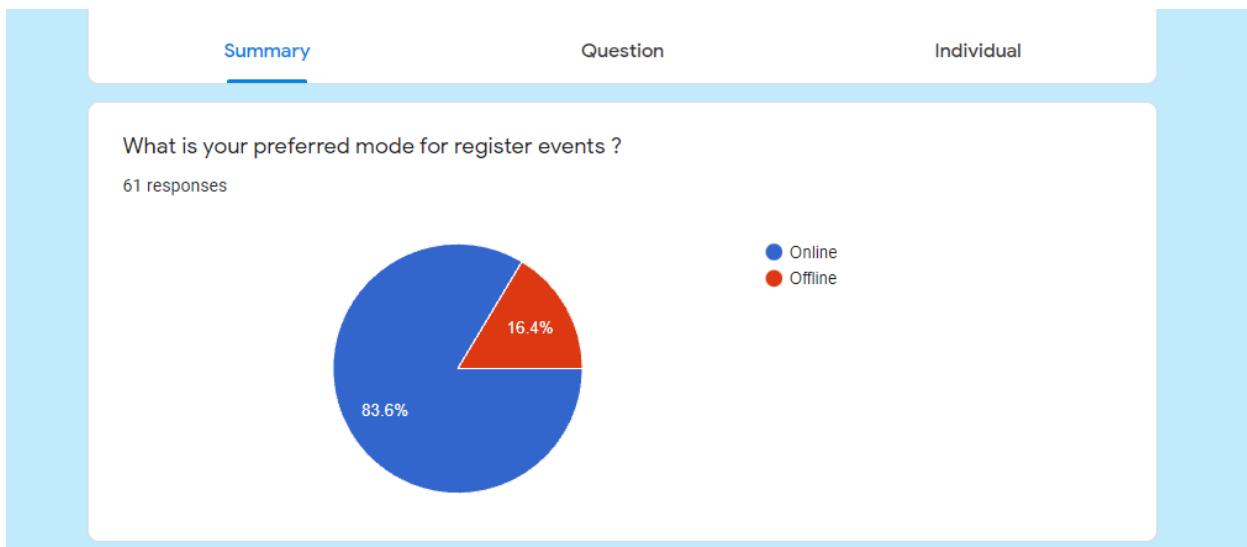
Your answer

Submit Page 1 of 1 **Clear form**

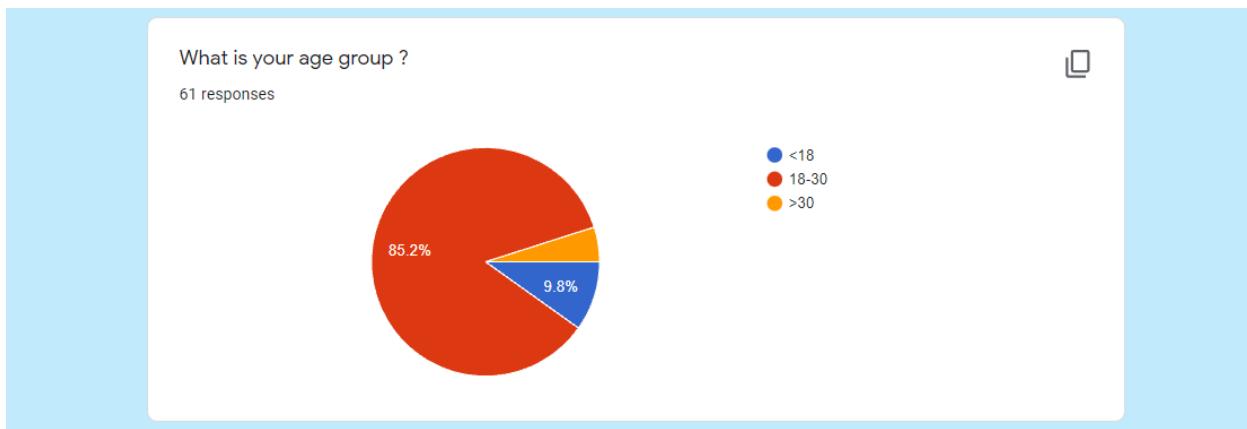
Never submit passwords through Google Forms.

This form was created inside of Dhirubhai Ambani Institute of Information and Communication Technology. [Report Abuse](#)

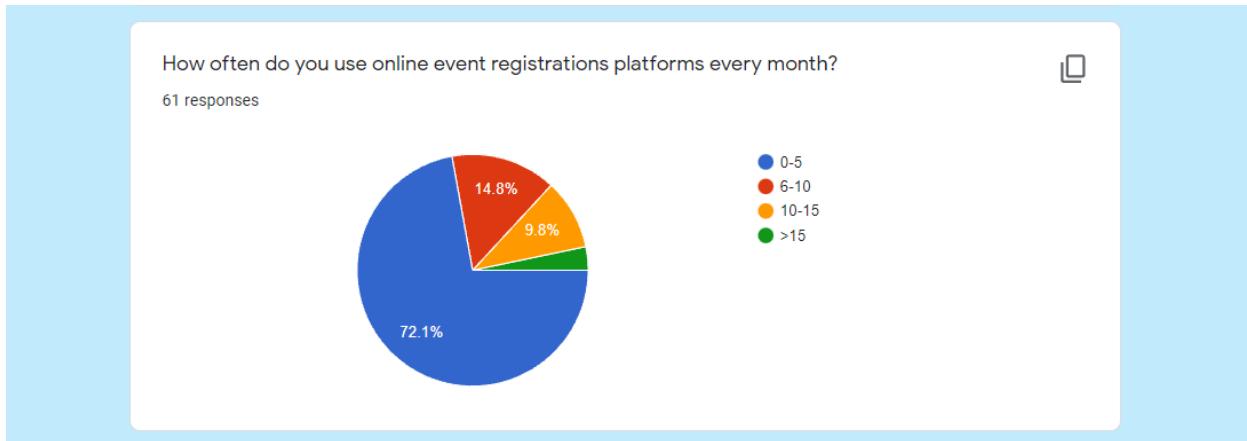
Google Forms



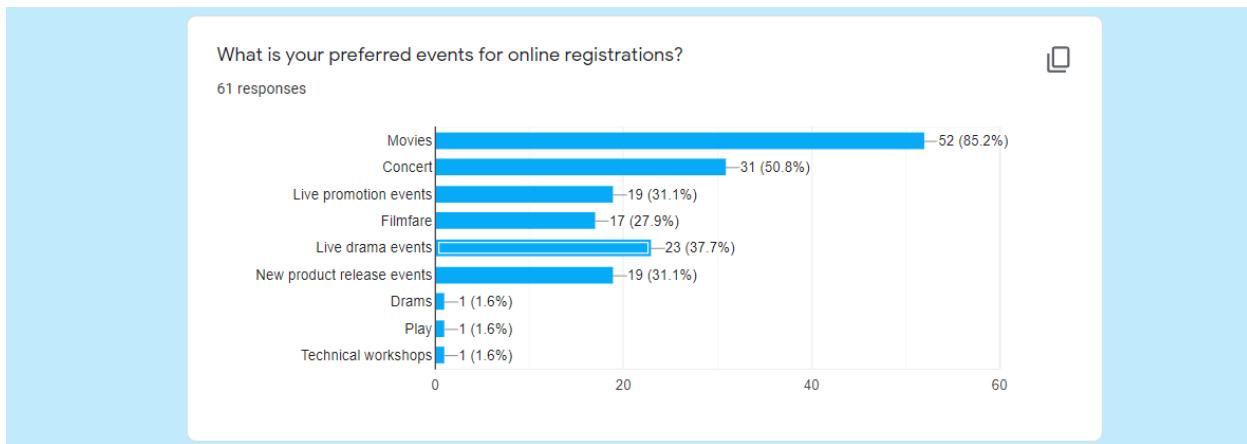
➤ Most customers prefer to book tickets online for different events.



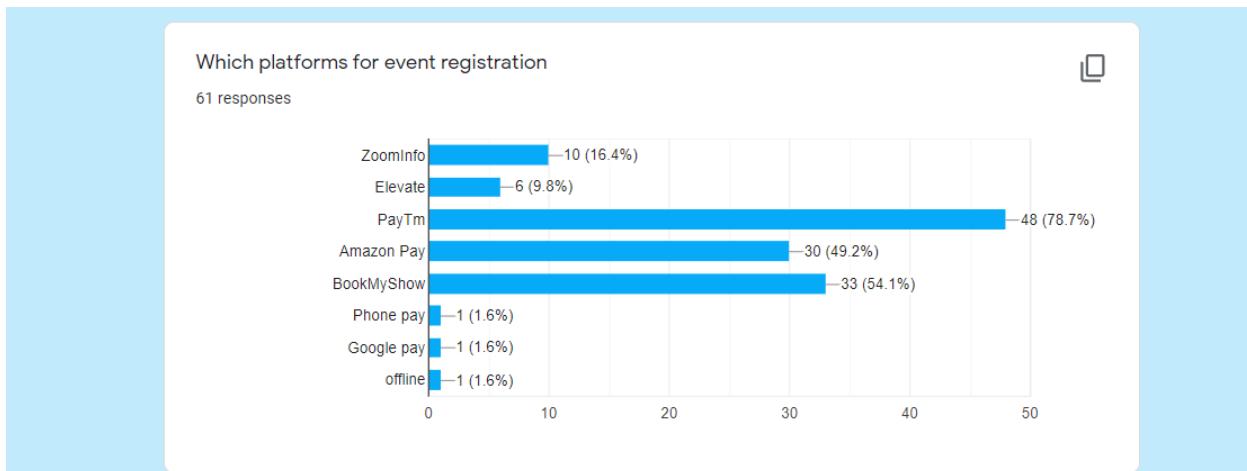
➤ Likely 86% of customers belong to the 18-30 age group.



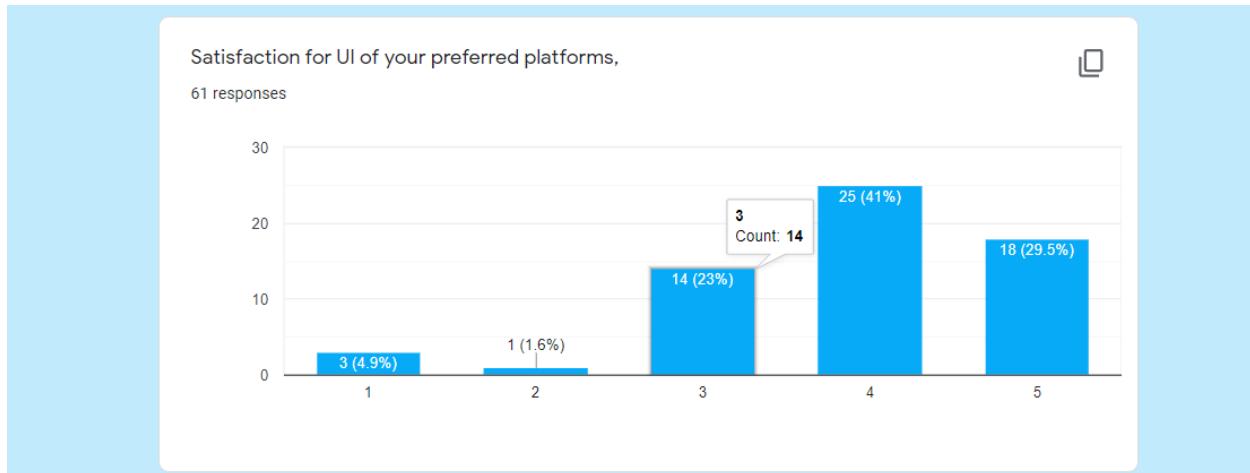
- Most of the customers go to around 0-5 events every month.



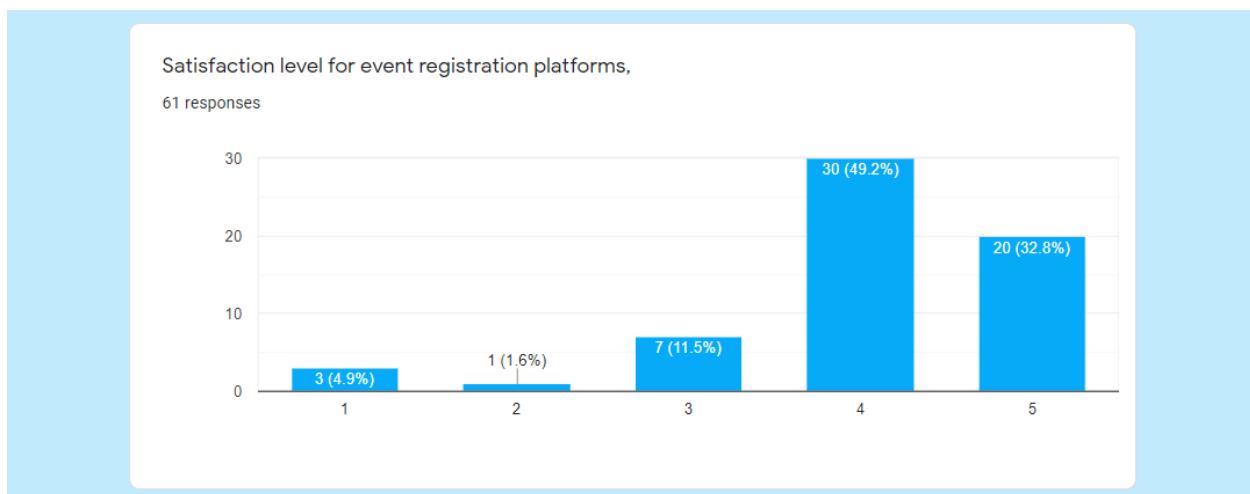
- As per the graph, most customers prefer movies and concert booking in online mode.



- As per the graph, most customers prefer PayTm, BookMyShow, and Amazon pay for booking in online mode.



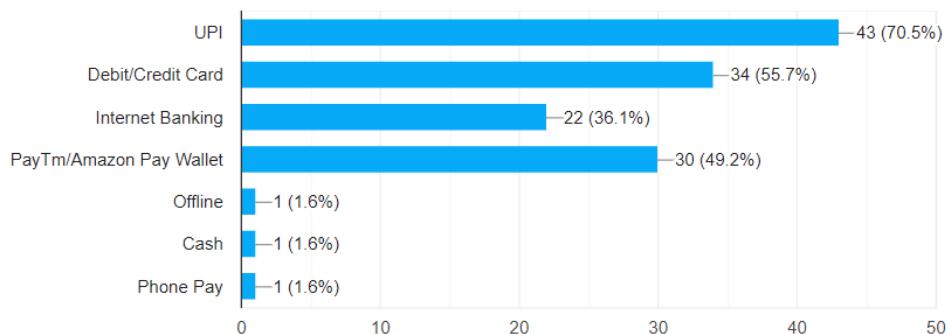
- Platform UI is good as per review for customers.



- Most of the customers are satisfied with online bookings.

Which mode of payment do you use ?

61 responses



- In online mode, most customers preferred UPI, Debit/credit card, and internet banking for payment.

2.4 Observation:

- System: Entertainment booking System
- Observation by:
 - Axit Dhola (Student of DAIICT)
 - Sanny Dhameliya (Student of DAIICT)
 - Bhadrayu Bhalodia (Student of DAIICT)
 - Kenil Bhingradiya (Student of DAIICT)
- Date: 07/10/2021
- Time: 18:30
- Duration: 30 minutes
- Place: Google Meet
- Real-time update in the database of any events and availability.
- It is easy to learn and user-friendly interface.
- The data privacy policy is robust.
- Sometimes in many platforms unable to log in doesn't get OTP or invalid OTP.
- Sometimes platform does not load.
- In most platforms, customer care service is very bad, and wait for a long time to get a reply or chat/call with a representative.
- Many platforms take feedback from users and do nothing with it to solve their or common issues.
- After reviewing many public reviews of some platforms like BookMyShow user says before book ticket there is a 100% refund policy and after the ticket is booked the ticket says no cancellation is available. So, no transparency in their policy.

[C]. Create Fact-Finding Chart

Objective	Technique	Subjects	Time Commitment
Background knowledge of the platform	Background readings	Few Similar projects and requirements of this platform	1 day
Preliminary meeting to identify problems and requirements regarding decisions	Interview	Administrator of platform	30 minutes
Preliminary requirements of organizers	Interview	Event organizer	30 minutes
Preliminary requirements of users	Interview	customer	30 minutes
To get Customer feedback on the system	Questionnaires	A regular customer at a platform	1 hour
To get knowledge of event booking platform	Observation	Customer	30 minutes
To get knowledge User-friendly UI	Observation	Customer	30 m

[D]. List Requirements

➤ NON Functional requirements:-

- **Reliability**
 - The reliability of the overall program depends on the reliability of the individual components. The main pillar of system reliability is the backup of the database which is continually maintained and updated to reflect the most recent changes.
 - Thus the overall stability of the system depends on the stability of the container and its underlying operating system.
- **Availability**
 - The system must always be available, which means that the user can access it using a web browser, limited only by the downtime of the server on which the system is running. In the event of a hardware failure or database corruption, a replacement page will be displayed. Even in the event of a hardware failure or database corruption, database backups should be retrieved from the server and recorded by the administrator. Then the service will be restarted. This means 24X7 availability.
- **Maintainability**

- A commercial database is used for database maintenance and the application server takes care of the site. In the event of an error, the program will be reset. In addition, the software design is carried out with consideration of modularity so that maintainability can be carried out efficiently.
- **Portability**
 - The application is based on HTML and scripting language. Thus, the end-user part is fully portable and any system using any web browser should be able to use the functionality of the system, including any hardware platform that is available or will be available in the future.
 - An end-user is using this system on any OS; either it is Windows or Linux.
 - The system shall run on PC, Laptops, and PDAs, etc.
- **Security**
 - The system uses SSL (secured socket layer) in all transactions that include any confidential customer information.
 - The system must automatically log out all customers after a period of inactivity.
 - The system should not leave any cookies on the customer's computer containing the user's password.
 - The system's back-end servers shall only be accessible to authenticated administrators.
 - Sensitive data will be encrypted before being sent over insecure connections like the internet.
- **Accessibility**
 - The system will be a web-based application that is going to be accessible on the web browser.
- **Back up**
 - We will take a backup in our system database. To enable the administrator and the user to access the data from our system!
- **Performance**
 - The product shall be based on the web and has to be run from a web server.
 - The product shall take initial load time depending on internet connection strength which also depends on the media from which the product is run.
 - The performance shall depend upon hardware components of the client/customer
- **Accessibility**
 - The system shall provide handicap access.
 - The system shall provide multi-language support.
- **Supportability**
 - The source code developed for this system shall be maintained in the configuration management tool.

➤ **Functional Requirements: -**

- **Registration**
 - If a customer wants to book the ticket then he/she must be registered, an unregistered user can't book the ticket.
- **Login**
 - Customer logs in to the system by entering valid user IDs and passwords for booking the ticket.
- **Search Movie**
 - The system shall have a search function. Customer or visitors can search movies based on movie name, date, time, and venue
- **Seat Viewing**
 - The customer shall be shown a 2D image of the seats from which the desired seats are selected.

- **Ticket canceling**
 - The customer shall be given an option to cancel the ticket one hour before the movie with some fine.
- **Payment**
 - For the customer, many types of secure billing will be prepaid by debit or credit card. The security will be provided by a third party like Pay-Pal etc.
- **Logout**
 - After the payment or browse the movie, the customer will log out.
- **Generate ticket**
 - After booking, the system can generate the portable document file (.pdf) and then send one copy to the customer's Email-address and another one as an SMS to the customer's phone.
- **Add movies**
 - The system shall have a feature for the admin to add movies and their details.
- **Remove movies**
 - The system shall have a feature for the admin to remove movies.

