**Name:** Jaya Sabarish Reddy Remala

**Net Id:** jr6421

NYU Tandon School of Engineering

Computer Science and Engineering

CS6083, Fall 2023

**Assignment -2**

**Q-1:**

**(a) Draw an ER diagram that is consistent with this relational schema. Identify any weak entities within the schema and specify the cardinalities for all relationships.**

A notebook with writing on it

Description automatically generated

**(b) Create the schema in a database system. Choose appropriate attribute types, and define primary keys, foreign keys, and other constraints. You may use any relational database system, as long as** **it supports basic SQL, views, and some sort of triggers. Write the create table statements.**

CREATE TABLE Customers (

customer\_id INT(5) PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100),

phone\_number VARCHAR(15)

);

CREATE TABLE Events (

event\_id INT(5) PRIMARY KEY AUTO\_INCREMENT,

venue\_id INT(5) NOT NULL,

event\_date DATE NOT NULL,

start\_time TIMESTAMP NOT NULL,

end\_time TIMESTAMP NOT NULL,

FOREIGN KEY (venue\_id) REFERENCES Venues(venue\_id) on delete cascade

);

CREATE TABLE Events\_Comedians (

event\_id INT(5),

comedian\_id INT(5),

PRIMARY KEY AUTO\_INCREMENT (event\_id, comedian\_id),

FOREIGN KEY (event\_id) REFERENCES Events(event\_id) on delete cascade,

FOREIGN KEY (comedian\_id) REFERENCES Comedians(comedian\_id) on delete cascade

);

CREATE TABLE Comedians (

comedian\_id INT(5) PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

bio VARCHAR(4000),

phone VARCHAR(15) NOT NULL

);

CREATE TABLE Venues (

venue\_id INT(5) PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

address VARCHAR(100) NOT NULL,

contact\_info VARCHAR(15) NOT NULL,

capacity INT(5) NOT NULL

);

CREATE TABLE Reservations (

res\_id INT(5) PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT(5) NOT NULL,

ticket\_id INT(5) NOT NULL,

res\_date DATE NOT NULL,

tickets\_count INT(5) NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id) on delete cascade,

FOREIGN KEY (ticket\_id) REFERENCES Tickets(ticket\_id) on delete cascade

);

CREATE TABLE Reviews (

review\_id INT(5) PRIMARY KEY AUTO\_INCREMENT,

event\_id INT(5) NOT NULL,

customer\_id INT(5) NOT NULL,

rating INT(2) NOT NULL,

review\_text VARCHAR(4000),

FOREIGN KEY (event\_id) REFERENCES Events(event\_id) on delete cascade,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id) on delete cascade

);

CREATE TABLE Tickets (

ticket\_id INT(5) PRIMARY KEY AUTO\_INCREMENT,

event\_id INT(5) NOT NULL,

ticket\_type VARCHAR(50) NOT NULL,

price DECIMAL(8, 2) NOT NULL,

num\_tickets INT(5) NOT NULL,

FOREIGN KEY (event\_id) REFERENCES Events(event\_id) on delete cascade

);

**(c) Write the following SQL queries and execute them on your database. Show the queries and the screenshot of the results:**

*1. List all events along with the total number of tickets sold (reserved) for each event.*

SELECT e.event\_id, ifnull(sum(r.tickets\_count),0) total\_tickets FROM events e left join tickets t on e.event\_id = t.event\_id LEFT JOIN reservations r on r.ticket\_id = t.ticket\_id GROUP BY e.event\_id;

A screen shot of a computer

Description automatically generated

*2. Rank comedians based on the number of events they participate in. You should list the*

*comedians’ IDs and names from highest to lowest number of appearances.*

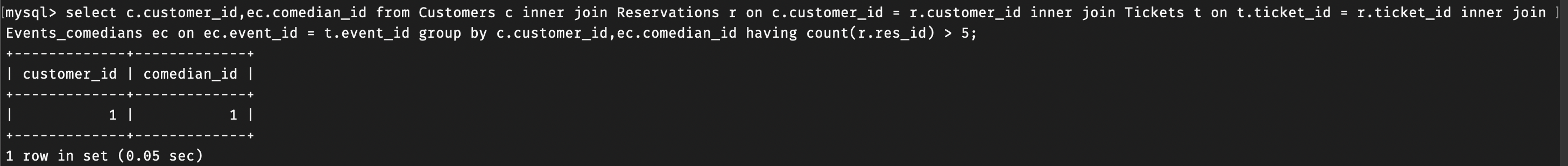
SELECT ec.comedian\_id, concat(c.first\_name," ",c.last\_name) as name , count(\*) as appearances FROM Events\_comedians ec INNER JOIN Comedians c on c.comedian\_id = ec.comedian\_id GROUP BY ec.comedian\_id ORDER BY appearances DESC LIMIT 10;