[E]. User Classes and Characteristics

- There are mainly four types of users
 - Event organizers
 - Event managers
 - Customers
 - Administrator
- The first one is event organizers who organize the events. The one who will come to put their event on a portal. They have access to all customers' data who register on this event and financial information.
- The second one is an event manager who manages all details about events like all requirements to show up on events, event details, etc. They have access to edit event details and customer names.
- The third one is customers, who want to participate in particular events. They have access to their profiles and all ongoing and upcoming event details.
- And the last one is the administrator who owns this platform. The administrator has access to the entire database.

[F]. Operating Environment

- Hardware:- i3 Processor or Android Version 5.0 or Higher
- Software:- Android OS or Windows 7 and up
- Connectivity Requirements:- Wi-Fi or Mobile Internet connectivity
- External Interface Requirements :
 - Third-party API:
 - Google's geographical locations API
 - Online bank's API
 - Different event platform API

[G]. Product Functions

- Providing real-time information about the below objects:-
 - Number of total upcoming events and their details
 - Total empty seat for every particular event
 - Food management
 - Your transaction history
 - Your account details
 - Completed events and live events detail
 - Your ticket status
 - Ticket details

[H]. Privileges

Data	Privileges
All User Details	Administrator, event organizer
Events details	Customer, Administrator, Manager, Organizer
Food details	Manager, administrator
Booking status(read,update,cancel)	Customer, administrator

[I]. Assumptions

- All required numerical information is in an accurate form.
- All third-party APIs should be working successively.
- Quick payment response from banks.
- Quick update in the database by all third-party providers.
- Each person must have a device with an internet connection.
- Users should have a basic idea about the internet and online payment and also should be familiar with computers.

[J]. Business Constraints

- The platform will be free to use to create new research projects and collaborate. To properly maintain the platform and bear the server costs, users must pay some predefined amount when their research paper gets published. Also, the project comes under an open-source license. So developers are welcomed to be a part of the development team.
- Creation of digital infrastructure, good internet connection, and fast, accessible database server that can handle large amounts of traffic at a time for this platform.

Section2: Noun Analysis.

1. Noun Analysis

Noun	Verb
Customer	Register
Event Organizer	Add/Remove events
Manager	Add/remove customer
Administrator	Manage platform
Name	search
Address	design
Gender	access
Phone No.	handle
Email ID	time
Age	organize
ID proof	Generate
Events	book
Food	Allow/prevent
Events Name	process
Login	contact
Password	upload
Tickets	update

Price	order
Payment	seats
Billing receipt	reserve
feedback	receive
Event Date	host
Event Time	edit
Refund	pay
Transaction ID	card
Customer ID	credit
Event ID	view
Event type ID	help
Ticket ID	negative
Manager ID	site
Vaccination Status	manage
Seat availability	add
Payment Method	modify
Destination	edit
Venue	change
Description	come

Motto	source
Different types	work
process	detail
owner	allow
Large data	selected
Information	points
Theaters	report
platform	record
Integrity	confirm
Financial data	personalized
Update	related
Credit card details	assign
Feedback reports	authorize
Different expects	
Event records	
View booking	
Cancellation report	
Payment report	
Account module	

Login ID	
Display	
Update profile	
Efficient	
Browser	
Software	
Procedure	
Internet	
resources	

2. Entity- Attributes:-

Candidate entity set	Candidate attribute Set	Candidate relationship set
customer	1. c_id 2. c_name 3. c_address 4. c_email 5. c_phoneno 6. c_age 7. password	login,add user
event	1. event_id 2. event_type_id 3. event_name 4. event_date 5. event_address 6. event_manager 7. event_start_time 8. event_end_time 9. event_size	Insert events and their details

event organizer	1. eo_id 2. event_id 3. eo_name 4. password	Details of event organizer
manager	1. m_id 2. m_name 3. m_phoneNo	Event managers
event_type	1. event_type_id 2. event_description	Preffed event types
ticket	1. t_id 2. c_id 3. t_price 4. seat_no 5. p_id	Ticket details
payment	1. p_id 2. c_id 3. P_type 4. f_id 5. offer_id 6. total_amount	Payment details and total payment details
food	1. f_id 2. f_name 3. f_price 4. f_quantity	Food details
offer	1. o_id 2. o_name 3. discount	Offer details
refund/cancellation	1. r_id 2. p_id 3. r_reason 4. r_amount 5. r_type	Return or cancellation detail

3. Rejected Nouns

Noun	Reject Reason
Large data	irrelevant
process	general
internet	Irrelevant
Data	Irrelevant
resources	redundant
ID proof	vague
deploy	duplicate
Description	Attribute
database	general
information	vague

4. Rejected Verbs

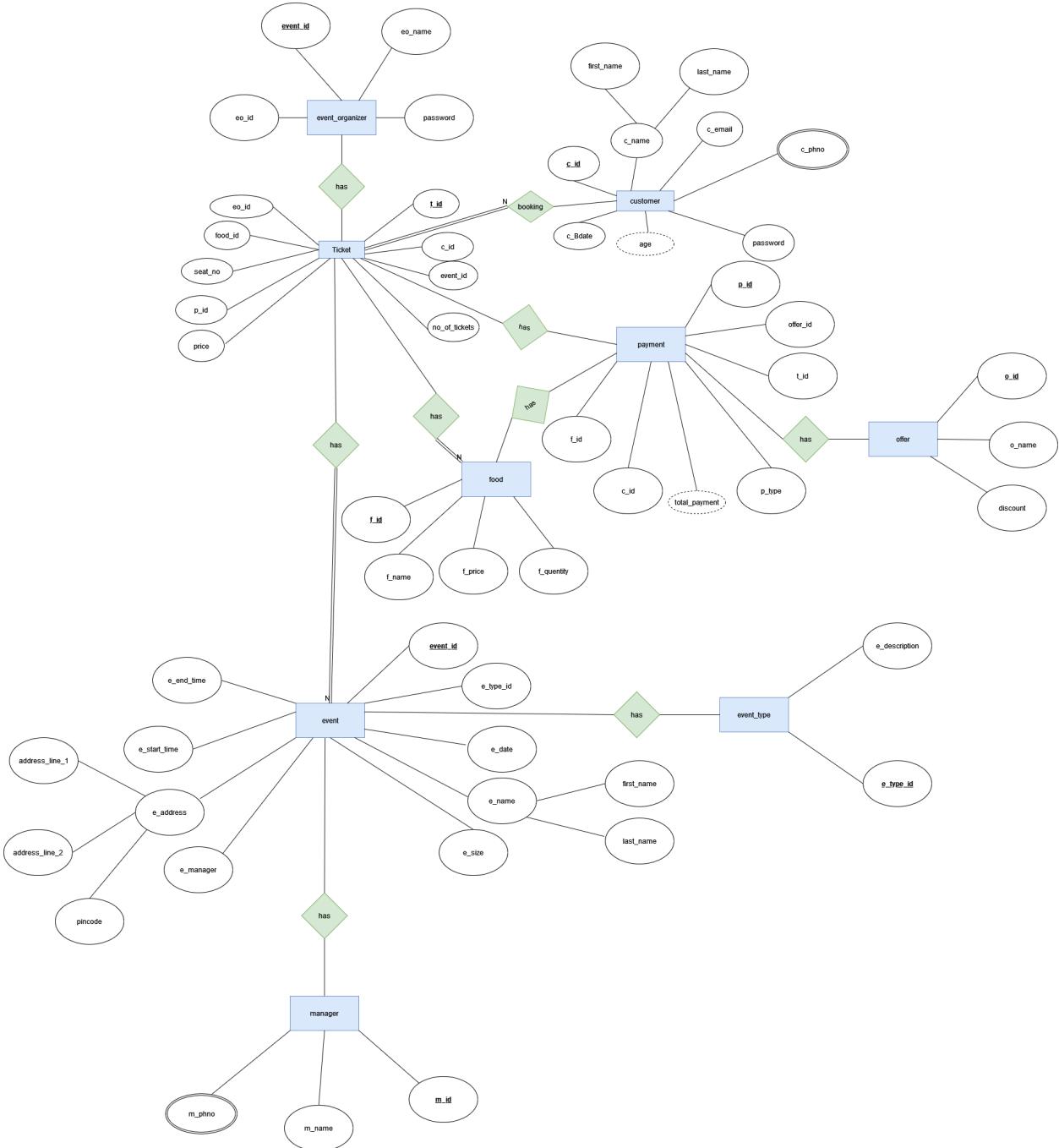
Verb	Reject Reason
edit	duplicate
modify	duplicate
access	duplicate
generate	general

assign	duplicate
authorize	general
view	general
help	general

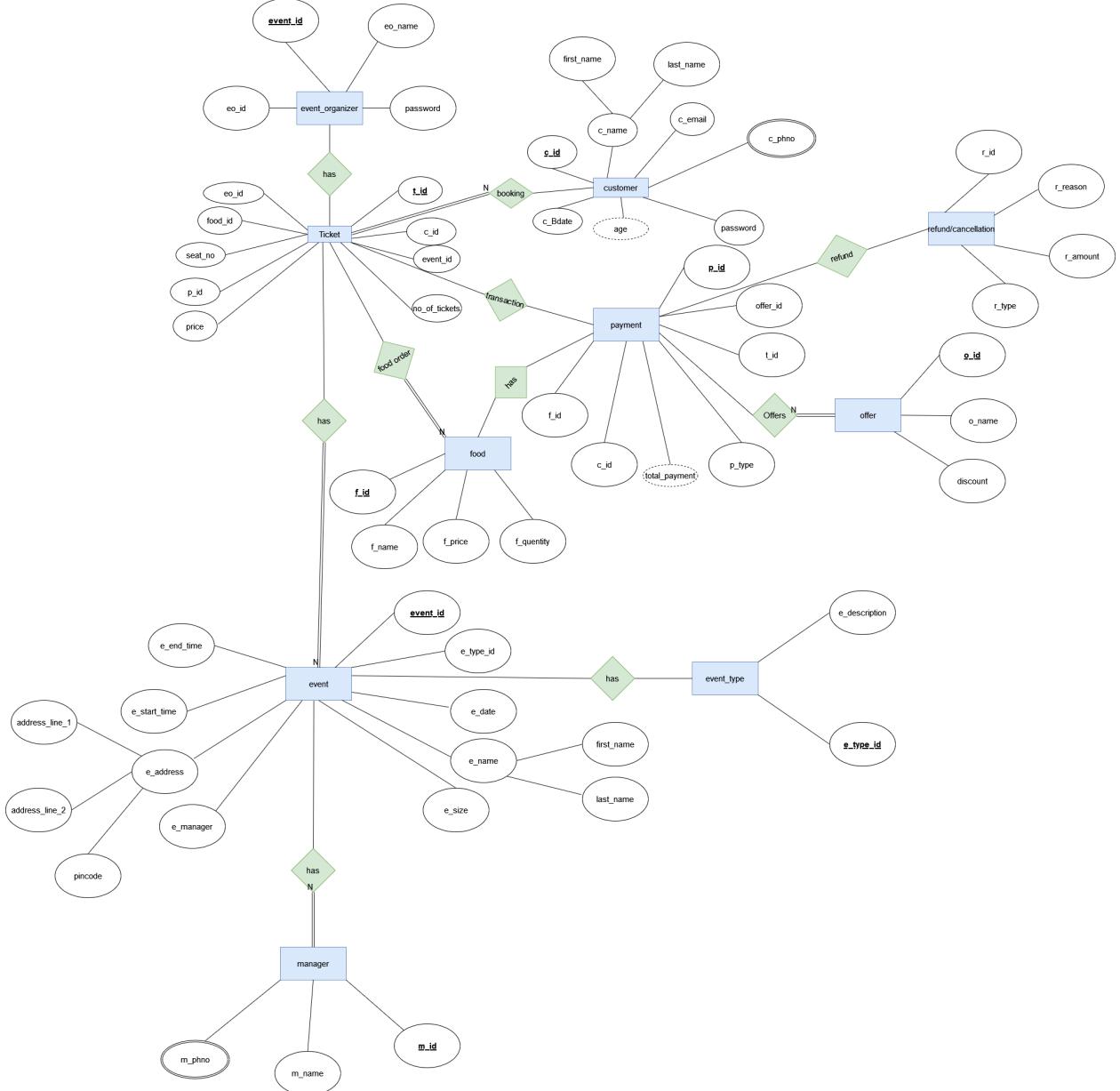
Section3: ER-Diagrams all versions

E-R Diagram:-

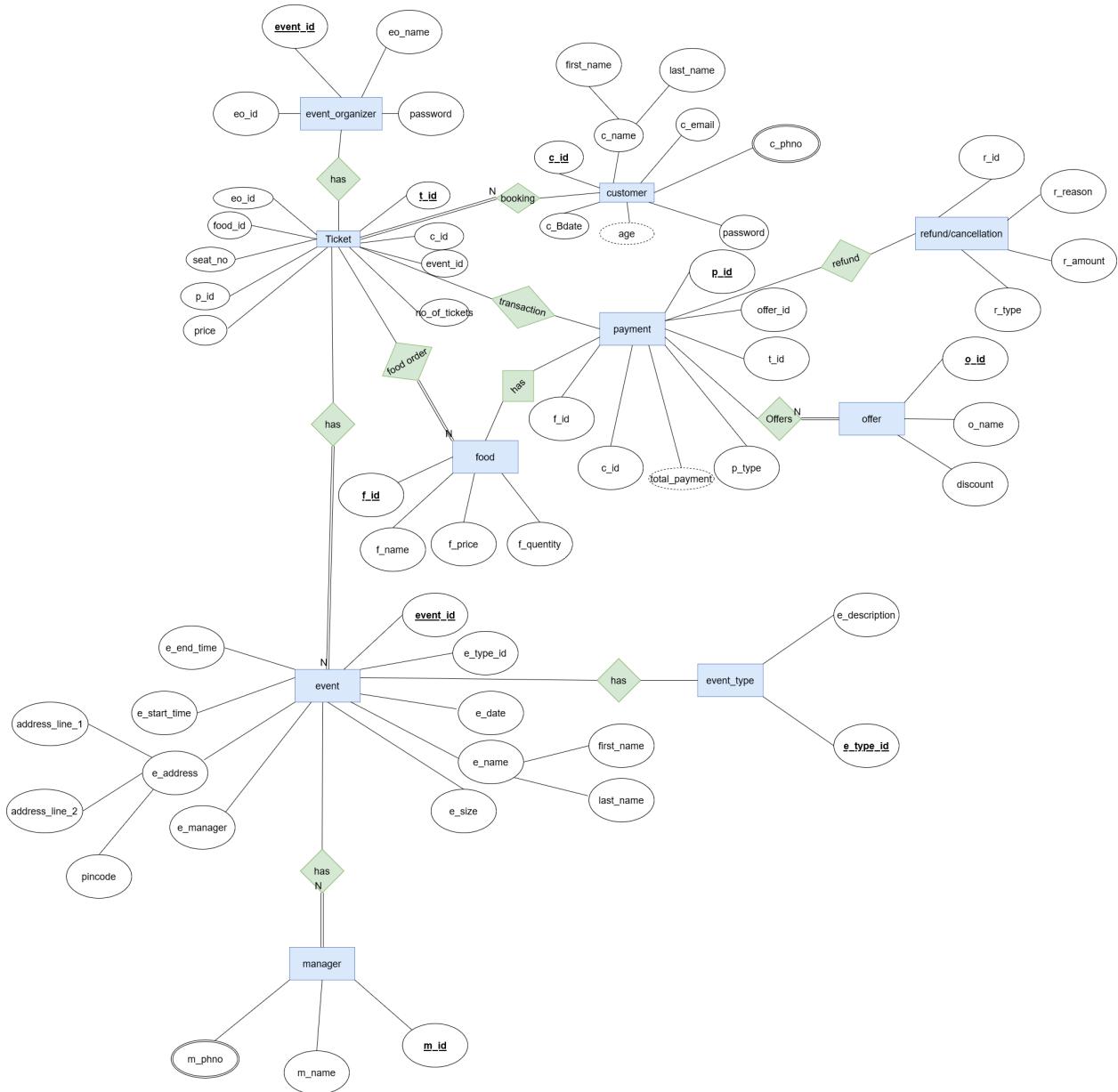
1. ER diagram (version 1)



2. ER diagram version 2

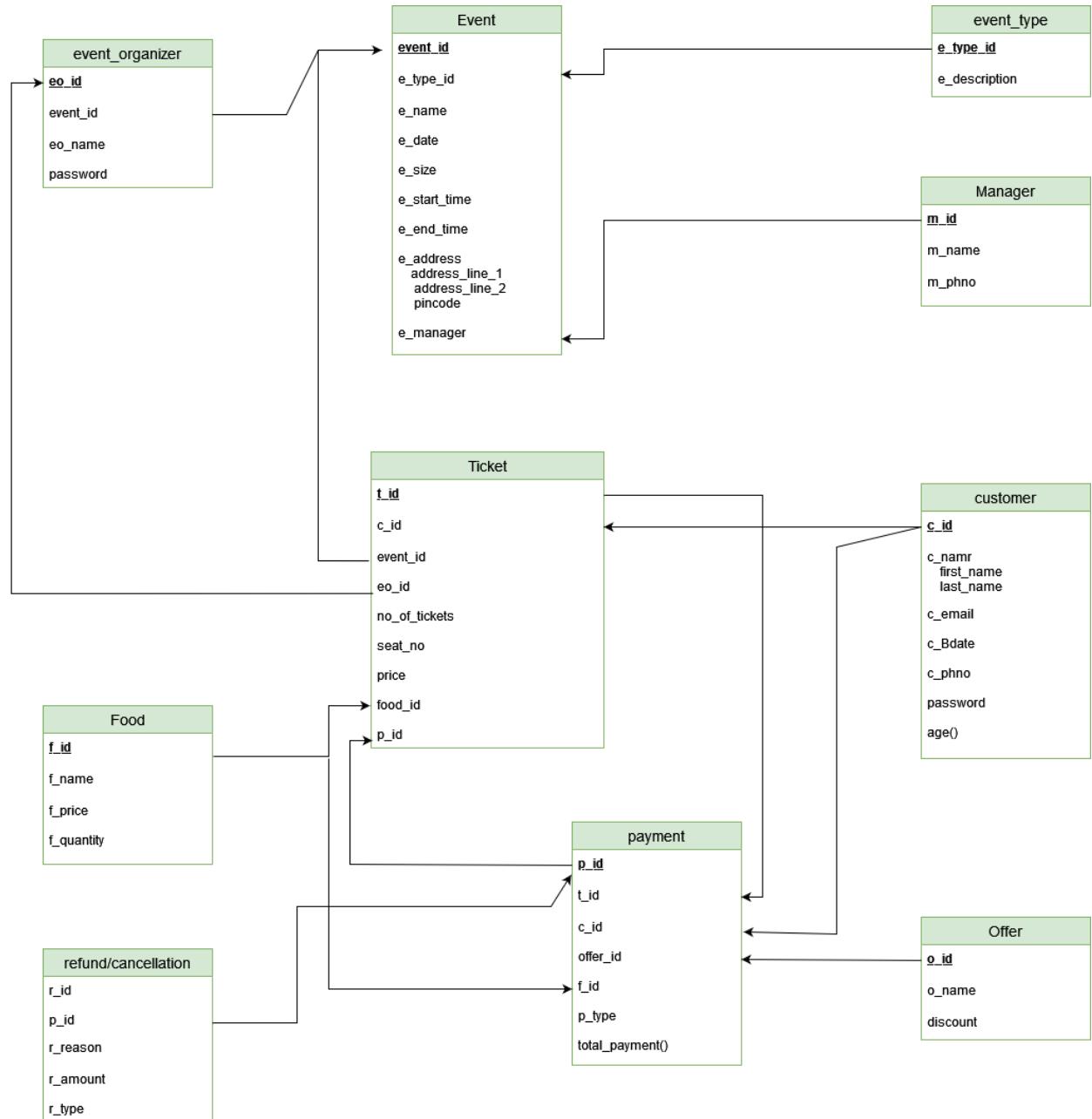


3. ER diagram final Version



Section4: Conversion of Final ER-Diagram to Relational Model.