*A screen shot of a computer

Description automatically generated*

*3. List all pairs of customers and comedians where the customer made reservations for more*

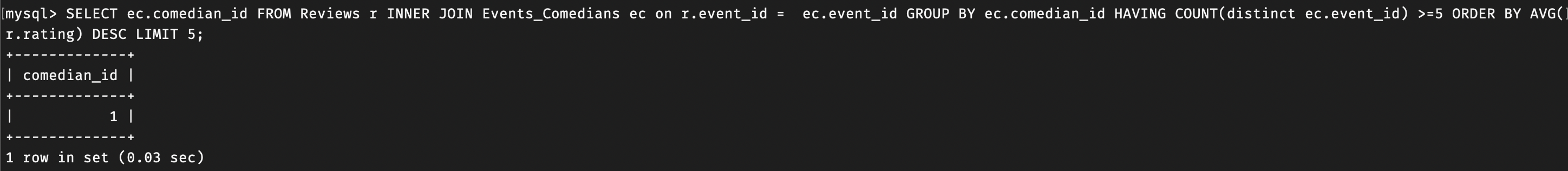
*than 5 different events in which the comedian appeared.*

select c.customer\_id,ec.comedian\_id from CUSTOMERS c INNER JOIN RESERVATIONS r on c.customer\_id = r.customer\_id INNER JOIN TICKETS t on t.ticket\_id = r.ticket\_id INNER JOIN EVENTS\_COMEDIANs ec on ec.event\_id = t.event\_id GROUP BY c.customer\_id,ec.comedian\_id having count(distinct ec.event\_id) > 5

*4. Find the top 5 comedians with the highest average rating based on their reviews, among*

*those comedians who have appeared in at least 5 different events.*

SELECT ec.comedian\_id FROM Reviews r INNER JOIN Events\_Comedians ec on r.event\_id = ec.event\_id GROUP BY ec.comedian\_id HAVING COUNT(distinct ec.event\_id) >=5 ORDER BY AVG(r.rating) DESC LIMIT 5;



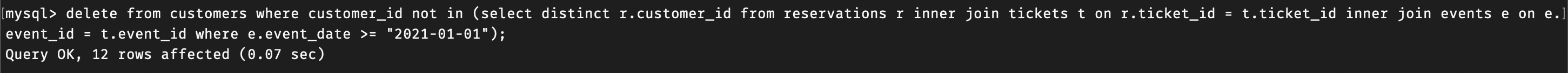
(d) Write the following SQL delete queries and execute them on your database. Show the delete

queries.

1. *Delete customers who have not attended any events after 2020 from the Customers table.*

delete from customers where customer\_id not in (select distinct r.customer\_id from reservations r inner join tickets t on r.ticket\_id = t.ticket\_id inner join events e on e.event\_id = t.event\_id where e.event\_date >= "2021-01-01")

**Assumption**: Foreign key with constraints can be worked for deletion via “ON DELETE CASCADE”.



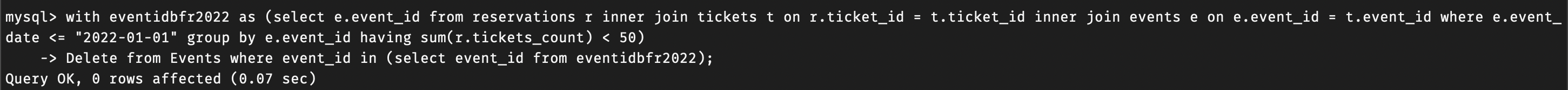
*2. Delete events that happened before 2022 and had less than 50 attendees from the Events*

*table.*

with eventidbfr2022 as (select e.event\_id from reservations r inner join tickets t on r.ticket\_id = t.ticket\_id inner join events e on e.event\_id = t.event\_id where e.event\_date <= "2022-01-01" group by e.event\_id having sum(r.tickets\_count) < 50)

Delete from Events where event\_id in (select event\_id from eventidbfr2022);

**Assumption**: Foreign key with constraints can be worked for deletion via “ON DELETE CASCADE”.

****

(e) Write the following SQL update and insert queries and execute them on your database. Show the

update queries.

*1. Update the end\_time of all future events held in venues with a capacity greater than 500*

*to be one hour later, unless there is another event starting within an hour after the original*

*end\_time of the event.*

With TEvents AS

( SELECT e.event\_id

FROM Events AS e JOIN Venues AS v ON e.venue\_id = v.venue\_id

WHERE v.capacity > 500

AND e.event\_date > CURDATE() -- Only future events

AND NOT EXISTS (

SELECT 1 FROM Events AS other WHERE other.venue\_id = e.venue\_id

AND other.event\_date = e.event\_date

AND other.event\_id <> e.event\_id -- Exclude the same event

AND TIMEDIFF(other.start\_time, e.end\_time) <= '01:00:00'

) )

UPDATE Events e SET e.end\_time = ADDTIME(e.end\_time, '01:00:00') -- Add one hour

WHERE e.event\_id IN (SELECT event\_id FROM TEvents);

**Assumption:** Events happened at same venue will update.

A computer screen with white text

Description automatically generated

*2. Suppose the company would like to thank its most loyal customers, by giving them free*

*tickets for an event. Write a query to insert into the Reservation table a reservation for*

*one VIP ticket, for all customers who have reserved more than 20 tickets during 2023, for*

*the first future event that has enough tickets of type VIP available for all of these*

*customers.*

insert into reservations(customer\_id,ticket\_id,res\_date,tickets\_count)

With Customerids as (

Select customer\_id from Reservations r inner join tickets t on r.ticket\_id = t.ticket\_id

inner join events e on e.event\_id = t.event\_id where e.event\_date >= '2023-01-01' and e.event\_date <= '2023-12-31' group by r.customer\_id having sum(r.tickets\_count) > 20

),

FutureEvents as(

Select t.ticket\_id as ticket\_id from tickets t inner join events e on t.event\_id = e.event\_id

where e.event\_date > CURRENT\_DATE and num\_tickets > (select count(\*) from Customerids) and t.ticket\_type = "VIP" order by e.event\_date ASC limit 1

),

insert\_vals as( select c.customer\_id as customer\_id,t.ticket\_id as ticket\_id

from Customerids c cross join FutureEvents t)

select customer\_id,ticket\_id,CURRENT\_DATE,1 from insert\_vals;

*A computer screen with white text

Description automatically generated*

A black screen with numbers

Description automatically generated

**(f)** Write expressions in Relational Algebra for queries 1 - 4 in part (c).

A close up of a notebook

Description automatically generated

**(g)** Consider adding an attribute called availability to the Tickets table, which stores the number of

tickets that are still available (i.e., have not been reserved yet). This attribute is set to num\_tickets

when a record in Tickets is first created. Create a trigger that automatically checks if the number

of tickets requested in a new reservation is still available. If no, the new reservation should be

rejected; otherwise, the availability column in the Tickets table should be updated appropriately.

CREATE TRIGGER ValidateTicketAvailability

BEFORE INSERT ON Reservations

FOR EACH ROW

BEGIN

DECLARE update\_query\_tickets INT;

DECLARE curr\_tickets INT;

-- Tickets in updated query

SELECT NEW.num\_tickets INTO update\_query\_tickets;

-- Current available tickets

SELECT IFNULL(num\_tickets, 0) INTO curr\_tickets

FROM Tickets

WHERE ticket\_id = NEW.ticket\_id;

-- Validation

IF update\_query\_tickets > curr\_tickets THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Not enough tickets available for this reservation';

ELSE

-- Update the availability in the Tickets table or initialize it if it doesn't exist

UPDATE Tickets

SET availability = IFNULL(availability, num\_tickets) - update\_query\_tickets

WHERE ticket\_id = NEW.ticket\_id;

END IF;

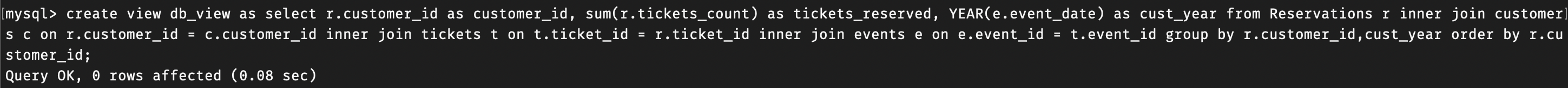
END;