4. RE relationship Diagram



Mapping E-R Model to Relational Model

- event_organizer(**eo_id**, event_id, eo_name, password)
 - Primary key:- eo_id
 - Foreign key:- event_id
- event(**event_id**, e_type_id, e_name, e_date, e_size, e_start_time, e_end_time, e_address, address_line1, address_line2, pincode, e_manager)
 - Primary key:- event_id
 - Foreign key:- event_type_id, e_manager
- event_type(**e_type_id**, e_desciption)
 - Primary key:- e_type_id
- manager(**m_id**, m_name, m_phno)
 - Primary key:- m_id
- customer (**c_id**, first_name, last_name, c_email, c_Bdate, c_phno, password, age)
 - Primary key:- c_id
- ticket (**t_id**, c_id, event_id, eo_id, no_of_tickets, seat_no, price, food_id, p_id)
 - Primary key:- t_id
 - Foreign key:- c_id, event_id
- payment (**p_id**, t_id, c_id, offer_id, f_id, p_type, total_payment)
 - Primary key:- p_id
 - Foreign key:- c_id, t_id
- food (**f_id**, f_name, f_price, f_quantity)
 - Primary key:- f_id
- offer (**o_id**, o_name, discount)
 - Primary key:- o_id
- refund/cancellation (**r_id**, p_id, r_reason, r_amount, r_type)
 - Primary key:- r_id
- booking (**t_id**, c_id)
 - Primary key:- t_id
- food_order(**f_id**, t_id)
 - Primary key:- f_id
- transaction(**t_id**, p_id)
 - Primary key:- t_id

Section 5: Normalization and Schema Refinement

→ Normalization & Schema Refinement

1) List all the Relations & Schemas with all details (Original Design of Database).

- event_organizer(eo_id, event_id, eo_name, password)
 - Primary key:- eo_id
 - Foreign key:- event_id
- event(event_id, e_type_id, e_name, e_date, e_size, e_start_time, e_end_time, e_address, address_line1, address_line2, pincode, e_manager)
 - Primary key:- event_id
 - Foreign key:- event_type_id, e_manager
- event_type(e_type_id, e_desciption)
 - Primary key:- e_type_id
- manager(m_id, m_name, m_phno)
 - Primary key:- m_id
- customer (c_id, first_name, last_name, c_email, c_Bdate, c_phno, age, password)
 - Primary key:- c_id
- ticket (t_id, c_id, event_id, eo_id, no_of_tickets, seat_no, price, food_id, p_id)
 - Primary key:- t_id
 - Foreign key:- c_id, event_id
- payment (p_id, t_id, c_id, p_type, f_id, offer_id, total_amount)
 - Primary key:- p_id
 - Foreign key:- c_id, t_id
- food (f_id, f_name, f_price, f_quantity)
 - Primary key:- f_id
- offer (o_id, o_name, discount)
 - Primary key:- o_id
- refund/cancellation (r_id, p_id, r_reason, r_amount, r_type)
 - Primary key:- r_id

- booking (t_id, c_id)
 - Primary key:- t_id
- food_order(f_id , t_id)
 - Primary key:- f_id
- transaction(t_id , p_id)
 - Primary key:- t_id

2) Identify and list all types of dependencies (PK, FK, Functional Dependencies) for each relation.

- event_organizer(eo_id -> eo_name, eo_id -> password)
- event(event_id -> e_name, event_id -> e_date, event_id -> e_size, event_id -> e_start_time, event_id -> e_end_time, event_id -> address_line1, event_id -> address_line2, event_id -> pincode)
- Event_type -> no dependencies
- manager(m_id ->m_name, m_id -> m_phno)
- customer (c_id -> c_first_name, c_id -> c_last_name, c_id -> c_address, c_id -> c_email, c_id ->c_phoneno,c_id -> c_Bdate, c_Bdate -> c_age, c_id -> password)
- ticket (t_id -> no_of_tickets, t_id -> price, seat_no, t_id -> p_id)
- payment (p_id -> p_type, p_id -> f_id, t_id -> total_amount)
- food (f_id -> f_name, f_id -> f_price)
- Offer -> no dependencies
- refund/cancellation (r_id -> r_reason, p_id -> r_amount, p_id -> r_type)

3) List of redundancies existing for every schema which is part of the database

- event_organizer(eo_id, event_id, eo_name, password)
 - Primary key:- eo_id
 - Foreign key:- event_id
 - Redundancy:-
 - Update anomaly:- It will set all foreign-keyed relation attributes to the updated value.
 - Insert anomaly:- You can not insert a new event without assigning an event.
 - Delete anomaly:- If deleted, you lost all data regarding that event organizer.
 - 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
 - 2NF:- There are no partial dependencies to this relation is already in 2NF.
 - 3NF:- There are no transitive dependencies, so this relation is already in 3NF.
- event(event_id, e_type_id, e_name, e_date, e_size, e_start_time, e_end_time, e_address, address_line1, address_line2, pincode, e_manager)
 - Primary key:- event_id
 - Foreign key:- event_type_id, e_manager
 - Redundancy:- There can be multiple event names with the same addresses or at the same time.
 - Update anomaly:- It will set all foreign-keyed relation attributes to the updated value.
 - Insert anomaly:- You can not insert a new event without assigning event type and event manager for that event.
 - Delete anomaly:- if deleted, you lost all data regarding that event.
 - 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
 - 2NF:- There are no partial dependencies to this relation is already in 2NF.
 - 3NF:- Here in this entity relationship, start time and end time is dependent on event name and event name is dependent on event_id, so because of these, there is transitive dependency.
 - event(event_id, e_type_id, e_name, e_size, e_address, address_line1, address_line2, pincode, e_manager)
 - time(e_id, start_time, end_time)
- event_type(e_type_id, e_desciption)
 - Primary key:- e_type_id
 - Update anomaly:- It will not affect anything
 - Insert anomaly:- It will not affect anything
 - Delete anomaly:- If deleted, you lost all data regarding that event type.
 - 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
 - 2NF:- There are no partial dependencies to this relation is already in 2NF.
 - 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

➤ manager(m_id, m_name, m_phno)

- Primary key:- m_id
- Update anomaly:- It will not affect anything
- Insert anomaly:-It will not affect anything
- Delete anomaly:- If delete you lose all data regarding that manager
- 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
- 2NF:- There are no partial dependencies to this relation is already in 2NF.
- 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

➤ customer (c_id, c_name, c_address, c_email, c_phoneno, c_Bdate, c_age, password)

- Primary key:- c_id
- Update anomaly:- if you update the birth date, then age will automatically change.
- Insert anomaly:-It will not affect anything.
- Delete anomaly:- If delete you lose all data regarding that customer
- 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
- 2NF:- There are no partial dependencies to this relation is already in 2NF.
- 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

➤ ticket (t_id, c_id, event_id, eo_id, no_of_tickets, price, food_id, t_price, seat_no, p_id)

- Primary key:- t_id
- Foreign key:- c_id, event_id
- Update anomaly:- It will not affect anything.
- Insert anomaly:- You can not insert a new event without assigning customers and an event.
- Delete anomaly:- If deleted, you lost all data regarding that ticket.
- 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
- 2NF:- There are no partial dependencies to this relation is already in 2NF.
- 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

➤ payment (p_id, c_id, p_type, f_id, offer_id, total_amount)

- Primary key:- p_id
- Foreign key:- c_id, t_id
- Update anomaly:- you cannot update details of payment directly.
- Insert anomaly:-You can not insert a new event without having a ticket.
- Delete anomaly:- If deleted, you lost all data regarding that payment.
- 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
- 2NF:- There are no partial dependencies to this relation is already in 2NF.
- 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

➤ food (f_id, f_name, f_price, f_quantity)

- Primary key:- f_id
- Update anomaly:- It will not affect anything
- Insert anomaly:- It will not affect anything.
- Delete anomaly:- It will not affect anything

- 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
- 2NF:- There are no partial dependencies to this relation is already in 2NF.
- 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

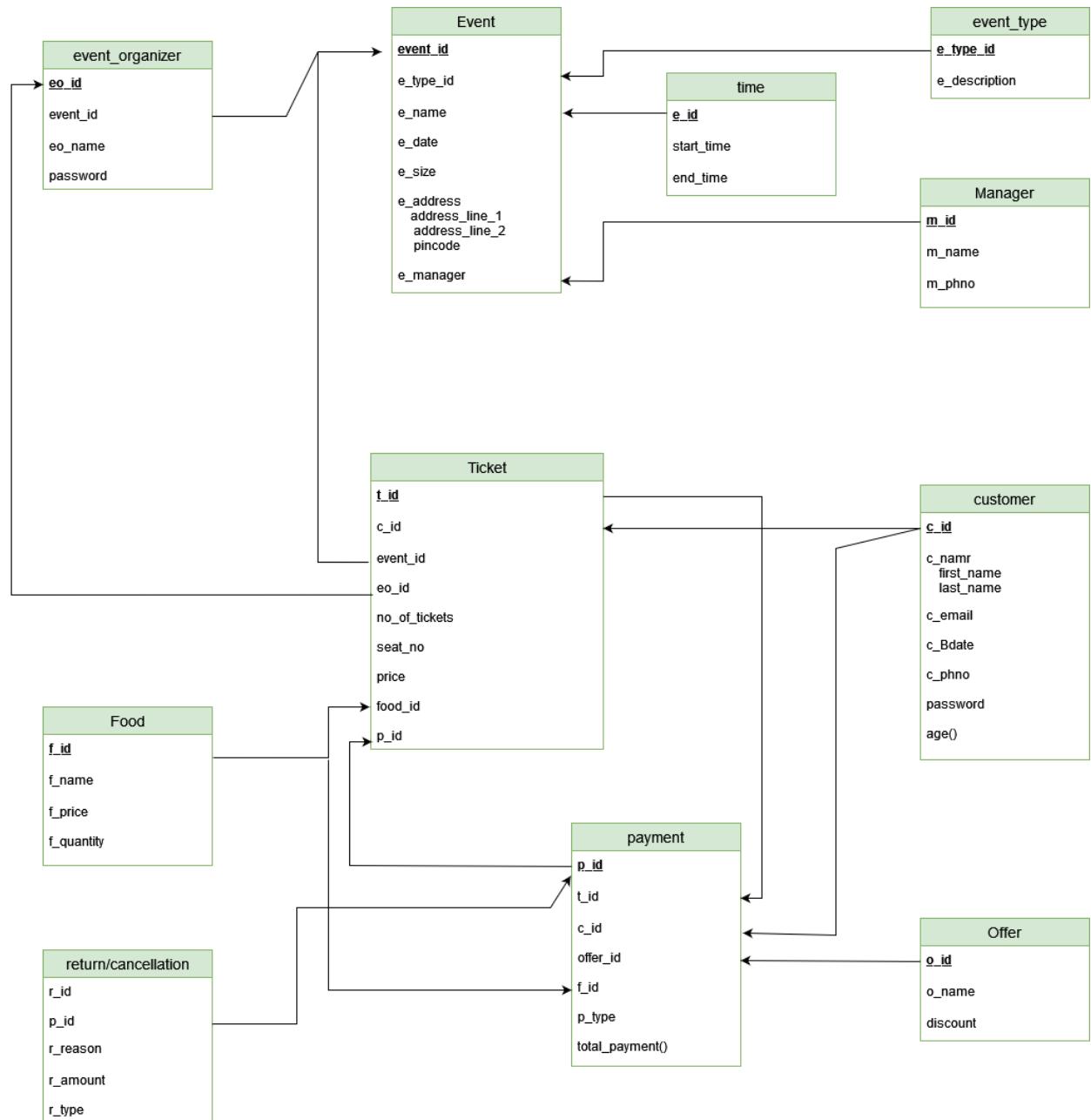
➤ offer (**o_id**, o_name, discount)

- Primary key:- o_id
- Update anomaly:- It will not affect anything.
- Insert anomaly:- It will not affect anything.
- Delete anomaly:- It will not affect anything.
- 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
- 2NF:- There are no partial dependencies to this relation is already in 2NF.
- 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

➤ refund/cancellation (**r_id**, p_id, r_reason, r_amount, r_type)

- Primary key:- r_id
- Update anomaly:- you can not update anything here.
- Insert anomaly:- You can not insert a new event without having a payment id.
- Delete anomaly:-
- 1NF:- Our relation is already in 1NF, and there are no multiple attributes.
- 2NF:- There are no partial dependencies to this relation is already in 2NF.
- 3NF:- There are no transitive dependencies, so this relation is already in 3NF.

4) Final ER Diagram:-



5) List all final Relations & Schemas with all details (final Design of Database)

➤ **event_organizer**(**eo_id**, **event_id**, **eo_name**, **password**)

- Primary key:- **eo_id**

- Foreign key:- event_id
- > event(event_id, e_type_id, e_name, e_date, e_size, e_address, address_line1, address_line2, pincode, e_manager)
- Primary key:- event_id
 - Foreign key:- event_type_id, e_manager, e_name
- > time(e_name, start_time, end_time)
- Primary key:- e_name
- > event_type(e_type_id, e_desciption)
- Primary key:- e_type_id
- > manager(m_id, m_name, m_phno)
- Primary key:- m_id
- > customer (c_id, c_name, c_address, c_email, c_phoneno, c_age, password)
- Primary key:- c_id
- > ticket (t_id, c_id, t_price, seat_no, p_id)
- Primary key:- t_id
 - Foreign key:- c_id, event_id
- > payment (p_id, c_id, p_type, f_id, offer_id, total_amount)
- Primary key:- p_id
 - Foreign key:- c_id, t_id
- > food (f_id, f_name, f_price, f_quantity)
- Primary key:- f_id
- > offer (o_id, o_name, discount)
- Primary key:- o_id
- > refund/cancellation (r_id, p_id, r_reason, r_amount, r_type)
- Primary key:- r_id
- > booking (t_id, c_id)
- Primary key:- t_id

- food_order(**f_id**, t_id)
 - Primary key:- f_id
- transaction(**t_id**, p_id)
 - Primary key:- t_id

Section 6: SQL Final DDL Scripts, Insert statements, 40 SQL Queries with Snapshots of output of each query

6) DDL Scripts:-

1. Manager

```
CREATE TABLE "lab_project".manager
```

```
(
```

```
    m_id bigint NOT NULL,  
    m_name character varying(20) NOT NULL,  
    m_phno bigint NOT NULL,  
    PRIMARY KEY (m_id)
```

```
);
```

```
COPY lab_project.manager (m_id,m_name,m_phno) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\manager.csv' CSV HEADER;
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Left Panel (Browser):** Shows the database schema structure under "lab_project".
- Top Bar:** PgAdmin 4, File, Object, Tools, Help.
- Toolbar:** Includes icons for file operations like Open, Save, Print, and Database navigation.
- Query Editor:** Contains the SQL code for creating the table and inserting data from a CSV file.
- Data Output:** Displays the resulting table structure and data.
- Bottom Status Bar:** Shows system icons, language (ENG IN), date (13-11-2021), and time (11:13 PM).

```
CREATE TABLE "lab_project".manager  
(  
    m_id bigint NOT NULL,  
    m_name character varying(20) NOT NULL,  
    m_phno bigint NOT NULL,  
    PRIMARY KEY (m_id)  
);  
  
COPY lab_project.manager (m_id,m_name,m_phno) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\manager.csv'  
CSV HEADER;  
  
SELECT * FROM lab_project.manager;
```

m_id	m_name	m_phno
1	Marylee	7116351328
2	Eden	2045564822
3	Gilberto	6568134627
4	Monti	7443130280
5	Malachi	2173486886
6	Janeen	1225135189
7	Orland	6805081585
8	Northrup	6741380425

Successfully run. Total query runtime: 113 msec. 100 rows affected.

2. Customer

```

CREATE TABLE "lab_project".customer
(
    c_id bigint NOT NULL,
    first_name character varying(20) NOT NULL,
    last_name character varying(20) NOT NULL,
    c_email character varying(40) NOT NULL,
    "c_Bdate" date NOT NULL,
    c_phno bigint NOT NULL,
    password character varying(64) NOT NULL,
    age integer NOT NULL,
    PRIMARY KEY (c_id)
);
COPY lab_project.customer (c_id, first_name, last_name, c_email, "c_Bdate", c_phno, password, age) FROM
'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\customer.csv' CSV HEADER;

```

The screenshot shows the pgAdmin 4 interface with the following details:

- Schemas:** lab_project (selected)
- Tables:** customer (selected)
- Query Editor:**

```

110    c_id bigint NOT NULL,
111    first_name character varying(20) NOT NULL,
112    last_name character varying(20) NOT NULL,
113    c_email character varying(40) NOT NULL,
114    "c_Bdate" date NOT NULL,
115    c_phno bigint NOT NULL,
116    password character varying(64) NOT NULL,
117    age integer NOT NULL,
118    PRIMARY KEY (c_id)
119 );
120
121
122 COPY lab_project.customer (c_id, first_name, last_name, c_email, "c_Bdate", c_phno, password, age) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\customer.csv' CSV HEADER;
123
124
125 SELECT * FROM lab_project.customer;
126

```
- Data Output:** A table showing 100 rows of customer data.

c_id	first_name	last_name	c_email	c_Bdate	c_phno	password	age
1	Avictor	Simister	asimister0@ft.com	2020-12-19	6125020777	zJPevAOg9Gb	48
2	Hall	Pauer	hpauer1@nhs.uk	2021-01-18	4791786935	36C0dQZLB9	47
3	Grete	Warrington	gwarrington2@trellian.com	2020-12-02	4884626727	U2M1Hv9x7V5	17
4	Krystal	Rawet	krawet3@mysql.com	2021-05-07	9016615888	vElzzB80	29
5	Fannie	Winfred	fwinfred4@tumblr.com	2021-07-28	6075169081	Df7Nk4v	49
6	Any	Lindholm	alindholm5@163.com	2021-10-13	6884313863	h0aFBrwNGSrv	39
7	Averyl	Buddington	abuddington6@xrea.com	2021-05-10	7777777777	zJPeV0g9Gb	50
8	Claire	Gaine	cgaine7@forbes.com	2021-10-10	8888888888	zJPeV0g9Gb	50

Message bar: Successfully run. Total query runtime: 162 msec. 100 rows affected.

3. Offer

```
CREATE TABLE "lab_project".offer
```

```
(
```

```
    o_id bigint NOT NULL,  
    o_name character varying(40) NOT NULL,  
    discount integer NOT NULL,  
    PRIMARY KEY (o_id)
```

```
);
```

```
COPY lab_project.offer (o_id, o_name, discount) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\offer.csv' CSV HEADER;
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the schema structure of the database, including Schemas (3), Tables (8), and Trigger Functions (2).
- Query Editor:** Displays the SQL code used to create the table and copy data from a CSV file.
- Data Output:** Shows the resulting data in a table format.
- Status Bar:** Indicates the query was successfully run with a runtime of 61 msec and 100 rows affected.

```
139 SELECT * FROM lab_project.food;  
140  
141 CREATE TABLE "lab_project".offer  
(  
142     o_id bigint NOT NULL,  
143     o_name character varying(40) NOT NULL,  
144     discount integer NOT NULL,  
145     PRIMARY KEY (o_id)  
146 );  
147  
148  
149  
150  
151 COPY lab_project.offer (o_id, o_name, discount) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\offer.csv'  
152 CSV HEADER;  
153  
154  
155  
156  
Explain Messages Notifications Data Output  
o_id [PK] bigint o_name character varying(40) discount integer  
1 1 Iguanatall 46  
2 2 Dwarf Eelgrass 7  
3 3 Small's Rainily 44  
4 4 Denseflower Mullein 16  
5 5 Beachhead Iris 5  
6 6 Bulletwood 7  
7 7 Amazon Canoparmelia Lichen 13  
8 8 California Sunflower 43
```

Successfully run. Total query runtime: 61 msec. 100 rows affected.

4. Event_type

```
CREATE TABLE "lab_project".event_type
(
    e_type_id bigint NOT NULL,
    e_description character varying(1000),
    PRIMARY KEY (e_type_id)
);
```

```
COPY lab_project.event_type (e_type_id,e_description) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\event_type.csv' CSV HEADER;
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Schemas:** lab_project (selected)
- Tables:** event_type (selected)
- Query Editor:** Contains the SQL code for creating the table and importing data from a CSV file.
- Data Output:** Shows the imported data rows (7 rows) and a success message.
- System Bar:** Includes icons for file operations, search, and navigation, along with system status indicators like battery level, signal strength, and date/time (11:28 PM, 13-11-2021).

```
226 CSV HEADER;
227
228 CREATE TABLE "lab_project".event_type
229 (
230     e_type_id bigint NOT NULL,
231     e_description character varying(1000),
232     PRIMARY KEY (e_type_id)
233 );
234
235 COPY lab_project.event_type (e_type_id,e_description) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\event_type.csv'
236 CSV HEADER;
237
238 SELECT * FROM lab_project.event_type;
239
240
241
242
243
```

e_type_id	e_description
1	Fusce consequat. Nulla nisl. Nunc nisl.
2	Maecenas tristique, est et tempus semper, est quam pharetra magna, ac consequat metus sapien ut nunc. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Maur
3	Morbi porttitor lorem id ligula. Suspendisse ornare consequat lectus. In est risus, auctor sed, tristique in, tempus sit amet, sem.
4	Maecenas leo odio, condimentum id, luctus nec, molestie sed, justo. Pellentesque viverra pede ac diam. Cras pellentesque volutpat dul.
5	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin risus. Praesent lectus.
6	Aenean lectus. Pellentesque eget nunc. Donec quis orci eget orci vehicula condimentum.
7	Praesent id massa id nisl venenatis lacinia. Aenean sit amet justo. Morbi ut odio.

Successfully run. Total query runtime: 59 msec. 100 rows affected.

5. Payment

```
CREATE TABLE "lab_project".payment
(
    p_id bigint NOT NULL,
    t_id bigint NOT NULL,
    c_id bigint NOT NULL,
    offer_id bigint,
    f_id bigint,
    p_type character varying(20) NOT NULL,
    total_payment integer NOT NULL,
    PRIMARY KEY (p_id),
    CONSTRAINT C_ID_FK FOREIGN KEY (c_id)
        REFERENCES Lab_Project.customer (c_id) ON UPDATE CASCADE,
    CONSTRAINT T_ID_FK FOREIGN KEY (t_id)
        REFERENCES Lab_Project.ticket (t_id) ON UPDATE CASCADE
);  
COPY lab_project.payment(p_id,t_id,c_id,offer_id,f_id,p_type,total_payment) FROM 'E:\Lecture  
e-books\Samester - 5\IT214\Lab\Lab 8\payment.csv' CSV HEADER;
```

```

CREATE TABLE "lab_project".payment (
    p_id bigint PRIMARY KEY,
    t_id bigint,
    c_id bigint,
    offer_id bigint,
    f_id bigint,
    p_type character varying(20),
    total_payment integer NOT NULL
);

ALTER TABLE "lab_project".payment
ADD CONSTRAINT C_ID_FK FOREIGN KEY (c_id)
REFERENCES Lab_Project.customer (c_id) ON UPDATE CASCADE;

ALTER TABLE "lab_project".payment
ADD CONSTRAINT T_ID_FK FOREIGN KEY (t_id)
REFERENCES Lab_Project.ticket (t_id) ON UPDATE CASCADE;

COPY lab_project.payment(p_id,t_id,c_id,offer_id,f_id,p_type,total_payment)
FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\payment.csv' HEADER;

SELECT * FROM lab_project.payment;

```

p_id	t_id	c_id	offer_id	f_id	p_type	total_payment
1	1	1	1	1	Mat Lam Tam	31183
2	2	2	2	2	Sonsing	23100
3	3	3	3	3	Rank	27529
4	4	4	4	4	Quo Lux	871
5	5	5	5	5	Kanlam	39971
6	6	6	6	6	Prodder	19545
7	7	7	7	7	Redhold	47506
8	8	8	8	8	Alphazap	42294

Successfully run. Total query runtime: 56 msec. 100 rows affected.

6. Time

CREATE TABLE "lab_project".time

```

(
    e_id bigint NOT NULL,
    e_start_time time(0) without time zone NOT NULL,
    e_end_time time(0) without time zone NOT NULL,
    PRIMARY KEY (e_name)
);

```

COPY lab_project.time (e_name, e_start_time, e_end_time) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\time.csv'

CSV HEADER;

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects under the schema 'lab_project'. The 'Tables' node is expanded, showing 11 tables: customer, event, event_organizer, event_type, food, manager, offer, payment, refund/cancellation, seat_no, and ticket. The 'ticket' table is currently selected. The main window contains a Query Editor tab with the following SQL code:

```

211 CREATE TABLE "lab_project".time
212 (
213     e_name character varying(100) NOT NULL,
214     e_start_time time(0) without time zone NOT NULL,
215     e_end_time time(0) without time zone NOT NULL,
216     PRIMARY KEY (e_name)
217 );
218
219
220
221
222
223
224
225
226
227
228
229 COPY lab_project.time (e_name, e_start_time, e_end_time) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\time.csv'
230 CSV HEADER;
231
232 SELECT * FROM lab_project.time;
233
234
235

```

Below the Query Editor, a Data Output tab is open, displaying the results of the query. The table has three columns: e_name, e_start_time, and e_end_time. The data is as follows:

	e_name	e_start_time	e_end_time
1	1	07:10:00	20:16:00
2	2	05:45:00	05:44:00
3	3	21:26:00	02:32:00
4	4	18:02:00	19:27:00
5	5	16:46:00	14:15:00
6	6	02:26:00	04:42:00
7	7	00:52:00	16:00:00
8	8	07:05:00	12:40:00

A green success message at the bottom right of the data grid states: "Successfully run. Total query runtime: 57 msec. 100 rows affected."

7. Event

CREATE TABLE "lab_project".event

```

(
    event_id bigint NOT NULL,
    e_type_id integer NOT NULL,
    e_name character varying(100) NOT NULL,
    e_date date NOT NULL,
    e_size integer NOT NULL,
    address_line_1 character varying(50) NOT NULL,
    address_line_2 character varying(50),
    pincode integer NOT NULL,
    e_manager bigint NOT NULL,
    PRIMARY KEY (event_id),

```

CONSTRAINT E_TYPE_ID_FK FOREIGN KEY (e_type_id)

REFERENCES lab_project.event_type (e_type_id) ON UPDATE CASCADE,

CONSTRAINT E_MANAGER_FK FOREIGN KEY (e_manager)

REFERENCES Lab_Project.manager (m_id) ON UPDATE CASCADE,

CONSTRAINT E_NAME_FK FOREIGN KEY (e_name)

REFERENCES lab_project.time (e_name) ON UPDATE CASCADE

);

COPY

lab_project.event(event_id,e_type_id,e_name,e_date,e_size,address_line_1,address_line_2,pincode,e_manager)
FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\event.csv'

CSV HEADER;

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database schema, including Schemas, Tables, and Functions. The main area is the Query Editor, which contains the following SQL code:

```
206     e_manager BIGINT NOT NULL,  
207     PRIMARY KEY (event_id),  
208     CONSTRAINT E_TYPE_ID_FK FOREIGN KEY (e_type_id)  
209         REFERENCES lab_project.event_type (e_type_id) ON UPDATE CASCADE,  
210     CONSTRAINT E_MANAGER_FK FOREIGN KEY (e_manager)  
211         REFERENCES Lab_Project.manager (m_id) ON UPDATE CASCADE,  
212     CONSTRAINT E_NAME_FK FOREIGN KEY (e_name)  
213         REFERENCES lab_project.time (e_name) ON UPDATE CASCADE  
214 );  
215  
216 COPY lab_project.event(event_id,e_type_id,e_name,e_date,e_size,address_line_1,address_line_2,pincode,e_manager) FROM 'E:\Lecture e-books'\  
217 CSV HEADER;  
218  
219  
220 SELECT * FROM lab_project.event;  
221  
222  
223
```

The Data Output tab is active, showing the results of the query. The table has the following columns:

event_id	e_type_id	e_name	e_date	e_size	address_line_1	address_line_2	pincode	e_manager
1	1	1 1	2020-12-21	1370	63869 Fulton Drive	599 Hovde Circle	31397912	1
2	2	2 2	2020-11-25	1533	375 Mendota Street	36 Grover Drive	8807227	2
3	3	3 3	2021-10-14	928	5728 Mayer Terrace	6 Belfuss Court	15428104	3
4	4	4 4	2021-09-05	1182	4623 Sloan Junction	011 Prairie Rose Junction	35102784	4
5	5	5 5	2021-06-27	2108	93 Shopko Drive	9472 Lunder Circle	13859714	5
6	6	6 6	2021-09-22	1921	6206 Maywood Way	1 Columbus Avenue	16070442	6
7	7	7 7	2020-11-21	301	698 Farmco Place	31397912	1	1
8	8	8 8	2020-12-18	2654	4 Waubesa Trail	31397912	1	1

A green success message at the bottom right of the Data Output table says "Successfully run. Total query runtime: 160 msec. 100 rows affected."

8. Event_organizer

```
CREATE TABLE "lab_project".event_organizer
(
    eo_id bigint NOT NULL,
    event_id integer NOT NULL ,
    eo_name character varying(20) NOT NULL,
    password character varying(64) NOT NULL,
    PRIMARY KEY (eo_id),
    CONSTRAINT EVENT_ID_FK FOREIGN KEY (event_id)
        REFERENCES lab_project.event (event_id) ON UPDATE CASCADE
);
```

```
COPY lab_project.event_organizer (eo_id,event_id,eo_name,password) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\event_organizer.csv'
```

CSV HEADER;

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the database structure under "Schemas (3)" and "Tables (11)".
- Query Editor:** Displays the SQL code for creating the table and inserting data from a CSV file.
- Data Output:** Shows the resulting table with 8 rows of data.
- Status Bar:** Shows the message "Successfully run. Total query runtime: 63 msec. 100 rows affected."
- System Tray:** Shows the date and time as 13-11-2021 11:42 PM.

```
CREATE TABLE "lab_project".event_organizer
(
    eo_id bigint NOT NULL,
    event_id integer NOT NULL ,
    eo_name character varying(20) NOT NULL,
    password character varying(64) NOT NULL,
    PRIMARY KEY (eo_id),
    CONSTRAINT EVENT_ID_FK FOREIGN KEY (event_id)
        REFERENCES lab_project.event (event_id) ON UPDATE CASCADE
);
COPY lab_project.event_organizer (eo_id,event_id,eo_name,password) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\event_organizer.csv' CSV HEADER;
SELECT * FROM lab_project.event_organizer;
```

eo_id	event_id	eo_name	password
1	1	Riffpath	L7yPfFdJIS
2	2	Eadel	Sse3FXZx
3	3	Voomm	uAM5mr
4	4	Aivee	tka03DhmWd
5	5	Mynte	KttR69YZ
6	6	Skinder	IcmnVfV
7	7	Agivu	3sTdeLrCdCk
8	8	Voomm	iObakAuC

9. Food

```
CREATE TABLE "lab_project".food
```

```
(
```

```
    f_id bigint NOT NULL,  
    f_name character varying(20) NOT NULL,  
    f_price integer NOT NULL,  
    f_quantity integer NOT NULL,  
    PRIMARY KEY (f_id)
```

```
);
```

```
COPY lab_project.food (f_id, f_name, f_price, f_quantity) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\food.csv'
```

CSV HEADER;

The screenshot shows the pgAdmin 4 interface with the following details:

- Schemas:** lab_project (selected).
- Tables:** customer, event, event_organizer, event_type, food, manager, offer, payment, refund/cancellation, seat_no, ticket.
- Query Editor:** Contains the SQL code for creating the food table and copying data from a CSV file.
- Data Output:** A table showing the data inserted into the food table.
- Status Bar:** Shows a green success message: "Successfully run. Total query runtime: 175 msec. 100 rows affected."

```
123  SELECT * FROM lab_project.customer;  
124  
125  CREATE TABLE "lab_project".food  
126  (  
127      f_id bigint NOT NULL,  
128      f_name character varying(20) NOT NULL,  
129      f_price integer NOT NULL,  
130      f_quantity integer NOT NULL,  
131      PRIMARY KEY (f_id)  
132  );  
133  
134 ;  
135  
136  
137  COPY lab_project.food (f_id, f_name, f_price, f_quantity) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\food.csv'  
138  CSV HEADER;  
139  
140
```

f_id	f_name	f_price	f_quantity
1	Charmene	959	58
2	Hyacinthe	594	16
3	Davida	694	40
4	Verene	391	57
5	Ebba	497	55
6	Tyne	620	9
7	Afton	115	80
8	Othilia	347	52

10. Ticket

```
CREATE TABLE "lab_project".ticket
(
    t_id bigint NOT NULL,
    c_id bigint NOT NULL,
    event_id bigint NOT NULL,
    eo_id bigint NOT NULL,
    no_of_tickets integer NOT NULL,
    seat_no integer NOT NULL,
    price integer NOT NULL,
    food_id bigint,
    p_id bigint NOT NULL,
    PRIMARY KEY (t_id),
    CONSTRAINT C_ID_FK FOREIGN KEY (c_id)
        REFERENCES lab_project.customer (c_id) ON UPDATE CASCADE,
    CONSTRAINT EVENT_ID_FK FOREIGN KEY (event_id)
        REFERENCES lab_project.event (event_id) ON UPDATE CASCADE,
    CONSTRAINT C_seat_ID_FK FOREIGN KEY (seat_no)
        REFERENCES lab_project.seat_no (st_id) ON UPDATE CASCADE
);
COPY lab_project.ticket (t_id, c_id, event_id, eo_id, no_of_tickets, seat_no, price, food_id, p_id) FROM
'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\ticket.csv'
CSV HEADER;
```

```

183   CONSTRAINT C_ID_FK FOREIGN KEY (c_id)
184     REFERENCES lab_project.customer (c_id) ON UPDATE CASCADE,
185   CONSTRAINT EVENT_ID_FK FOREIGN KEY (event_id)
186     REFERENCES lab_project.event (event_id) ON UPDATE CASCADE,
187   CONSTRAINT C_seat_ID_FK FOREIGN KEY (seat_no)
188     REFERENCES lab_project.seat_no (st_id) ON UPDATE CASCADE
189 );
190
191
192
193 COPY lab_project.ticket (t_id, c_id, event_id, eo_id, no_of_tickets, seat_no, price, food_id, p_id) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\refund.csv' HEADER;
194
195
196
197
198 SELECT * FROM lab_project.ticket;
199
200

```

t_id	c_id	event_id	eo_id	no_of_tickets	seat_no	price	food_id	p_id
1	1	1	1	7	1	3734	1	1
2	2	2	2	2	2	1151	2	2
3	3	3	3	2	3	340	3	3
4	4	4	4	5	4	4046	4	4
5	5	5	5	4	5	3685	5	5
6	6	6	6	1	6	1350	6	6
7	7	7	7	3	7	928	7	
8	8	8	8	3	8	1602	8	

Successfully run. Total query runtime: 70 msec. 100 rows affected.

11. Refund/cancellation

CREATE TABLE "lab_project"."refund/cancellation"

(

```

r_id bigint NOT NULL,
p_id bigint NOT NULL,
r_reason character varying(200) NOT NULL,
r_amount bigint NOT NULL,
r_type character varying(20) NOT NULL,
PRIMARY KEY (r_id)

```

);

COPY lab_project.refund/cancellation (r_id, p_id, r_reason, r_amount, r_type) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\refund.csv'

CSV HEADER;

```

154 SELECT * FROM lab_project.order;
155
156 CREATE TABLE "lab_project"."refund/cancellation"
157 (
158     r_id bigint NOT NULL,
159     p_id bigint NOT NULL,
160     r_reason character varying(200) NOT NULL,
161     r_amount bigint NOT NULL,
162     r_type character varying(20) NOT NULL,
163     PRIMARY KEY (r_id)
164 );
165
166
167 COPY lab_project.refund/cancellation (r_id, p_id, r_reason, r_amount, r_type) FROM 'E:\Lecture e-books\Semester - 5\IT214\Lab\Lab 8\ref
168 CSV HEADER';
169
170 SELECT * FROM lab_project.refund/cancellation;
171

```

r_id	p_id	r_reason	r_amount	r_type
1	1	83 Duis consequat dui nec nisi volutpat eleifend. Donec ut dolor. Morbi vel lectus in quam fringilla rhoncus.	529399118503	Latlu
2	2	55 In sagittis dui vel nisl. Duis ac nibh. Fusce lacus purus, aliquet at, feugiat non, pretium quis, lectus.	3019784280420	Redh
3	3	94 Suspendisse potenti. In eleifend quam a odio. In hac habitasse platea dictumst.	1817329507449	Flexic
4	4	62 Duis aliquam convallis nunc. Proin at turpis a pede posuere nonummy. Integer non velit.	1691783142941	Greer
5	5	23 Integer tincidunt ante vel ipsum. Praesent blandit lacinia erat. Vestibulum sed magna at nunc commodo placerat.	1507815331913	Namf
6	6	9 In sagittis dui vel nisl. Duis ac nibh. Fusce lacus purus, aliquet at, feugiat non, pretium quis, lectus.	4744023979582	Job
7	7	46 Duis aliquam convallis nunc. Proin at turpis a pede posuere nonummy. Integer non velit.		

✓ Successfully run. Total query runtime: 61 msec. 10 rows affected.

12. Seat_no

CREATE TABLE seat_no

(

```

st_id integer NOT NULL,
seat_number1 integer DEFAULT NULL,
seat_number2 integer DEFAULT NULL,
seat_number3 integer DEFAULT NULL,
seat_number4 integer DEFAULT NULL,
seat_number5 integer DEFAULT NULL,
seat_number6 integer DEFAULT NULL,
seat_number7 integer DEFAULT NULL,
PRIMARY KEY (st_id)
);
```

```
COPY seat_no (st_id,
seat_number1,seat_number2,seat_number3,seat_number4,seat_number5,seat_number6,seat_number7 ) FROM
'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\seat_no.csv'
```

CSV HEADER;

The screenshot shows the pgAdmin 4 interface with the following details:

- Browser:** Shows the schema tree for the 'public' schema, including tables like auth_group, auth_permission, auth_user, etc.
- Query Editor:** Displays the SQL commands used to create the table and copy data from a CSV file.

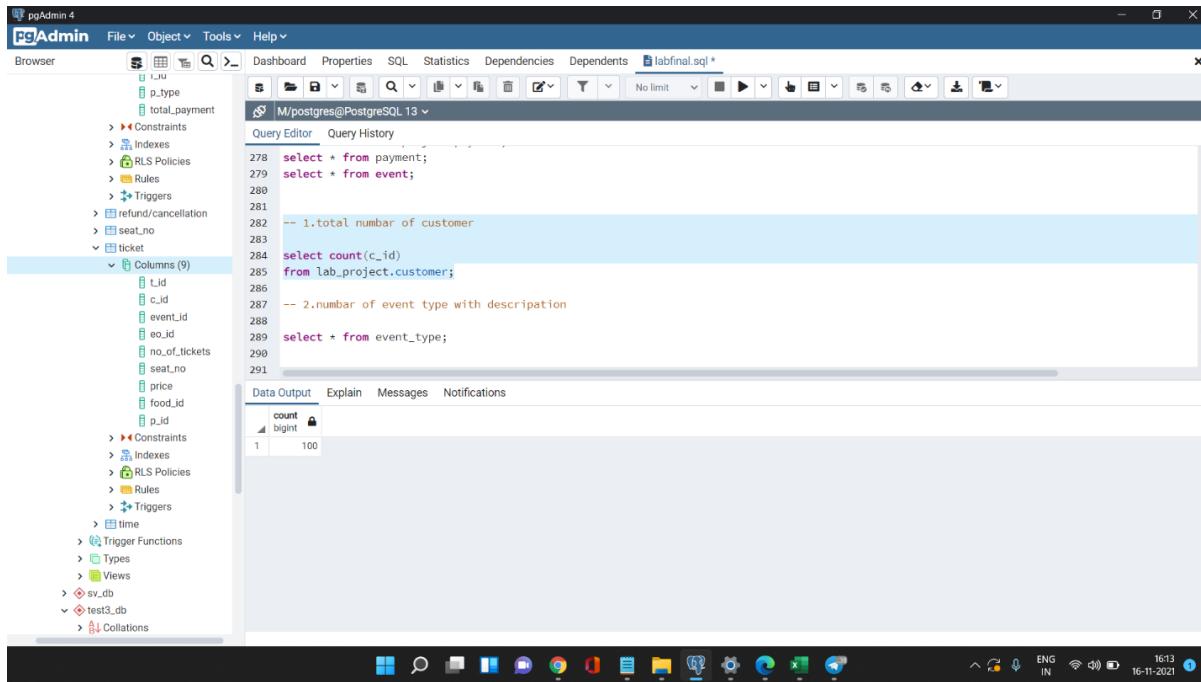

```

118 CREATE TABLE seat_no
119 (
120     st_id integer NOT NULL,
121     seat_number1 integer DEFAULT NULL,
122     seat_number2 integer DEFAULT NULL,
123     seat_number3 integer DEFAULT NULL,
124     seat_number4 integer DEFAULT NULL,
125     seat_number5 integer DEFAULT NULL,
126     seat_number6 integer DEFAULT NULL,
127     seat_number7 integer DEFAULT NULL,
128     PRIMARY KEY (st_id)
129 );
130
131 COPY seat_no (st_id, seat_number1,seat_number2,seat_number3,seat_number4,seat_number5,seat_number6,seat_number7 ) FROM 'E:\Lecture e-books\Samester - 5\IT214\Lab\Lab 8\seat_no.csv'
132 CSV HEADER;
133 select * from seat_no;
134 
```
- Data Output:** A table showing the copied data with 8 rows and columns for seat_number1 through seat_number7.
- Status Bar:** Shows a green message: "Successfully run. Total query runtime: 481 msec. 100 rows affected."

Queries: -

1. Total number of customers in the database.

```
select count(c_id)  
from lab_project.customer;
```



The screenshot shows the pgAdmin 4 interface. On the left is a tree view of the database schema, including tables like t_id, p_type, total_payment, and ticket, along with their columns (t_id, c_id, event_id, eo_id, no_of_tickets, seat_no, price, food_id, p_id). The main window contains the following SQL code:

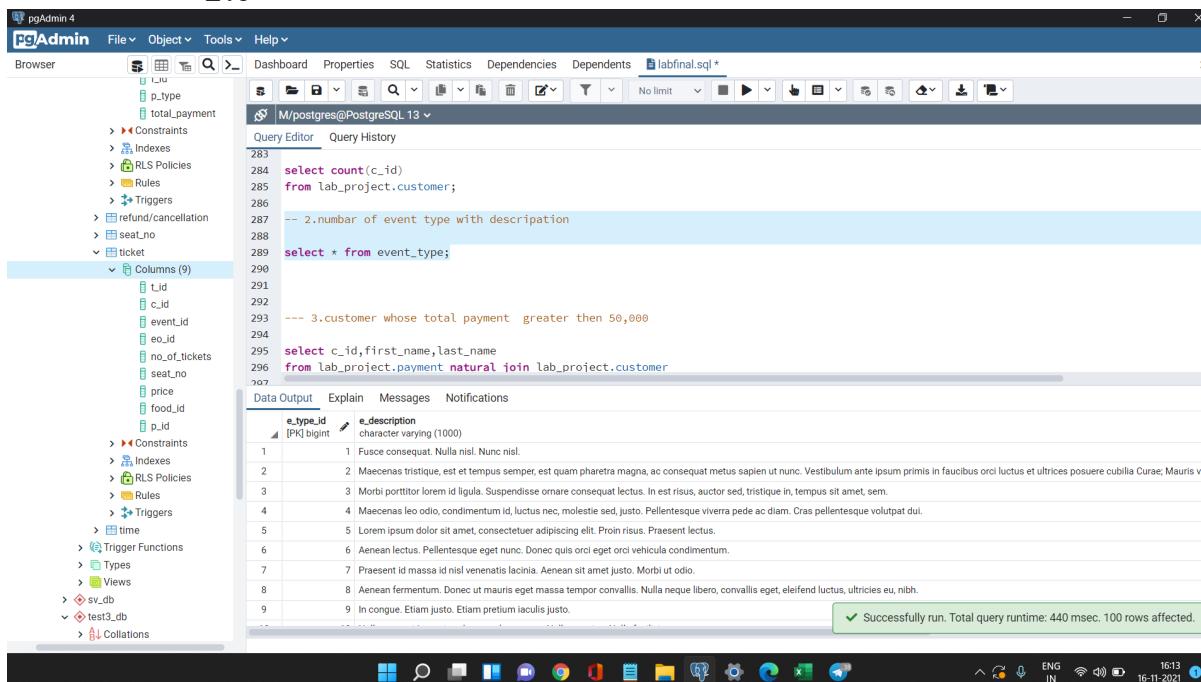
```
278 select * from payment;  
279 select * from event;  
280  
-- 1.total number of customer  
282  
283 select count(c_id);  
from lab_project.customer;  
285  
-- 2.number of event type with description  
286  
288 select * from event_type;  
289  
290  
291
```

The Data Output tab shows the result of the first query:

count	bignum
1	100

2. All different event types.

```
select * from event_type;
```



The screenshot shows the pgAdmin 4 interface. The schema tree is identical to the previous screenshot. The main window contains the following SQL code:

```
283  
284 select count(c_id)  
from lab_project.customer;  
285  
286  
-- 2.number of event type with description  
287  
288 select * from event_type;  
289  
290  
291  
292  
293 --- 3.customer whose total payment greater than 50,000  
294  
295 select c_id,first_name,last_name  
from lab_project.payment natural join lab_project.customer  
296  
297
```

The Data Output tab shows the results of the second query:

e_type_id	e_description
1	Fusce consequat. Nulla nisl. Nunc nisl.
2	Maecenas tristique, est et tempus semper, est quam pharetra magna, ac consequat metus sapien ut nunc. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Mauris vi
3	Morbi porttitor lorem id ligula. Suspendisse ornare consequat lectus. In est risus, auctor sed, tristique in, tempus sit amet, sem.
4	Maecenas leo odio, condimentum id, luctus nec, molestie sed, justo. Pellentesque viverra pede ac diam. Cras pellentesque volutpat dul.
5	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin risus. Praesent lectus.
6	Aenean lectus. Pellentesque eget nunc. Donec quis orci eget orci vehicula condimentum.
7	Prasent id massa id nisl venenatis lacinia. Aenean sit amet justo. Morbi ut odio.
8	Aenean fermentum. Donec ut mauris eget massa tempor convallis. Nulla neque libero, convallis eget, eleifend luctus, ultricies eu, nibh.
9	In congue. Etiam justo. Etiam pretium iaculis justo.

A green success message at the bottom right of the data output pane says "Successfully run. Total query runtime: 440 msec. 100 rows affected."

3. Customer details whose total payment is greater than 50000.

```
select c_id,first_name,last_name  
from lab_project.payment natural join lab_project.customer  
where total_payment > 50000;
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays the database schema with tables like payment, customer, and ticket. The main area is the Query Editor containing the following SQL code:

```
-- 3.customer whose total payment greater than 50,000  
--  
select c_id,first_name,last_name  
from lab_project.payment natural join lab_project.customer  
where total_payment > 50000;  
  
-- 4.list of customer whose event_type_id is 20  
--  
select c_id,first_name,last_name  
from lab_project.ticket natural join lab_project.event natural join lab_project.customer  
where e_type_id=20;
```

The Data Output tab shows the results of the first query:

c_id	first_name	last_name
1	Avictor	Simister
2	Archaimbaud	Adams
3	Gunther	Sterke
4	Viki	Tamburl
5	Duffy	Djuricic
6	Opaline	Shurocks
7	Joseph	Dykins
8	Elisha	McGown
9	Grady	Linscott
10	Haleigh	Sergeant

A green success message at the bottom right indicates: "Successfully run. Total query runtime: 272 msec. 29 rows affected."

4. List of all customers whose event type id is 20.

```
select c_id,first_name,last_name  
from lab_project.ticket natural join lab_project.event natural join lab_project.customer  
where e_type_id=20;
```

The screenshot shows the pgAdmin 4 interface with a query editor containing the following SQL code:

```

297 where total_payment > 50000;
298
299 -- 4.list of customer whose event_type_id is 20
300 select c_id,first_name,last_name
301 from lab_project.ticket natural join lab_project.event natural join lab_project.customer
302 where e_type_id=20;
303
304 --5.list of offer whose discount 40
305
306 select o_id,o_name
307 from offer
308 where discount > 40;
309
310 --6. sum of total payment of all the event
311

```

The results pane shows a single row of data:

c_id	first_name	last_name
20	Rafaelle	Speck

A green success message at the bottom right indicates: "Successfully run. Total query runtime: 364 msec. 1 rows affected."

5. List of all offers which give discount more than 40%.

```
select o_id,o_name
from offer
where discount > 40;
```

The screenshot shows the pgAdmin 4 interface with a query editor containing the same SQL code as the previous screenshot:

```

297 where total_payment > 50000;
298
299 -- 4.list of customer whose event_type_id is 20
300 select c_id,first_name,last_name
301 from lab_project.ticket natural join lab_project.event natural join lab_project.customer
302 where e_type_id=20;
303
304 --5.list of offer whose discount 40
305
306 select o_id,o_name
307 from offer
308 where discount > 40;
309
310 --6. sum of total payment of all the event
311

```

The results pane shows 10 rows of data:

o_id	o_name
1	Iguanatal
2	Small's Rainilly
3	California Sunflower
4	Spreading Sandwort
5	Dead Finish
6	Nutrush
7	Northern Green Orchid
8	Nenesia
9	Black Ironbox
10	Plum

A green success message at the bottom right indicates: "Successfully run. Total query runtime: 260 msec. 21 rows affected."

6. Total payments of all the event.

```
select sum(total_payment)  
from payment;
```

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays a database structure with tables like 'otter', 'p.type', 'total_payment', 'Columns (9)', and 'ticket'. The 'ticket' table is currently selected. In the center, the Query Editor window contains the following SQL code:

```
307 from otter;  
308 where discount > 40;  
309  
310 --6. sum of total payment of all the event  
311  
312 select sum(total_payment)  
313 from payment;  
314  
315  
316  
317 --7.  
318 select c_id  
319 from payment  
320 where p_type='Kanlam';  
321
```

Below the Query Editor, the Data Output tab shows the results of the first query:

sum
3708166

A green message bar at the bottom right indicates: "Successfully run. Total query runtime: 278 msec. 1 rows affected."

7. List of all customers whose payment type is kanlam(dummy data).

```
select c_id  
from payment  
where p_type='Kanlam';
```

pgAdmin 4

Browser

```

    |> Constraints
    |> Indexes
    |> RLS Policies
    |> Rules
    |> Triggers
    |> refund/cancellation
    |> seat_no
    |> ticket
    |> Columns (9)
        |> t_id
        |> c_id
        |> event_id
        |> eo_id
        |> no_of_tickets
        |> seat_no
        |> price
        |> food_id
        |> p_id
    > Constraints
    > Indexes
    > RLS Policies
    > Rules
    > Triggers
    > time
    > Trigger Functions
    > Types
    > Views
    > sv.db
    > test3.db
    > Collations

```

Query Editor Query History

```

313 from payment;
314
315
316
317 --7.
318 select c_id
319 from payment
320 where p_type='Kanlam';
321
322 -- 8.
323 select event_id
324 from event
325 where e_date BETWEEN '2020-01-01' AND '2021-06-01';
326
327

```

Data Output Explain Messages Notifications

c_id	
1	5
2	48

✓ Successfully run. Total query runtime: 349 msec. 2 rows affected.

8. List of all events which took place between 2020-01-01 to 2021-06-01.

```

select event_id
from event
where e_date BETWEEN '2020-01-01' AND '2021-06-01';

```

pgAdmin 4

Browser

```

    |> Constraints
    |> Indexes
    |> RLS Policies
    |> Rules
    |> Triggers
    |> refund/cancellation
    |> seat_no
    |> ticket
    |> Columns (9)
        |> t_id
        |> c_id
        |> event_id
        |> eo_id
        |> no_of_tickets
        |> seat_no
        |> price
        |> food_id
        |> p_id
    > Constraints
    > Indexes
    > RLS Policies
    > Rules
    > Triggers
    > time
    > Trigger Functions
    > Types
    > Views
    > sv.db
    > test3.db
    > Collations

```

Query Editor Query History

```

319 from payment;
320 where p_type='Kanlam';
321
322 -- 8.
323 select event_id
324 from event
325 where e_date BETWEEN '2020-01-01' AND '2021-06-01';
326
327 --9.
328 select event_id
329 from event
330 where e_size > 1000;
331
332 --10.

```

Data Output Explain Messages Notifications

event_id	[PK] bigint
1	1
2	2
3	7
4	8
5	9
6	13
7	14
8	17
9	18
10	19

✓ Successfully run. Total query runtime: 267 msec. 52 rows affected.

9. List of events with event size is greater than 1000.

```
select event_id  
from event  
where e_size > 1000;
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including tables like 'event', 'customer', and 'ticket'. The main area is a 'Query Editor' containing the following SQL code:

```
-- 8.  
select event_id  
from event  
where e_size > 1000;  
-- 9.  
select event_id  
from event  
where e_size > 1000;  
-- 10.  
select c_id,first_name,last_name  
from ticket natural join customer  
where no_of_tickets > 5
```

Below the code, there is a 'Data Output' tab showing a table with one column 'event_id' and ten rows of data:

event_id
1
2
3
4
5
6
7
8
9
10

A green message bar at the bottom right indicates: 'Successfully run. Total query runtime: 186 msec. 75 rows affected.'

10. List of customers who booked more than five tickets at a time.

```
select c_id,first_name,last_name  
from ticket natural join customer  
where no_of_tickets > 5;
```

pgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

    p_id
    p.type
    total_payment
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
  > refund/cancellation
  > seat_no
  > ticket
  > Columns (9)
    t_id
    c_id
    event_id
    eo_id
    no_of_tickets
    seat_no
    price
    food_id
    p_id
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
  > time
  > Trigger Functions
  > Types
  > Views
  > sv.db
  > test3.db
  > Collations

```

Query Editor Query History

```

325 where e_date BETWEEN '2020-01-01' AND '2021-06-01';
326
327 --9.
328 select event_id
329 from event
330 where e_size > 1000;
331
332 --10.
333 select c_id,first_name,last_name
334 from ticket natural join customer
335 where no_of_tickets > 5
336
337
338

```

Data Output Explain Messages Notifications

c_id	first_name	last_name
1	Avictor	Simister
2	Legra	Edmundson
3	Werner	Letts
4	Venus	Glaum
5	Duffy	Djuricic
6	Haleigh	Sergeant
7	Kellen	MacKowle
8	Cassandra	Ottam
9	Dewie	Viegas
10	Glynis	Raigatt

Successfully run. Total query runtime: 343 msec. 23 rows affected.

11. detail of all customers.

Select *

From customer;

pgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

    event
    event_organizer
    event_type
    food
    manager
    offer
    payment
    refund/cancellation
    Columns
    Constraints
    Indexes
    RLS Policies
    Rules
    Triggers
    seat_no
    ticket
    time
  > Trigger Functions (2)
    (insert_cust()
    (insert_payment())
  > Views (5)
    customer_count
    event_count
    m_count
    payment_count
    payment_type_view
  > public
  > sv.db
  > Subscriptions
  > dvrentral
  > postgres
  & Local Group Policy

```

Query Editor Query History

```

264
265 delete from "refund/cancellation";
266
267
268 --11
269 select *
270 from customer;
271
272 --12
273 select count(c_id)
274 from payment
275 where offer_id != 0;
276
277 --13
278 select sum(no_of_tickets)
279 from ticket
280 group by event_id;
281

```

Explain Messages Notifications Data Output

c_id	first_name	last_name	c_email	c_Bdate	c_phno	password	age
1	Avictor	Simister	asimister@ft.com	2004-07-12	6125020777	zJPevAOg9Gb	17
2	Hall	Pauer	hpauer1@nhs.uk	2008-02-03	4791786935	36C0dQZL89	13
3	Grete	Warrington	gwarrington2@trellian.com	2009-12-16	4884626727	U2M1H9x7V5	12
4	Krystal	Rawet	krawet3@mysql.com	2007-11-08	9016615888	vEizzl8O	14
5	Fannie	Winfred	fwinfred4@tumblr.com	2010-12-28	6075169081	DF7Nk4v	11
6	Any	Lindholm	alindholm5@163.com	2006-10-20	6884313863	h0aFBwNGSmv	15
7	Averyl	Buddington	abuddington6@xrea.com	2002-05-01	54%	Successfully run. Total query runtime: 67 msec. 100 rows affected.	11:03 PM
8	Claire	Gaine	cgaine7@forbes.com	2001-10-01	ENG IN	20-11-2021	11:03 PM

12. Number of customers who avail offers.

```
select count(c_id)
from payment
where offer_id != 0;
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects: event, event_organizer, event_type, food, manager, offer, payment, refund/cancellation, seat_no, ticket, time, Trigger Functions (2), Types, Views (5), public, sv_db, Subscriptions, dvrental, and postgres. The main area is the Query Editor, showing the following SQL code:

```
266
267
268 --11
269 select count(c_id)
270 from payment
271 where offer_id in (select offer_id from offer where o_id != 0);
272
273 --12
274 select count(c_id)
275 from payment
276 where offer_id != 0;
277
278 --13
279 select sum(no_of_tickets)
280 from ticket
281 group by event_id;
```

The Data Output tab shows the results of the first query:

count	bigint
1	70

A green message bar at the bottom right indicates: "Successfully run. Total query runtime: 183 msec. 1 rows affected."

13. The total number of tickets for every event.

```
select sum(no_of_tickets)
from ticket
group by event_id;
```

PgAdmin File Object Tools Help

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
  > seat_no
  > ticket
  > time
  > Trigger Functions (2)
    < insert_cust()
    < insert_payment()
  > Types
  > Views (5)
    > customer_count
    > event_count
    > m_count
    > payment_count
    > payment_type_view
  > public
  > sv_db
  > Subscriptions
  > dvrdental
  > postgres
  > I Login/Group Roles

```

Query Editor Query History

```

276 where offer_id := 0;
277
278 --13
279 select sum(no_of_tickets)
280 from ticket
281 group by event_id;
282
283 --14
284 select first_name, last_name
285 from ticket natural join customer natural join food
286 order by f_price ;
287
288 --15
289 select *
290 from customer
291 where first_name = 'Tedi' and last_name = 'Tull';
292

```

Explain Messages Notifications Data Output

	sum	bigint
1	7	
2	3	
3	2	
4	2	
5	4	
6	4	
7	3	
8	2	

Successfully run. Total query runtime: 61 msec. 100 rows affected.

14. name of the customer who orders food,

```

select first_name, last_name
from ticket left outer join customer on ticket.c_id=customer.c_id left outer join food on ticket.food_id=food.f_id
where ticket.food_id!=0
order by f_price ;

```

PgAdmin File Object Tools Help

Browser

```

> f_id
> f_name
> f_price
> f.quantity
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> manager
> offer
> payment
> refund/cancellation
> seat_no
> ticket
  > Columns (9)
    > t_id
    > c_id
    > event_id
    > eo_id
    > no_of_tickets
    > seat_no
    > price
    > food_id
    > p_id
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> time
  > Trigger Functions (2)
    < insert_cust()

```

Query Editor Query History

```

327 from ticket
328 group by event_id;
329
330 --14
331 select first_name, last_name
332 from ticket left outer join customer on ticket.c_id=customer.c_id left outer join food on ticket.food_id=food.f_id
333 where ticket.food_id!=0
334 order by f_price ;
335
336 --15
337 select *
338 from customer
339 where first_name = 'Tedi' and last_name = 'Tull';
340
341 --16
342 select e_name
343 from event

```

Explain Messages Notifications Data Output

	first_name	last_name
1	Werner	Lets
2	Malvina	Twidale
3	Frank	Bull
4	Tabina	Leyden
5	Bette-ann	Beeson
6	Hobard	Forkan
7	Bryn	Mougin
8	Dwight	Feaver

Successfully run. Total query runtime: 84 msec. 70 rows affected.

15. detail of customer whose first name is Tedi and last name is Tull.

```
select *  
from customer  
where first_name = 'Tedi' and last_name = 'Tull';
```

The screenshot shows the pgAdmin 4 interface. On the left is a tree view of database objects under '201901066_db/postgres@PostgreSQL 13'. The 'Trigger Functions (2)' node is selected. In the center is the 'Query Editor' window with the following SQL code:

```
--15  
289 select *  
290 from customer  
291 where first_name = 'Tedi' and last_name = 'Tull';  
292  
--16  
294 select e_name  
295 from event  
296 where e_date between '01-01-2021' and '11-11-2021';  
297  
--17  
299 select max(e_size)  
300 from event;  
301  
302
```

Below the editor is the 'Data Output' tab, which displays the results of the query:

c_id	first_name	last_name	c_email	c_Bdate	c_phno	password	age
1	15	Tedi	Tull	ttulle@phoca.cz	2000-01-03	4669952866	L0PyA04

A green message bar at the bottom right indicates: 'Successfully run. Total query runtime: 65 msec. 1 rows affected.'

16. The event happens between 1-1-2021 to 11-11-2021.

```
select e_name  
from event  
where e_date between '01-01-2021' and '11-11-2021';
```

pgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
> Trigger Functions (2)
  > insert_cust()
  > insert_payment()
> Types
> Views (5)
  > customer_count
  > event_count
  > m_count
  > payment_count
  > payment_type_view
> public
> sv_db
> Subscriptions
> dvrental
> postgres
  & Login/Group Roles

```

Query Editor Query History

```

286 order by f_price ;
287
288 --15
289 select *
290 from customer
291 where first_name = 'Tedi' and last_name = 'Tull';
292
293 --16
294 select e_name
295 from event
296 where e_date between '01-01-2021' and '11-11-2021';
297
298
299 --17
300 select max(e_size)
301 from event;
302

```

Explain Messages Notifications Data Output

e_name	character varying (100)
1	baby
2	standup comedy
3	murder
4	romio and juliat
5	hamlet
6	aheliya
7	lagan
8	bharat natyam

Successfully run. Total query runtime: 71 msec. 47 rows affected.

17. maximum number of capacity for the event,

select max(e_size)
from event;

pgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
> Trigger Functions (2)
  > insert_cust()
  > insert_payment()
> Types
> Views (5)
  > customer_count
  > event_count
  > m_count
  > payment_count
  > payment_type_view
> public
> sv_db
> Subscriptions
> dvrental
> postgres
  & Login/Group Roles

```

Query Editor Query History

```

295 from event
296 where e_date between '01-01-2021' and '11-11-2021';
297
298
299 --17
300 select max(e_size)
301 from event;
302
303 --18
304 select p_type
305 from payment
306 group by p_type;
307
308 --19
309 select sum(r_amount)
310 from "refund/cancellation";
311

```

Explain Messages Notifications Data Output

max	integer
1	2933

Successfully run. Total query runtime: 116 msec. 1 rows affected.

18. Total type of payment.

```
select p_type  
from payment  
group by p_type;
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including event, event_organizer, event_type, food, manager, offer, payment, refund/cancellation, seat_no, ticket, time, Trigger Functions (2), Types, Views (5), and various system objects like public, sv_db, and dvrental. The main window contains a Query Editor with the following SQL code:

```
295 from event  
296 where e_date between '01-01-2021' and '11-11-2021';  
297  
298 --17  
299 select max(e_size)  
300 from event;  
301  
302 --18  
303 select p_type  
304 from payment  
305 group by p_type;  
306  
307  
308 --19  
309 select sum(r_amount)  
310 from "refund/cancellation";  
311
```

The Data Output tab shows the results of the query:

p_type
wallet
UPI
credit card
internet Banking
debit card

A green success message at the bottom right of the main window states: "Successfully run. Total query runtime: 61 msec. 5 rows affected."

19. Sum of the total refunded amount.

```
select sum(r_amount)  
from "refund/cancellation";
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including tables like event, event_type, food, manager, offer, payment, refund/cancellation, and trigger functions insert_cust() and insert_payment(). The main window contains a SQL query editor with the following code:

```

305 from payment
306 group by p_type;
307
308 --19
309 select sum(r_amount)
310 from "refund/cancellation";
311
312 --20
313 select first_name, last_name
314 from customer
315 where age>18;
316
317
318 --complex
319
320 --1
321

```

The Data Output tab shows the result of the query:

	sum
1	317702

A green success message at the bottom right indicates: "Successfully run. Total query runtime: 77 msec. 1 rows affected."

20. name of the customer whose age is greater than 18.

```
select first_name, last_name
from customer
where age>18;
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including tables like event, event_type, food, manager, offer, payment, refund/cancellation, and trigger functions insert_cust() and insert_payment(). The main window contains a SQL query editor with the following code:

```

305 from payment
306 group by p_type;
307
308 --19
309 select sum(r_amount)
310 from "refund/cancellation";
311
312 --20
313 select first_name, last_name
314 from customer
315 where age>18;
316
317
318 --complex
319
320 --1
321

```

The Data Output tab shows the result of the query:

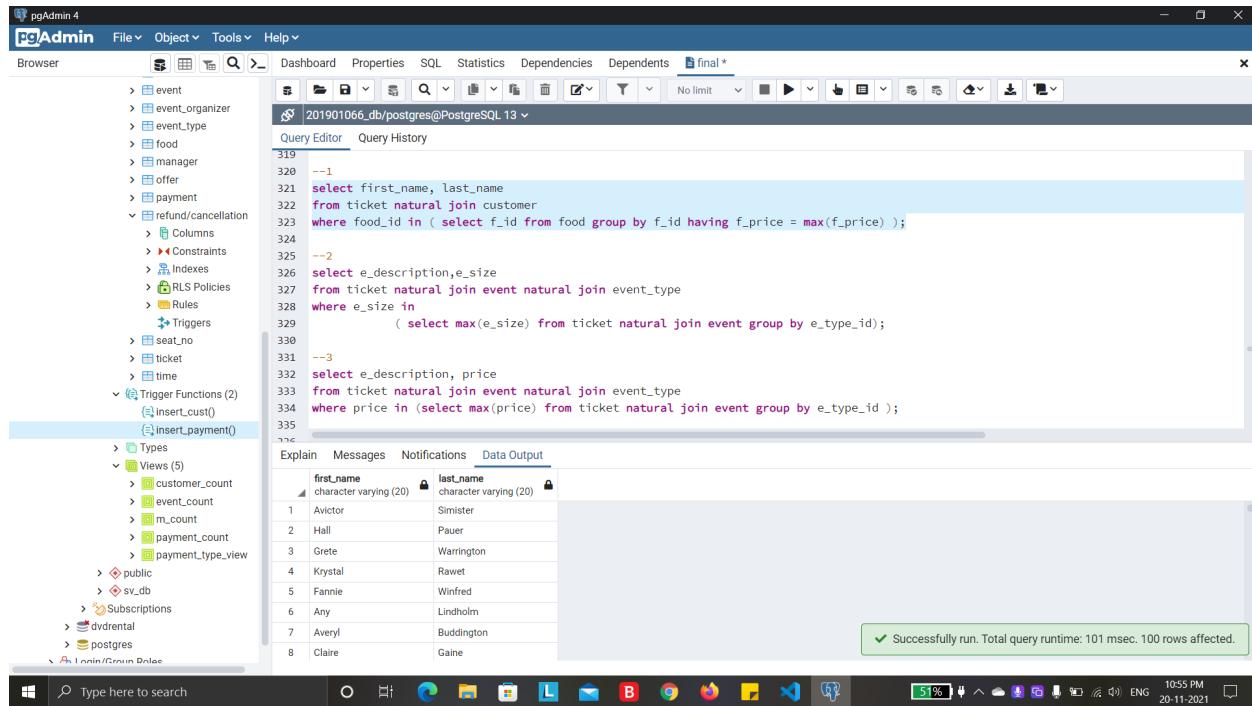
	first_name	last_name
1	Averyl	Buddington
2	Claire	Gaine
3	Legra	Edmundson
4	Tedi	Tull
5	Viki	Tamburi
6	Joseph	Dykins
7	Garner	Lattie
8	Kellen	MacKowle

A green success message at the bottom right indicates: "Successfully run. Total query runtime: 206 msec. 28 rows affected."

Complex queries: -

1. Name of customer who pays the maximum amount for food.

```
select first_name, last_name  
from ticket natural join customer  
where food_id in ( select f_id from food group by f_id having f_price = max(f_price) );
```



The screenshot shows the pgAdmin 4 interface with a query editor window. The query is as follows:

```
--1  
select first_name, last_name  
from ticket natural join customer  
where food_id in ( select f_id from food group by f_id having f_price = max(f_price) );  
--2  
select e_description,e_size  
from ticket natural join event natural join event_type  
where e_size in  
        ( select max(e_size) from ticket natural join event group by e_type_id );  
--3  
select e_description, price  
from ticket natural join event natural join event_type  
where price in (select max(price) from ticket natural join event group by e_type_id );  
--4
```

The results table shows the following data:

	first_name	last_name
1	Avictor	Simister
2	Hall	Pauer
3	Grete	Warrington
4	Krystal	Rawet
5	Fannie	Winfred
6	Any	Lindholm
7	Averyl	Buddington
8	Claire	Gaine

A green success message at the bottom right of the results pane states: "Successfully run. Total query runtime: 101 msec. 100 rows affected."

2. For event type, what is the maximum event size.

```
select e_description,e_size  
from ticket natural join event natural join event_type  
where e_size in  
        ( select max(e_size) from ticket natural join event group by e_type_id );
```

PgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
Trigger Functions (2)
  < insert_cust()
  < insert_payment()
  > Types
  > Views (5)
    > customer_count
    > event_count
    > m_count
    > payment_count
    > payment_type_view
  > public
  > sv_db
  > Subscriptions
  > dvrental
  > postgres
  & Login/Group Roles

```

Query Editor Query History

```

201901066_db/postgres@PostgreSQL 13 ▾
No limit ▾
319
320 --1
321 select first_name, last_name
322 from ticket natural join customer
323 where food_id in ( select f_id from food group by f_id having f_price = max(f_price) );
324
325 --2
326 select e_description, e_size
327 from ticket natural join event natural join event_type
328 where e_size in
329   ( select max(e_size) from ticket natural join event group by e_type_id );
330
331 --3
332 select e_description, price
333 from ticket natural join event natural join event_type
334 where price in (select max(price) from ticket natural join event group by e_type_id );
335
336

```

Explain Messages Notifications Data Output

e_description	e_size
Play	2654
Movie	2904
Circus	2805
Drama	2933
Short film	2578
Live event	2717
Concert	2735

✓ Successfully run. Total query runtime: 64 msec. 7 rows affected.

3. Maximum price of a ticket for every event type.

```

select e_description, price
from ticket natural join event natural join event_type
where price in (select max(price) from ticket natural join event group by e_type_id );

```

PgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
Trigger Functions (2)
  < insert_cust()
  < insert_payment()
  > Types
  > Views (5)
    > customer_count
    > event_count
    > m_count
    > payment_count
    > payment_type_view
  > public
  > sv_db
  > Subscriptions
  > dvrental
  > postgres
  & Login/Group Roles

```

Query Editor Query History

```

201901066_db/postgres@PostgreSQL 13 ▾
No limit ▾
324
325 --2
326 select e_description, e_size
327 from ticket natural join event natural join event_type
328 where e_size in
329   ( select max(e_size) from ticket natural join event group by e_type_id );
330
331 --3
332 select e_description, price
333 from ticket natural join event natural join event_type
334 where price in (select max(price) from ticket natural join event group by e_type_id );
335
336 --4
337 select first_name, last_name, no_of_tickets
338 from ticket natural join customer
339 where no_of_tickets = ( select max(no_of_tickets) from ticket );
340

```

Explain Messages Notifications Data Output

e_description	price
Movie	4593
Concert	4389
Live event	4026
Drama	4819
Play	4571
Short film	4807
Circus	4518

✓ Successfully run. Total query runtime: 101 msec. 7 rows affected.

4. Name of customer who booked maximum number of tickets.

```
select first_name, last_name, no_of_tickets  
from ticket natural join customer  
where no_of_tickets = ( select max(no_of_tickets) from ticket );
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including tables like event, event_type, food, manager, offer, payment, refund/cancellation, RLS Policies, Rules, Triggers, seat_no, ticket, time, and trigger functions insert_cust() and insert_payment(). Below these are views such as customer_count, event_count, m_count, payment_count, and payment_type_view. The right pane contains the Query Editor with the following SQL code:

```
334 where price in (select max(price) from ticket natural join event group by e_type_id);  
335  
336 --4  
337 select first_name, last_name, no_of_tickets  
338 from ticket natural join customer  
339 where no_of_tickets = ( select max(no_of_tickets) from ticket );  
340  
341 --5  
342 select first_name, last_name  
343 from customer  
344 where length(password) > 8;  
345  
346 --6  
347 select e_name  
348 from time  
349 where e_start_time between '08:00:00 AM' and '11:59:59 AM';  
350
```

Below the code, the Data Output tab is selected, showing a table with the results:

	first_name	last_name	no_of_tickets
1	Avictor	Slimister	7
2	Legra	Edmundson	7
3	Wenher	Letts	7
4	Venus	Glaum	7
5	Duffy	Djuricic	7
6	Haleigh	Sergeant	7
7	Kellen	Mackowle	7
8	Cassandra	Ottam	7

A green success message at the bottom right of the data grid states: "Successfully run. Total query runtime: 92 msec. 15 rows affected."

5. name of the customer whose password length is greater than 8.

```
select first_name, last_name  
from customer  
where length(password) > 8;
```

PgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
> Trigger Functions (2)
  < insert_cust()
  < insert_payment()
> Types
> Views (5)
  > customer_count
  > event_count
  > m_count
  > payment_count
  > payment_type_view
> public
> sv_db
> Subscriptions
> dvrental
> postgres
  & Login/Group Roles

```

Query Editor Query History

```

334 where price in (select max(price) from ticket natural join event group by e_type_id );
335
336 --4
337 select first_name, last_name, no_of_tickets
338 from ticket natural join customer
339 where no_of_tickets = ( select max(no_of_tickets) from ticket );
340
341 --5
342 select first_name, last_name
343 from customer
344 where length(password) > 8;
345
346 --6
347 select e_name
348 from time
349 where e_start_time between '08:00:00 AM' and '11:59:59 AM';
350

```

Explain Messages Notifications Data Output

	first_name	last_name
1	Avitor	Simister
2	Hall	Pauer
3	Grete	Warrington
4	Any	Lindholm
5	Dael	Moreing
6	Legra	Edmundson
7	Venus	Glum
8	Gussie	De Gowe

Successfully run. Total query runtime: 59 msec. 62 rows affected.

6. Name of event happens between 8:00 AM to 11:59 AM.

```

select e_name
from time
where e_start_time between '08:00:00 AM' and '11:59:59 AM';

```

PgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
> Trigger Functions (2)
  < insert_cust()
  < insert_payment()
> Types
> Views (5)
  > customer_count
  > event_count
  > m_count
  > payment_count
  > payment_type_view
> public
> sv_db
> Subscriptions
> dvrental
> postgres
  & Login/Group Roles

```

Query Editor Query History

```

334 from ticket natural join customer
335 where no_of_tickets = ( select max(no_of_tickets) from ticket );
336
337 --5
338 select first_name, last_name
339 from customer
340 where length(password) > 8;
341
342 --6
343 select e_name
344 from time
345 where e_start_time between '08:00:00 AM' and '11:59:59 AM';
346
347 --7
348 select distinct(o_name), discount
349 from payment natural join offer
350 where discount = ( select max(discount) from offer );
351
352
353
354
355

```

Explain Messages Notifications Data Output

	e_name
1	lagan
2	bharat natyam
3	great royal
4	rajkamal
5	farewell
6	woyzek
7	dance divane
8	fabulous 5

Successfully run. Total query runtime: 89 msec. 16 rows affected.

7. Name of offer which gives the maximum discount.

```
select distinct(o_name), discount  
from payment natural join offer  
where discount = ( select max(discount) from offer );
```

The screenshot shows the pgAdmin 4 interface. The left sidebar is titled 'Browser' and contains a tree view of database objects:

- event
- event_organizer
- event_type
- food
- manager
- offer
- payment
- refund/cancellation
 - Columns
 - Constraints
 - Indexes
 - RLS Policies
 - Rules
 - Triggers
- seat_no
- ticket
- time
- Trigger Functions (2)
 - insert_cust()
 - insert_payment()
- Types
- Views (5)
 - customer_count
 - event_count
 - m_count
 - payment_count
 - payment_type_view
- public
- sv_db
- Subscriptions
- dvrental
- postgres
- Latin/Cyrillic Dialects

The main area has tabs for 'Query Editor' and 'Query History'. The 'Query Editor' tab is active, showing the following SQL code:

```
330 from ticket natural join customer
331 where no_of_tickets = ( select max(no_of_tickets) from ticket );
340
341 --5
342 select first_name, last_name
343   from customer
344 where length(password) > 8;
345
346 --6
347 select e_name
348   from time
349 where e_start_time between '08:00:00 AM' and '11:59:59 AM';
350
351 --7
352 select distinct(o_name), discount
353   from payment natural join offer
354 where discount = ( select max(discount) from offer )
355
```

The 'Data Output' tab is selected, displaying the results of the last query:

e_name
[PK] character varying (100)
1 lagan
2 bharat natyam
3 great royal
4 rajkamal
5 farewell
6 woyzeck
7 dance diwane
8 fabulous 5

A green success message at the bottom right says: "Successfully run. Total query runtime: 89 msec. 16 rows affected."

8. maximum amount for food by customer.

```
select max(f_price * f_quantity) as cnt  
from ticket natural join customer natural join food;
```

PgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
Trigger Functions (2)
  < insert_cust()
  < insert_payment()
  > Types
  > Views (5)
    > customer_count
    > event_count
    > m_count
    > payment_count
    > payment_type_view
  > public
  > sv_db
  > Subscriptions
  > dvrental
  > postgres
  & Login/Group Roles

```

Query Editor Query History

```

348   from time
349   where e_start_time between '08:00:00 AM' and '11:59:59 AM';
350
351   --7
352   select distinct(o_name), discount
353   from payment natural join offer
354   where discount = ( select max(discount) from offer );
355
356   --8
357   select max(f_price * f_quantity) as cnt
358   from ticket natural join customer natural join food;
359
360   --9
361   select eo_name
362   from event_organizer
363   where length(password) > 10;
364

```

Explain Messages Notifications Data Output

cnt	eo_name
1	81686

✓ Successfully run. Total query runtime: 93 msec. 1 rows affected.

9. Name of event organize whose password length is greater than 10.

```

select eo_name
from event_organizer
where length(password) > 10;

```

PgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

```

> event
> event_organizer
> event_type
> food
> manager
> offer
> payment
> refund/cancellation
  > Columns
  > Constraints
  > Indexes
  > RLS Policies
  > Rules
  > Triggers
> seat_no
> ticket
> time
Trigger Functions (2)
  < insert_cust()
  < insert_payment()
  > Types
  > Views (5)
    > customer_count
    > event_count
    > m_count
    > payment_count
    > payment_type_view
  > public
  > sv_db
  > Subscriptions
  > dvrental
  > postgres
  & Login/Group Roles

```

Query Editor Query History

```

353   from payment natural join offer
354   where discount = ( select max(discount) from offer );
355
356   --8
357   select max(f_price * f_quantity) as cnt
358   from ticket natural join customer natural join food;
359
360   --9
361   select eo_name
362   from event_organizer
363   where length(password) > 10;
364
365
366   --10
367   select e_description, sum(no_of_tickets)
368   from ticket natural join event natural join event_type
369

```

Explain Messages Notifications Data Output

eo_name
character varying (20)
1 Alive
2 Agivu
3 Kwlith
4 Wordify
5 Youfeed
6 Chatterbridge
7 Viva
8 Skidoo

✓ Successfully run. Total query runtime: 68 msec. 30 rows affected.

10. Total number of tickets for every event type.

```
select e_description, sum(no_of_tickets)
from ticket natural join event natural join event_type
group by e_type_id;
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including tables like event, event_organizer, food, manager, offer, payment, refund/cancellation, and various triggers and functions. The main area is a query editor window titled 'Query History' containing the following SQL code:

```
362 select e_description
363   from event_organizer
364  where length(password) > 10;
365
366 --10
367 select e_description, sum(no_of_tickets)
368   from ticket natural join event natural join event_type
369  group by e_type_id;
370
371 --11
372 select first_name, last_name
373   from "refund/cancellation" natural join payment natural join customer
374  where r_amount > 50000;
375
376 --12
377 select e_description, e_name, price
378   from ticket natural join event natural join event_type
379
```

Below the code, there is an 'Explain' tab, a 'Messages' tab, a 'Notifications' tab, and a 'Data Output' tab. The 'Data Output' tab is selected and shows a table with the results of the query:

e_description	sum
Movie	89
Concert	24
Live event	39
Drama	42
Play	67
Short film	79
Circus	42

A green success message at the bottom right of the data output area states: 'Successfully run. Total query runtime: 80 msec. 7 rows affected.'

11. Name of customer who gets a refund.

```
select first_name, last_name
from "refund/cancellation" natural join payment natural join customer
where r_amount > 3000;
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a tree view of database objects, including tables like event, event_organizer, event_type, food, manager, offer, payment, refund/cancellation, seat_no, ticket, time, and trigger functions. A specific trigger function, insert_payment(), is selected. The main pane contains a query editor with the following SQL code:

```
201901066.db/postgres@PostgreSQL 13
SELECT e_description, sum(no_of_tickets)
FROM ticket NATURAL JOIN event NATURAL JOIN event_type
GROUP BY e_type_id;
--10
SELECT first_name, last_name
FROM "refund/cancellation" NATURAL JOIN payment NATURAL JOIN customer
WHERE r_amount > 3000;
--11
SELECT e_description, e_name, price
FROM ticket NATURAL JOIN event NATURAL JOIN event_type
--12
```

The Data Output tab is active, showing the results of the second query:

	first_name	last_name
1	Archaibaud	Adams
2	Werner	Lets
3	Joseph	Dykins
4	Norrie	Merritt
5	Orlando	Rubanenko
6	Rowena	Abrahamoff
7	Jonis	Dearing
8	Othella	Fyrth

A green message box at the bottom right indicates: "Successfully run. Total query runtime: 70 msec. 10 rows affected."

12. Event type event name with the highest ticket price.

```
select e_description, e_name, price  
from ticket natural join event natural join event_type  
where price in (select max(price) from ticket);
```

The screenshot shows the pgAdmin 4 interface. The left sidebar displays a database browser with various schema and table structures. The main area features a query editor with the following SQL code:

```
368 FROM ticket natural join event natural join event_type
369 group by e_type_id;
370
371 --11
372 select first_name, last_name
373 from "refund/cancellation" natural join payment natural join customer
374 where r_amount > 3000;
375
376 --12
377 select e_description, e_name, price
378 from ticket natural join event natural join event_type
379 where price in (select max(price) from ticket);
380
381
382 --13
383 select first_name, last_name, f_price
384 from payment natural join customer natural join food
385
```

The query editor includes tabs for Explain, Messages, Notifications, and Data Output. The Data Output tab shows the results of the query:

e_description	e.name	price
character varying (1000)	character varying (100)	integer
1 Drama	ramayana	4819

A green checkmark icon in the bottom right corner indicates that the query was successfully run. The status bar at the bottom right shows the date and time as 20-11-2021 10:58 PM.

13. customer who pay highest for food.

```
select first_name, last_name, f_price  
from payment natural join customer natural join food  
order by f_price DESC  
limit 1;
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** Includes File, Object, Tools, Help, and a search bar.
- Left Sidebar:** Shows the database schema structure under "Browser".
- Query Editor:** Displays the SQL query being run.
- Data Output:** Shows the results of the query, which is a single row:

first_name	last_name	f_price
Duffy	Djuricic	994
- Status Bar:** Shows a green checkmark indicating success and the message "Successfully run. Total query runtime: 125 msec. 1 rows affected."
- Taskbar:** Shows the Windows taskbar with various application icons and system status.

14. name of the manager who handles a most number of events.

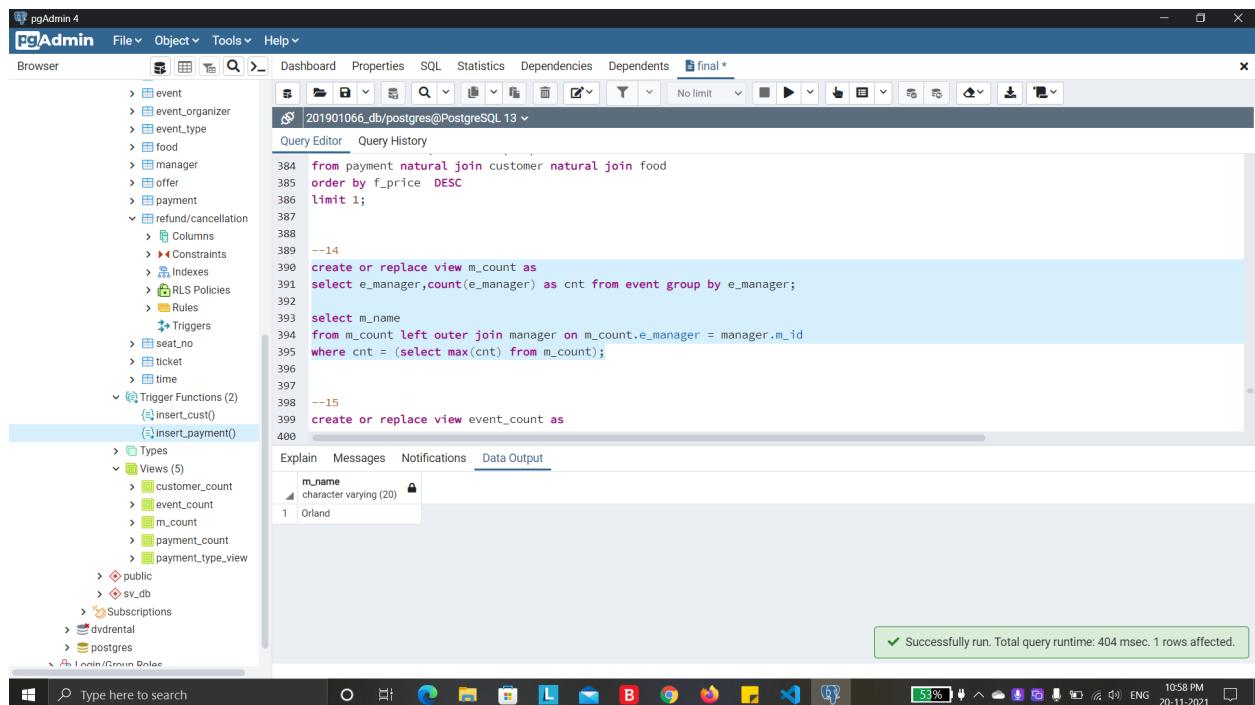
create or replace view m_count as

```
select e_manager,count(e_manager) as cnt from event group by e_manager;
```

```
select m_name
```

```
from m_count left outer join manager on m_count.e_manager = manager.m_id
```

```
where cnt = (select max(cnt) from m_count);
```



The screenshot shows the pgAdmin 4 interface with the following details:

- Toolbar:** Includes File, Object, Tools, Help, Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and a final* button.
- Query Editor:** Contains the SQL code for creating views and selecting manager names based on event counts.
- Results:** Shows the output of the query, displaying a single row for 'Orland'.
- Status Bar:** Shows a green success message: "Successfully run. Total query runtime: 404 msec. 1 rows affected."
- System Tray:** Shows battery level at 59%, network status, and system date/time (20-11-2021 10:58 PM).

15. Total number of events for every event type.

create or replace view event_count as

```
select e_type_id, count(event_id) as cnt from event group by e_type_id;
```

```
select event_type.e_type_id,e_description,cnt
```

```
from event_count left outer join event_type on event_count.e_type_id = event_type.e_type_id
```

```
where cnt = (select max(cnt) from event_count);
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays database objects like 'event', 'event_organizer', 'event_type', etc., and a 'Trigger Functions' section containing 'insert_cust()' and 'insert_payment()'. The 'Views' section lists 'customer_count', 'event_count', 'm_count', 'payment_count', and 'payment_type_view'. The 'Query Editor' tab is active, showing the SQL code for creating the 'event_count' view and then selecting data from it. The 'Data Output' tab shows the results of the query, which consists of two rows: 'Movie' (e_type_id 1) and 'Short film' (e_type_id 6), both with a count of 19. A green success message at the bottom right indicates the query ran successfully with a runtime of 136 msec and 2 rows affected.

```
395 where cnt = (select max(cnt) from m_count);
396
397
398 --15
399 create or replace view event_count as
400 select e_type_id, count(event_id) as cnt from event group by e_type_id;
401
402
403 select event_type.e_type_id,e_description,cnt
404 from event_count left outer join event_type on event_count.e_type_id = event_type.e_type_id
405 where cnt = (select max(cnt) from event_count);
406
407 --16
408 create or replace view customer_count as
409 select e_type_id, count(no_of_tickets) as cnt from ticket natural join event group by e_type_id;
410
411
```

e_type_id	e_description	cnt
1	Movie	19
2	Short film	19

✓ Successfully run. Total query runtime: 136 msec. 2 rows affected.

16. Total number of tickets for every event type.

create or replace view customer_count as

```
select e_type_id, count(no_of_tickets) as cnt from ticket natural join event group by e_type_id;
```

```
select event_type.e_type_id,e_description,cnt
```

```
from customer_count left outer join event_type on customer_count.e_type_id = event_type.e_type_id;
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects including tables like event, event_type, and views like customer_count and payment_count. In the center, the 'Query Editor' pane contains the following SQL code:

```
484 from event_count left outer join event_type on event_count.e_type_id = event_type.e_type_id
485 where cnt = (select max(cnt) from event_count);
486
487 --16
488 create or replace view customer_count as
489 select e_type_id, count(no_of_tickets) as cnt from ticket natural join event group by e_type_id;
490
491 select event_type.e_type_id,e_description,cnt
492 from customer_count left outer join event_type on customer_count.e_type_id = event_type.e_type_id;
493
494 --17
495 create or replace view payment_count as
496 select e_type_id, sum(total_payment) as cnt from payment natural join ticket natural join event group by e_type_id;
497
498
```

Below the code, the 'Data Output' tab is selected, showing a table with the following data:

e_type_id	e_description	cnt
1	Movie	19
2	Concert	6
3	Live event	11
4	Drama	12
5	Play	18
6	Short film	19
7	Circus	15

A green success message at the bottom right of the output area reads: "Successfully run. Total query runtime: 100 msec. 7 rows affected."

17. total amount received for every event type.

create or replace view payment_count as

```
select e_type_id, sum(total_payment) as cnt from payment natural join ticket natural join event group by e_type_id;
```

```
select event_type.e_type_id,e_description,cnt
```

```
from payment_count left outer join event_type on payment_count.e_type_id = event_type.e_type_id;
```

The screenshot shows the PgAdmin 4 interface. On the left is the object browser tree, which includes nodes for event, event_organizer, event_type, food, manager, offer, payment, refund/cancellation, RLS Policies, Rules, Triggers, seat_no, ticket, time, Trigger Functions (2), Types, Views (5), public, sv_db, Subscriptions, dvrental, and postgres. The 'payment' node is expanded, showing its sub-tables: customer_count, event_count, m_count, payment_count, and payment_type_view.

The main window contains a query editor with the following SQL code:

```
414
415
416 --17
417 create or replace view payment_count as
418 select e_type_id, sum(total_payment) as cnt from payment natural join ticket natural join event group by e_type_id;
419
420
421 select event_type.e_type_id,e_description,cnt
422 from payment_count left outer join event_type on payment_count.e_type_id = event_type.e_type_id;
423
424 --18
425 select p_type, count(p_type)
426 from payment
427 group by p_type;
428
429 --19
430
```

Below the query editor is a data output pane showing the results of the second query:

e_type_id	e_description	cnt
1	Movie	619643
2	Concert	166676
3	Live event	170247
4	Drama	419634
5	Play	553942
6	Short film	665774
7	Circus	345209

A green success message at the bottom right of the data output pane says "Successfully run. Total query runtime: 72 msec. 7 rows affected."

18. different payment types used by customers.

```
select p_type, count(p_type)
from payment
group by p_type;
```

The screenshot shows the pgAdmin 4 interface. On the left is the 'Browser' pane displaying database schema objects like event, event_organizer, food, manager, offer, payment, refund/cancellation, RLS Policies, Rules, Triggers, seat_no, ticket, time, Trigger Functions, Types, Views, and various sub-objects. The 'Query Editor' pane contains the following SQL code:

```
419
420 select event_type.e_type_id,e_description,cnt
421 from payment_count left outer join event_type on payment_count.e_type_id = event_type.e_type_id;
422
423 --18
424
425 select p_type, count(p_type)
426 from payment
427 group by p_type;
428
429
430 create or replace view payment_type_view as
431 select p_type, sum(total_payment) as cnt from payment group by p_type;
432
433 select *
434 from payment_type_view
435
```

The 'Data Output' tab shows the results of the query:

p_type	count
wallet	24
UPI	16
credit card	20
internet Banking	19
debit card	21

A green success message at the bottom right of the query editor says: "Successfully run. Total query runtime: 144 msec. 5 rows affected."

19. total amount received by every payment method.

```
create or replace view payment_type_view as  
select p_type, sum(total_payment) as cnt from payment group by p_type;
```

```
select *  
from payment_type_view  
order by cnt;
```

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects including event, event_organizer, event_type, food, manager, offer, payment, refund/cancellation, seat_no, ticket, time, Trigger Functions (2), Types, Views (5), and public. The 'Trigger Functions (2)' node has two entries: '\$insert_cust()' and '\$insert_payment()'. The 'Views (5)' node contains 'customer_count', 'event_count', 'm_count', 'payment_count', and 'payment_type_view'. The 'payment_type_view' entry is highlighted with a blue selection bar. The main window shows a 'Query Editor' tab with the following SQL code:

```
--19  
428 create or replace view payment_type_view as  
429 select p_type, sum(total_payment) as cnt from payment group by p_type;  
430  
431 select *  
432 from payment_type_view  
433 order by cnt;  
434  
435 --20  
436 create or replace function insert_refund()  
437 returns trigger  
438 as $refund_insert_triger$  
439 begin  
440 update "refund/cancellation"  
441 set r_amount = ( select total_payment from payment where payment.p_id = new.p_id )  
442 where "refund/cancellation".p_id = new.p_id;
```

Below the code, the 'Data Output' tab is selected and shows a table with the following data:

p_type	cnt
UPI	433555
internet Banking	594720
debit card	618748
credit card	632167
wallet	661935

A green success message at the bottom right of the query editor says: 'Successfully run. Total query runtime: 123 msec. 5 rows affected.'

20. create a trigger for getting refund amount from customer payment details.

```
create or replace function insert_refund()  
returns trigger  
as $refund_insert_triger$  
begin  
update "refund/cancellation"  
set r_amount = ( select total_payment from payment where payment.p_id = new.p_id )  
where "refund/cancellation".p_id = new.p_id;  
return new;  
end  
$refund_insert_triger$  
language plpgsql;
```

```

create trigger refund_insert_triger
after insert or update on "refund/cancellation"
for each row
when (pg_trigger_depth() < 1)--!
execute function insert_refund();

```

```
select * from "refund/cancellation";
```

The screenshot shows the pgAdmin 4 interface with the following details:

- Left Panel (Browser Tree):** Shows the database schema structure, including tables like event, event_organizer, food, manager, offer, payment, and refund/cancellation.
- Query Editor:** Displays the SQL code for creating a trigger and executing it. The code is as follows:

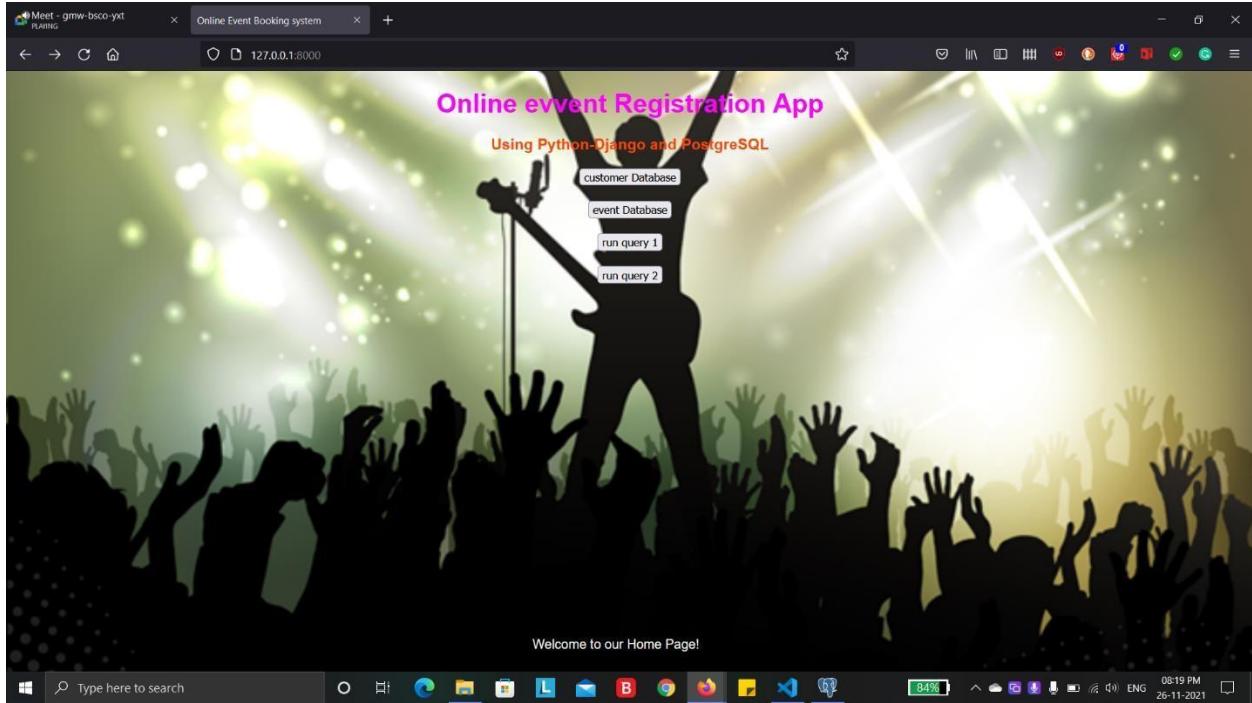

```

442 update "refund/cancellation"
443 set r_amount = ( select total_payment from payment where payment.p_id = new.p_id )
444 where "refund/cancellation".p_id = new.p_id;
445 return new;
446 end
447 $refund_insert_triger$
448 language plpgsql;
449
450
451 create trigger refund_insert_triger
452 after insert or update on "refund/cancellation"
453 for each row
454 when (pg_trigger_depth() < 1)--!
455 execute function insert_refund();
456
457 select * from "refund/cancellation";
458
      
```
- Result Grid:** Shows the results of the query, which is an empty table named "refund/cancellation" with columns r_id, p_id, r_reason, r_amount, and r_type. The grid has 7 rows of sample data.
- Status Bar:** Shows the message "Query returned successfully in 128 msec." and "Successfully run. Total query runtime: 62 msec. 10 rows affected."
- System Tray:** Shows battery level at 54%, network status, and system time (11:00 PM, 20-11-2021).

Section 7: Project Code with output screenshots

Screenshots of Website for online event registration:

(1) Homepage



(2) Customer Database

Show Customer Data

	customer id	first name	last name	email ID	birth date	phone no	password	age	
1	Avictor	Simister	asimister0@ft.com		July 12, 2004	6125020777	zJPevAOg9Gb	17	Edit delete
2	Hall	Pauer	hpauer1@nhs.uk		Feb. 3, 2008	4791786935	36C0dQZL89	13	Edit delete
3	Grete	Warrington	gwarrington2@trellian.com		Dec. 16, 2009	4884626727	UZM1Hv9x7V5	12	Edit delete
4	Krystal	Rawet	krawet3@mysql.com		Nov. 8, 2007	9016615888	vEizzl8O	14	Edit delete
5	Fannie	Winfred	twinfred4@tumblr.com		Dec. 28, 2010	6075169081	Df7Nx4v	11	Edit delete
6	Any	Lindholm	alindholm5@163.com		Oct. 20, 2006	6884313863	h0aFBrwNGSrv	15	Edit delete
7	Averyl	Buddington	abuddington6@rea.com		May 13, 2002	9348757591	7wWuS33S	19	Edit delete
8	Claire	Gaine	cgaine7@forbes.com		Oct. 9, 2001	6194207607	Z81hVocD	20	Edit delete
9	Archaimbaud	Adams	adams8@deliciousdays.com		March 11, 2003	8085300910	40SGA7z	18	Edit delete
10	Dael	Moreing	dmoreing9@linkedin.com		Nov. 26, 2010	5126793256	wQ8wwv9aDd6	11	Edit delete
11	Hardy	Gowrich	hgowricha@independent.co.uk		Aug. 11, 2003	7801740743	KfI0z9	18	Edit delete
12	Legra	Edmundson	ledmundsonb@lizjournals.com		Aug. 12, 2001	5584879607	DLBbjRGYu	20	Edit delete
13	Wernher	Letts	wletts@chicagotribune.com		June 5, 2004	8815510407	KU2lzlX	17	Edit delete
14	Gunther	Sterke	gsterker@nationalgeographic.com		Dec. 6, 2010	4824335489	20AyE4	11	Edit delete
15	Tedi	Tull	ttullo@phoca.cz		Jan. 3, 2000	4669952866	L0PyA04	21	Edit delete
16	Venus	Glaum	vglaum@nydailynews.com		Jan. 3, 2005	6772731380	aEWjY3j2SD3G	16	Edit delete
17	Gussie	De Gowe	gdegoweg@cocolog-nifty.com		March 21, 2007	6967087165	aHNtqWc6z	14	Edit delete
18	Viki	Tamburi	vtamburi@storify.com		Sept. 3, 2001	2436675559	Ei2W4u	20	Edit delete
19	Duffy	Djuricic	djuricic@macromedia.com		May 19, 2009	3099184155	emKgIyY	12	Edit delete
20	Rafaellie	Speek	rspeekj@themeforest.net		March 12, 2004	6371654474	Ti71m17x29	17	Edit delete
21	Opaline	Shurrocks	oshrrocks@columbia.edu		Aug. 8, 2010	7192738528	DxpoS6bS9NXU	11	Edit delete
22	Monique	Bellzner	mbellzner1@fc2.com		March 1, 2004	4687372354	7WTQITSSz	17	Edit delete
23	Joseph	Dykins	jdykinsm@statcounter.com		Feb. 3, 2002	3114930008	r0fvXlRo	19	Edit delete
24	Nowell	Veevers	nveeversn@yandex.ru		March 3, 2005	6284683237	s4NbTEnpiv	16	Edit delete
25	Hugh	Pietranek	hpietraneko@ow.ly		Aug. 24, 2009	6051111824	ZvjMqKib	12	Edit delete

(3) Insert in the customer database

Insert Customer Data

Customer ID	101
Customer First Name	Kenil
Customer Last Name	Bhingradiya
Customer Email adress	kenil001@gmail.com
Customer birth date	26 / 01 / 2002
Customer phone number	12345678
Customer password	sdgb426@
Customer age	20
insert	CustomerKenil saved successfully!!

(4) Event Database

event_id	event type id	event name	event date	event size	address line 1	address line 2	pincode	event manager
1	1	baby	Feb. 28, 2021	1370	63869 Fulton Drive	599 Hovde Circle	31397912	1
2	3	standup comedy	Aug. 3, 2021	1533	375 Mendota Street	36 Grover Drive	8807227	2
3	6	dhindhora	Sept. 26, 2020	928	5728 Mayer Terrace	6 Bellfuss Court	15428104	3
4	1	murder	May 18, 2021	1182	4623 Sloan Junction	011 Prairie Rose Junction	35102784	4
5	5	romio and juliat	Feb. 24, 2021	2108	93 Shopko Drive	9472 Lunder Circle	13859714	5
6	5	hamlet	Oct. 1, 2021	1921	6206 Maywood Way	1 Columbus Avenue	16070442	6
7	6	aheliya	Feb. 4, 2021	301	698 Farmco Place	37 Manitowish Crossing	30724527	7
8	5	angles in america	Jan. 15, 2020	2654	4 Waubesa Trail	10 Pankratz Junction	34661776	8
9	1	veer	Feb. 20, 2020	193	737 Dakota Plaza	24 Arizona Circle	29122232	9
10	1	lagan	Aug. 19, 2021	256	85 Corry Parkway	54 Johnson Pass	13293165	10
11	3	bharat natyam	Sept. 29, 2021	1174	8 Hoard Junction	5858 Old Shore Park	7250389	11
12	5	look back in anger	Nov. 10, 2021	652	50 Shopko Plaza	91629 Kinsman Avenue	10081357	12
13	6	toxic	Jan. 3, 2020	1871	0 Harper Parkway	5084 Elgar Pass	25294112	13
14	4	idea unlimited	Jan. 2, 2021	2305	52257 Hintze Crossing	4 Jackson Way	5470250	7
15	7	great royal	July 5, 2020	228	8430 Basil Point	326 Di Loreto Court	8578740	15
16	6	kriti	April 13, 2021	45	7979 Goodland Parkway	88 Pond Place	37476100	16
17	7	rajkamal	July 13, 2020	1463	4 Sunfield Plaza	4610 Pierstorff Plaza	6640296	17
18	6	the maid	May 19, 2020	2130	88 Hovde Avenue	8518 Canary Trail	30718080	18
19	6	positive	Jan. 15, 2021	548	87 Luster Center	61995 Moland Trail	24888722	19
20	4	farewell	Feb. 28, 2021	2198	290 Scoville Crossing	301 Melvin Circle	13448348	20
21	6	ouch	May 5, 2021	199	6827 Mallory Avenue	5 Gale Center	35660459	21
22	7	great raimen	June 1, 2020	716	90189 Fuller Terrace	11 Clyde Gallagher Terrace	16997343	22
23	5	woyzeck	May 22, 2021	1945	86333 Vermont Road	2838 Hanson Street	26331080	7
24	5	waiting for godot	Aug. 6, 2021	321	8 Ohio Drive	904 Myrtle Drive	4707237	24
25	1	satyameva jayte	June 7, 2020	883	71 Bayside Pass	9 5th Drive	9372696	25

(5) Insert in Event Database

Event ID	101
Event Type ID	1
Event Name	Razi
Event Date	28 / 01 / 2002
Event Size	250
Event address line 1	fdg stnru ttryun r6j4
Event address line 2	rynr rtu5
Event Pincode	395006
Event manager	12
Insert	event Razi saved successfully!!!

(6) Query 1: Display all the customers whose age is above 18.

The screenshot shows a Windows desktop environment with a web browser window open. The browser title bar reads "Meet - gmw-bsco-yxt PLAIN" and "Query 1". The URL in the address bar is "127.0.0.1:8000/runQueryCustomer?". The main content area displays a table titled "Show 18+ customers". The table has columns: customer id, first name, last name, email ID, birth date, phone no, password, and age. The data consists of 64 rows of customer information, with the last row highlighted in green. The browser's taskbar at the bottom shows various pinned icons and the system tray indicates it's 08:23 PM on 26-11-2021 with battery level at 82%.

customer id	first name	last name	email ID	birth date	phone no	password	age
7	Averyl	Buddington	abuddington6@xrea.com	May 13, 2002	9348757591	7wWuS33S	19
8	Claire	Gaine	cgaine7@forbes.com	Oct. 9, 2001	6194207607	Z81hVocD	20
9	Archaimbaud	Adams	aadams8@deliciousdays.com	March 11, 2003	8085300910	405GA7z	18
11	Hardy	Gowrich	hgowricha@independent.co.uk	Aug. 11, 2003	7801740743	KrlOz9	18
12	Legra	Edmundson	ledmundsonb@bizjournals.com	Aug. 12, 2001	5584879607	DLBbRGYu	20
15	Tedi	Tull	ttulle@phoca.cz	Jan. 3, 2000	4669952866	L0PyA04	21
18	Viki	Tamburi	vtamburh@storify.com	Sept. 3, 2001	2436675559	E12W4u	20
23	Joseph	Dykins	jdykinsm@statcounter.com	Feb. 3, 2002	3114930008	rOfvXIRo	19
29	Garner	Lattie	glatties@paypal.com	Oct. 3, 2001	6312998676	WG4hZjAJ8L	20
30	Brendis	Stenhouse	bstenhouse@whitehouse.gov	April 22, 2003	3603402623	DexYUdV	18
32	Hank	Matiasiek	hmatiasiek@ow.ly	Aug. 2, 2003	9631893135	W3DV0TK0	18
34	Kellen	MacKowle	kmackowlex@wired.com	March 28, 2000	2336951512	IanMw2CPI	21
36	Dewitt	Viegas	dviegasz@feedburner.com	May 6, 2002	9923126899	AAvghLzn	19
37	Renaldo	Gethings	rgethings10@list-manage.com	May 29, 2001	6138935378	uP3fJ4jX5F	20
39	Kaia	Ludron	kliudron12@stumbleupon.com	July 20, 2002	7791301450	qmk6jVNQqa	19
41	Alvan	Fores	afores14@360.cn	Sept. 17, 2002	8708801186	6YCIPzpjNId	19
44	Savina	Doy	sdoyle17@infoseek.co.jp	May 23, 2001	4972228230	1bk7Rv6	20
45	Elizabeth	Mclan	emcian18@theatlantic.com	Sept. 13, 2001	2291242680	nslpBLZwDR	20
46	Norrie	Meritt	nmerritt19@eventbrite.com	Oct. 9, 2000	9138868701	K9Ra9xKHIM	21
47	Falito	Di Boldi	fdboldi1a@nationalgeographic.com	June 12, 2000	3094926567	WgDTTpXA	21
51	Tabina	Leyden	tleyden1e@example.com	June 20, 2003	5678316908	kkOacCLB	18
52	Sileas	Khoter	skhoter1f@instagram.com	April 20, 2001	2926079234	rVjzTMxTcfAi	20
53	Tandy	Fyldes	tfyldes1g@stumbleupon.com	Oct. 19, 2003	4923528856	aNydrgrk88	18
56	Alif	Oris	aoris1j@digg.com	Feb. 5, 2001	7383221711	eka54mShk2	20
59	Wilton	Ianne	wianne1m@amazonaws.com	Feb. 25, 2002	4688718502	ulJTh6	19
64	Alfy	Bland	abland1r@acquirethisname.com	Nov. 22, 2003	3566902996	losRhcYiwQ	18

(7) Query 2: Display the total number of booked tickets event-wise.

The screenshot shows a Windows desktop environment with a web browser window open. The browser title bar reads "Meet - gmw-bsco-yxt PLAIN" and "Query 2". The URL in the address bar is "127.0.0.1:8000/runQueryEvent?". The main content area displays a table titled "Total booked tickets for all events". The table has two columns: Event Name and Total registered Tickets. The data consists of 25 rows of event names and their corresponding ticket counts, with the last row highlighted in green. The browser's taskbar at the bottom shows various pinned icons and the system tray indicates it's 08:23 PM on 26-11-2021 with battery level at 82%.

Event Name	Total registered Tickets
krish 3	4
great raimen	3
waiting for godot	3
idea unlimited	1
backstreet boys	4
gadheda	4
great royal	1
badla	5
the upcoming	7
maha bharat	3
fenoes	5
rama mandal	7
dhindhora	2
rembo	4
bhuj the prie of india	2
nayantara	7
shershah	3
look back in anger	7
basics of photography	6
war	7
satyamave jayte	3
gemini	2
hamlet	1
apple phone	5
farewell	1
beyond the horizon	4

CODES FOR WEBPAGE USING PYTHON AND DJANGO: -

1) Insert, Delete, Customer And query

```
from django.shortcuts import render
from Event_Registration.models import CustModel,EventModel
from django.contrib import messages
from Event_Registration.forms import Custforms,Eventforms
from django.db import connection

def homepage(request):
    return render(request, 'Homepage.html')

def showcust(request):
    showall=CustModel.objects.all()
    return render(request, 'Index.html', {"data":showall})

def InsertCust(request):
    if request.method == "POST":
        if request.POST.get('c_id') and request.POST.get('first_name') and request.POST.get('last_name') and
request.POST.get('c_email') and request.POST.get('c_Bdate') and request.POST.get('c_phno') and
request.POST.get('password') and request.POST.get('age'):
            saverecord=CustModel()
            saverecord.c_id = request.POST.get('c_id')
            saverecord.first_name = request.POST.get('first_name')
            saverecord.last_name = request.POST.get('last_name')
            saverecord.c_email = request.POST.get('c_email')
            saverecord.c_Bdate = request.POST.get('c_Bdate')
            saverecord.c_phno = request.POST.get('c_phno')
            saverecord.password = request.POST.get('password')
            saverecord.age = request.POST.get('age')
            saverecord.save()
            messages.success(request, 'Customer' + saverecord.first_name + ' saved successfully!!!')
            return render(request, 'insert.html')
    else:
        return render(request, 'insert.html')

def EditCust(request, c_id):
```

```

editmpobject = CustModel.objects.get(c_id=c_id)
return render(request, 'Edit.html', {"CustModel":editmpobject})

def UpdateCust(request, c_id):
    updatecustmor=CustModel.objects.get(c_id=c_id)
    form=Custforms(request.POST, instance=updatecustmor)
    if form.is_valid():
        form.save()
        messages.success(request, 'record updated Successfully!!')
    return render(request, 'Edit.html', {"CustModel":updatecustmor})

def DeleteCust(request, c_id):
    deletecust=CustModel.objects.get(c_id=c_id)
    deletecust.delete()
    showdata=CustModel.objects.all()
    return render(request, "Index.html", {"data": showdata})

def sheweview(request):
    showall=EventModel.objects.all()
    return render(request, 'IndexEvent.html', {"data":showall})

def Insertevent(request):
    if request.method == "POST":
        if request.POST.get('event_id') and request.POST.get('e_type_id') and request.POST.get('e_name') and request.POST.get('e_date') and request.POST.get('e_size') and request.POST.get('address_line_1') and request.POST.get('address_line_2') and request.POST.get('pincode') and request.POST.get('e_manager'):
            saverecord=EventModel()
            saverecord.event_id = request.POST.get('event_id')
            saverecord.e_type_id = request.POST.get('e_type_id')
            saverecord.e_name = request.POST.get('e_name')
            saverecord.e_date = request.POST.get('e_date')
            saverecord.e_size = request.POST.get('e_size')
            saverecord.address_line_1 = request.POST.get('address_line_1')
            saverecord.address_line_1 = request.POST.get('address_line_2')
            saverecord.pincode = request.POST.get('pincode')
            saverecord.e_manager = request.POST.get('e_manager')
            saverecord.save()

```

```

        messages.success(request, 'event ' + saverecord.e_name + ' saved successfully!!!')
        return render(request, 'InsertEvent.html')

    else:
        return render(request, 'InsertEvent.html')

def EditEvent(request, event_id):
    edittmpobject = EventModel.objects.get(event_id=event_id)
    return render(request, 'EditEvent.html', {"EventModel":edittmpobject})

def UpdateEvent(request, event_id):
    updateEvent=EventModel.objects.get(event_id=event_id)
    form=Eventforms(request.POST, instance=updateEvent)
    if form.is_valid():
        form.save()
        messages.success(request, 'record updated Successfully!!')
        return render(request, 'EditEvent.html', {"EventModel":updateEvent})

def DeleteEvent(request, event_id):
    deleteEvent=EventModel.objects.get(event_id=event_id)
    deleteEvent.delete()
    showdata=EventModel.objects.all()
    return render(request, "IndexEvent.html", {"data": showdata})

def runQueryCustomer(request):
    raw_query = "select * from customer where age>=18 order by c_id;"

    cursor = connection.cursor()
    cursor.execute(raw_query)
    alldata=cursor.fetchall()

    return render(request,'runQueryCustomer.html',{"data":alldata})

def runQueryEvent(request):
    raw_query = "select e_name, sum(no_of_tickets) from ticket natural join event group by e_name;"
```

```

cursor = connection.cursor()
cursor.execute(raw_query)
alldata=cursor.fetchall()

return render(request,'runQueryEvent.html',{"data":alldata})

```

URLs

```

from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('showcust', views.showcust, name = "showcust"),
    path('insert', views.InsertCust, name="InsertTmp"),
    path('Edit/<int:c_id>', views.EditCust, name="EditCust"),
    path('Update/<int:c_id>', views.UpdateCust, name="UpdateCust"),
    path('Delete/<int:c_id>', views.DeleteCust, name="DeleteCust"),
    path('runQueryCustomer',views.runQueryCustomer,name="runQueryCustomer"),

    path("",views.homepage,name="homepage"),

    path('showevent', views.showevent, name = "showevent"),
    path('insertevent', views.Insertevent, name="Insertevent"),
    path('Editevent/<int:event_id>', views.EditEvent, name="EditEvent"),
    path('Updateevent/<int:event_id>', views.UpdateEvent, name="UpdateEvent"),
    path('Deleteevent/<int:event_id>', views.DeleteEvent, name="DeleteEvent"),
    path('runQueryEvent',views.runQueryEvent,name="runQueryEvent"),
]

```

Setting modulo for frontend

```
from django.db import models

class CustModel(models.Model):
    c_id=models.BigIntegerField(primary_key=True)
    first_name=models.CharField(max_length=20)
    last_name=models.CharField(max_length=20)
    c_email=models.CharField(max_length=40)
    c_Bdate=models.DateField()
    c_phno=models.BigIntegerField()
    password=models.CharField(max_length=64)
    age=models.IntegerField()
    class Meta:
        db_table = "customer"

class EventModel(models.Model):
    event_id=models.BigIntegerField(primary_key=True)
    e_type_id=models.BigIntegerField()
    e_name=models.CharField(max_length=100)
    e_date=models.DateField()
    e_size=models.IntegerField()
    address_line_1=models.CharField(max_length=50)
    address_line_2=models.CharField(max_length=50)
    pincode=models.IntegerField()
    e_manager=models.IntegerField()
    class Meta:
        db_table = "event"
```

Connect Frontend with Backend

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'LabProject',  
        'USER': 'postgres',  
        'PASSWORD': 'admin',  
        'HOST': 'localhost',  
        'PORT': '5432',  
    }  
}
```

Link for entire code:-

https://drive.google.com/drive/folders/1jBXk1oACNrgbvBCPIa5U-gsK5e47_SQS?usp=sharing