**(h)** Create a view that shows for each customer and each year the total number of tickets they have

reserved during the year. The view should contain as attributes the customer\_id, the year, and the

total number of tickets.

create view db\_view as select r.customer\_id as customer\_id, sum(r.tickets\_count) as tickets\_reserved, YEAR(e.event\_date) as cust\_year from Reservations r inner join customers c on r.customer\_id = c.customer\_id inner join tickets t on t.ticket\_id = r.ticket\_id inner join events e on e.event\_id = t.event\_id group by r.customer\_id,cust\_year order by r.customer\_id;



**(i)** Utilizing the previously created view, output all users who have reserved more than 100 tickets

total (meaning, adding across all years).

select customer\_id, sum(tickets\_reserved) as total\_tickets from db\_view group by customer\_id having total\_tickets > 100;

A black screen with white text

Description automatically generated

**Assumption**: After delete,update and insert commands performed, there might be a small mismatch wrt original dataset comparing the rows. As insert is performed, count of tickets with be +1.

**Q-2:**

(a) Design a database for the above scenario using the ER model. Draw the ER diagram, show the

cardinalities of all relationships, and identify primary keys and any weak entities. List any assumptions you have as well.

A paper with writing on it

Description automatically generated

(b) Convert your ER diagram into a relational schema. Identify all tables, attributes, primary keys, and

foreign keys.

CREATE TABLE Customers (

customer\_id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

phone\_number VARCHAR(15),

email\_address VARCHAR(100)

);

CREATE TABLE Boats (

boat\_id INT AUTO\_INCREMENT PRIMARY KEY,

boat\_name VARCHAR(50),

boat\_class VARCHAR(50) NOT NULL,

max\_capacity INT NOT NULL

);

CREATE TABLE Tags (

tag\_id INT AUTO\_INCREMENT PRIMARY KEY,

tag\_description VARCHAR(50) NOT NULL

);

CREATE TABLE Boat\_Tags (

boat\_id INT,

tag\_id INT,

PRIMARY KEY (boat\_id, tag\_id),

FOREIGN KEY (boat\_id) REFERENCES Boats(boat\_id),

FOREIGN KEY (tag\_id) REFERENCES Tags(tag\_id)

);

CREATE TABLE Rentals (

rental\_id INT AUTO\_INCREMENT PRIMARY KEY,

customer\_id INT,

number\_of\_guests INT,

boat\_id INT,

start\_date DATE,

end\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),

FOREIGN KEY (boat\_id) REFERENCES Boats(boat\_id)

);

CREATE TABLE Seasons (

season\_id INT AUTO\_INCREMENT PRIMARY KEY,

season\_info VARCHAR(50) NOT NULL,

start\_month INT NOT NULL,

start\_year INT

);

CREATE TABLE Prices (

price\_id INT AUTO\_INCREMENT PRIMARY KEY,

boat\_class VARCHAR(50) NOT NULL,

season\_id INT NOT NULL,

base\_price DECIMAL(8, 2) NOT NULL,

per\_person\_charge DECIMAL(8, 2) NOT NULL,

FOREIGN KEY (season\_id) REFERENCES Seasons(season\_id)

);

CREATE TABLE Docking\_Stations (

station\_id INT AUTO\_INCREMENT PRIMARY KEY,

station\_name VARCHAR(100) NOT NULL,

location VARCHAR(100) NOT NULL,

max\_number\_of\_boats INT NOT NULL

);

CREATE TABLE Docking\_Records (

docking\_id INT AUTO\_INCREMENT PRIMARY KEY,

station\_id INT,

rental\_id INT,

docking\_start\_date DATETIME NOT NULL,

docking\_end\_date DATETIME NOT NULL,

FOREIGN KEY (station\_id) REFERENCES Docking\_Stations(station\_id),

FOREIGN KEY (rental\_id) REFERENCES Rentals(rental\_id)

);

(c) Write SQL statements for the following questions or updates. If you cannot answer a query using your schema, then you have to modify your solutions in (a) and (b) appropriately.

*(i) Find customers who have rented both class "luxury" and "economy" boats, but have never*

*rented a boat with a "fishing equipment" tag.*

WITH Boat\_Fishing as( select b.boat\_id from Boats b inner join Boat\_Tags bt on b.boat\_id = bt.boat\_id inner join Tags t on t.tag\_id = bt.tag\_id and t.tag\_description = 'Fishing Equipment') SELECT r.customer\_id FROM Rentals r INNER JOIN Boats b ON b.boat\_id = r.boat\_id where b.boat\_class in ('luxury','economy') and b.boat\_id not in (select boat\_id from Boat\_Fishing) group by r.customer\_id having count(distinct b.boat\_class) > 1

A screen shot of a computer

Description automatically generated

*(ii) For each day in August 2023, list any docking stations that were completely full overnight.*

WITH Dates as(

Select DATE\_FORMAT(dr.docking\_start\_date,'%Y-%m-%d') as date\_aug,dr.station\_id , count(\*) as total\_count from Docking\_Records dr

Where dr.docking\_start\_date >= '2023-08-01' and dr.docking\_start\_date <= '2023-08-31'

and datediff(dr.docking\_end\_date,dr.docking\_start\_date) >= 1

Group by date\_aug,dr.station\_id)

SELECT d.\* from Dates d inner join Docking\_Stations ds on d.station\_id = ds.station\_id where d.total\_count = ds.max\_number\_of\_boats;

**Assumption:** Boat can arrive at any time, to change the day of docking the boat should atleast shift from the end points of date (11:59Pm to 12:01Am)

*A computer screen with white text

Description automatically generated*

*(iii) Find all boats that can accommodate at least 6 people and that are available from October 24*

*to November 7, 2023. Output the boat ID, the class, and the total cost of renting it for 6 people*

*during that time.*

Select b.boat\_id,b.boat\_class, (15\*(p.base\_price + 6\* p.per\_person\_charge)) as total\_cost from Boats b inner join Prices p on p.boat\_class = b.boat\_class inner join Seasons si on si.season\_id = p.season\_id where si.start\_month = 10 and si.start\_year = 2023 and b.max\_capacity >=6 and b.boat\_id not in (Select r.boat\_id from rentals r where r.start\_date >= '2023-10-24' and r.start\_date <= '2023-11-07');

**Assumption:** The trip is expected to commence on October 24 and conclude on November 7, encompassing a total duration of 15 days. Additionally, as indicated in the query, the pricing will be determined based on the seasonal rates applicable for the month of October, given that it is the initiation month of the journey.

*A screen shot of a computer program

Description automatically generated*

*(iv) Display the names and IDs of all customers who arrived at the docking place after 10pm in at*

*least two evenings during a single rental.*

WITH CUSTIDS as( Select r.customer\_id from RENTALS r Inner join Docking\_Records dr on r.rental\_id = dr.rental\_id Where DATE\_FORMAT(dr.docking\_start\_date,'%H') >= '22' Group by r.customer\_id having count(dr.docking\_id) >= 2) Select customer\_id,concat(first\_name,' ',last\_name) as name from Customers where customer\_id in (Select customer\_id from CUSTIDS)

*A computer screen with white text

Description automatically generated*

*(v) Given a maximum budget of 2,000 dollars, list all boats that are available for a 10-day rental*

*for 4 guests during August of 2022. You should output the boat ID, the class, the starting date of*

*the rental, and the total cost of the rental.*

SELECT b.boat\_id, (10\*(p.base\_price + 4\*p.per\_person\_charge)) as total\_cost, '2022-08-15' as start\_date from Boats b

inner join Prices p on b.boat\_class = p.boat\_class inner join Seasons si on p.season\_id = si.season\_id where 10\*(p.base\_price + 4\*p.per\_person\_charge) <= 2000 and b.max\_capacity >=4 and si.start\_month = 8 and si.start\_year = 2022

and b.boat\_id not in (select distinct r.boat\_id from Rentals r where r.start\_date BETWEEN '2022-08-15' and '2022-08-24' OR r.end\_date BETWEEN '2022-08-15' and '2022-08-24');

**Assumption:** As per announcements, assuming the start date to be 08-15-2022

*A screen shot of a computer

Description automatically generated*

(d) Create tables in the database system, and insert some sample data (5-10 tuples per table, but choose an interesting and meaningful data set, so that queries do not output empty results). Submit screenshots of what your tables look like after you inserted the data. Then execute the queries in (c) and submit the screenshots of the queries and outputs.

A screenshot of a computer

Description automatically generated

A black screen with white lines

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated