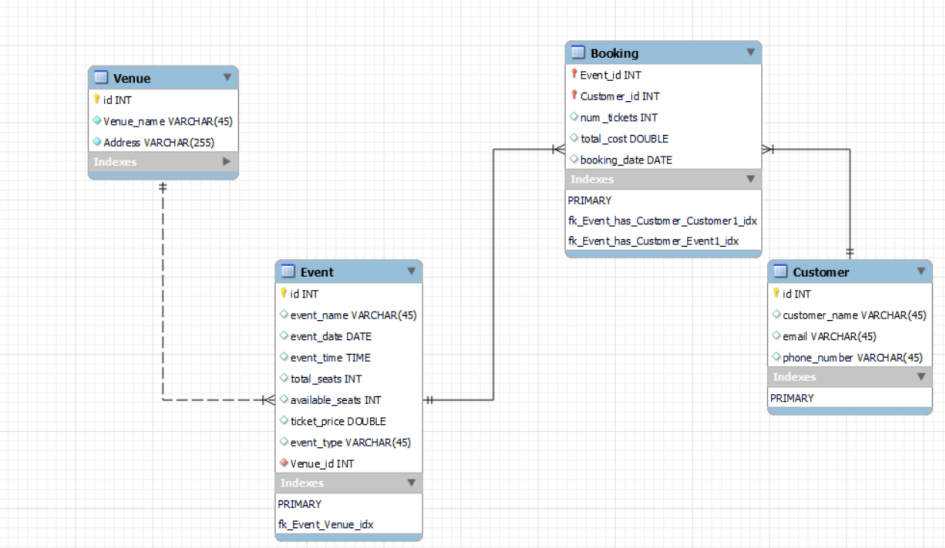
Ticket Booking Assignment

ER Diagram



TASK-1:

-- MySQL Workbench Forward Engineering

-- ---

-- Schema ticketbooking_hex_feb_2024

-- ---

-- ---

-- Schema ticketbooking_hex_feb_2024

-- ---

```
CREATE SCHEMA IF NOT EXISTS `ticketbooking_hex_feb_2024` DEFAULT CHARACTER SET utf8 ;
USE `ticketbooking_hex_feb_2024` ;
```

-- ---

-- Table `ticketbooking_hex_feb_2024`.`Venue`

-- ---

```
CREATE TABLE IF NOT EXISTS `ticketbooking_hex_feb_2024`.`Venue` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `Venue_name` VARCHAR(45) NOT NULL,
  `Address` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`id`))
```

```
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ticketbooking_hex_feb_2024`.`Customer`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ticketbooking_hex_feb_2024`.`Customer` (
  `id` INT NOT NULL,
  `customer_name` VARCHAR(45) NULL,
  `email` VARCHAR(45) NULL,
  `phone_number` VARCHAR(45) NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ticketbooking_hex_feb_2024`.`Event`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ticketbooking_hex_feb_2024`.`Event` (
  `id` INT NOT NULL,
  `event_name` VARCHAR(45) NULL,
  `event_date` DATE NULL,
  `event_time` TIME NULL,
  `total_seats` INT NULL,
  `available_seats` INT NULL,
  `ticket_price` DOUBLE NULL,
  `event_type` VARCHAR(45) NULL,
  `Venue_id` INT NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_Event_Venue_idx` (`Venue_id` ASC),
  CONSTRAINT `fk_Event_Venue`
    FOREIGN KEY (`Venue_id`)
    REFERENCES `ticketbooking_hex_feb_2024`.`Venue` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `ticketbooking_hex_feb_2024`.`Booking`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ticketbooking_hex_feb_2024`.`Booking` (
  `Event_id` INT NOT NULL,
  `Customer_id` INT NOT NULL,
  `num_tickets` INT NULL,
  `total_cost` DOUBLE NULL,
  `booking_date` DATE NULL,
  PRIMARY KEY (`Event_id`, `Customer_id`),
  INDEX `fk_Event_has_Customer_Customer1_idx` (`Customer_id` ASC) ,
  INDEX `fk_Event_has_Customer_Event1_idx` (`Event_id` ASC),
  CONSTRAINT `fk_Event_has_Customer_Event1`
    FOREIGN KEY (`Event_id`)
    REFERENCES `ticketbooking_hex_feb_2024`.`Event` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Event_has_Customer_Customer1`
    FOREIGN KEY (`Customer_id`)
    REFERENCES `ticketbooking_hex_feb_2024`.`Customer` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

TASK-2: Select, Where, Between, AND, LIKE:

1. Write a SQL query to insert at least 10 sample records into each table.

```
insert into venue(venue_name,address) values
('mumbai', 'marol andheri(w)'),
('chennai', 'IT Park'),
('pondicherry ', 'state beach');
```

```
insert into customer(id,customer_name,email,phone_number)
values
(1,'harry potter','harry@gmail.com','45454545'),
(2,'ronald weasley','ron@gmail.com','45454545'),
(3,'hermione granger','her@gmail.com','45454545'),
(4,'draco malfoy','drac@gmail.com','45454545'),
(5,'ginni weasley','ginni@gmail.com','45454545');

insert into
event(id,event_name,event_date,event_time,total_seats,available_seats,ticket_price,event_type,venue_id)
values
(4,'Late Ms. Lata Mangeshkar Musical', '2021-09-12','20:00',320,270,600,'concert',3),
(5,'CSK vs RCB', '2024-04-11','19:30',23000,3,3600,'sports',2),
(6,'CSK vs RR', '2024-04-19','19:30',23000,10,3400,'sports',2),
(7,'MI vs KKR', '2024-05-01','15:30',28000,100,8000,'sports',1);
```

2. Write a SQL query to list all Events.

```
Select * from event;
```

3. Write a SQL query to select events with available tickets.

```
select * from event
where available_seats is not null;
```

4. Write a SQL query to select events name partial match with ‘cup’.

```
select *
from event
where event_name like '%cup%';
```

5. Write a SQL query to select events with ticket price range is between 1000 to 2500.

```
select * from event
where ticket_price> 1000 and ticket_price< 2500;
```

6. Write a SQL query to retrieve events with dates falling within a specific range.

```
select *
from event
```

where event_date BETWEEN '2024-04-11' AND '2024-05-01';

7. Write a SQL query to retrieve events with available tickets that also have "Concert" in their name.

```
select * from event where
ticket_available is not null and event_name like '%concert%';
```

8. Write a SQL query to retrieve users in batches of 5, starting from the 6th user.

```
select *
from customer
limit 5,5;
```

9. Write a SQL query to retrieve bookings details contains booked no of ticket more than 4.

```
select customer_id, count(event_id) as Tickets_booked from booking
group by customer_id
having Tickets_booked>4;
```

10. Write a SQL query to retrieve customer information whose phone number end with '000'

```
select * from customer
where phone_number like'%000';
```

11. Write a SQL query to retrieve the events in order whose seat capacity more than 15000.

```
select * from event
where total_seats>15000
order by total_seats;
```

12. Write a SQL query to select events name not start with 'x', 'y', 'z'

```
select *
from event
where event_name NOT LIKE 'x%' AND event_name NOT LIKE 'y%' AND event_name NOT LIKE 'z%';
```

TASK-3: Aggregate functions, Having, Order By, GroupBy and Joins:

1. Write a SQL query to List Events and Their Average Ticket Prices.

```
select v.venue_name, avg(e.ticket_price) as Average_ticket_price
```

```
from venue v,event e
where v.id=e.venue_id
group by e.venue_id;
```

2. Write a SQL query to Calculate the Total Revenue Generated by Events.

```
select SUM((total_seats  - available_seats) *  ticket_price) from event;
```

3. Write a SQL query to find the event with the highest ticket sales.

```
select event_name,MAX((total_seats  - available_seats) *  ticket_price) as total_sales
from event
group by event_name
order by total_sales desc
limit 0,1;
```

4. Write a SQL query to Calculate the Total Number of Tickets Sold for Each Event.

```
select event_name, total_seats  - available_seats as total_tickets_sold
from event
group by event_name;
```

5. Write a SQL query to Find Events with No Ticket Sales.

```
select event_name
from event
where available_seats=total_seats;
```

6. Write a SQL query to Find the User Who Has Booked the Most Tickets.

```
select c.customer_name,sum(num_tickets) as total_tickets_booked
from customer c,booking b
where c.id=b.customer_id
group by c.id
order by total_tickets_booked desc
limit 0,1;
```

7. Write a SQL query to List Events and the total number of tickets sold for each month.

```
Select
```

```
    month(booking_date) as month,
    year(booking_date) as year,
    e.event_name,
    sum(e.total_seats-e.available_seats) AS total_tickets_sold
from
    events e
join
    booking on e.id = booking.event_id
group by
    year, month, e.event_name
order by
  year, month, e.event_name;
```

8. Write a SQL query to calculate the average Ticket Price for Events in Each Venue.

```
select v.venue_name, avg(e.ticket_price) as Average_ticket_price
from venue v,event e
where v.id=e.venue_id
group by e.venue_id;
```

9. Write a SQL query to calculate the total Number of Tickets Sold for Each Event Type.

```
select event_type,sum(total_seats-available_seats) as Tickets_sold
from event
group by event_type
order by tickets_sold desc;
```

10. Write a SQL query to calculate the total Revenue Generated by Events in Each Year.

```
Select year(booking_date) as year, sum(total_cost) as revenue_generated from booking group by event_id,year;
```

11. Write a SQL query to list users who have booked tickets for multiple events.

```
select c.customer_name, count(e.event_name) as events_booked
from event e,customer c, booking b
where e.id = b.event_id and
b.customer_id = c.id
group by c.customer_name
```

```
having events_booked>1;
```

12. Write a SQL query to calculate the Total Revenue Generated by Events for Each User.

```
select c.customer_name, sum(total_cost) as Revenue_generated
from
event e join booking b on e.id=b.event_id join customer c on b.customer_id=c.id
group by c.customer_name
order by Revenue_generated desc;
```

13. Write a SQL query to calculate the Average Ticket Price for Events in Each Category and Venue.

```
select event_type,venue_id,avg(ticket_price) from event
group by event_type,venue_id;
```

14. Write a SQL query to list Users and the Total Number of Tickets They've Purchased in the Last 30 Days.

```
select c.customer_name, sum(num_tickets) as Total_tickets_purchased
 from customer c join booking b on
 c.id=b.customer_id
 where booking_date between DATE_SUB('2024-04-30',INTERVAL 30 DAY) and  '2024-04-30'
 group by c.id
 order by Total_tickets_purchased desc;
```

Tasks 4: Subquery and its types

1.Calculate the Average Ticket Price for Events in Each Venue Using a Subquery.

```
select venue_id,avg(ticket_price) as Avg_Price
from event
where venue_id IN (select id from venue)
group by venue_id;
```

2. Find Events with More Than 50% of Tickets Sold using subquery.

```
select event_name
from event
where id in ( select id
                    from event
                     where (total_seats - available_seats) > (total_seats/2));
```

3. Calculate the Total Number of Tickets Sold for Each Event.

```
select event_name
from event
where ticket_price >  (select avg(ticket_price) from event);
```

4. Find Users Who Have Not Booked Any Tickets Using a NOT EXISTS Subquery.

```
select * from customer
where not exists (select 1
                  from booking b
                  where b.customer_id = customer.id);
```

5. List Events with No Ticket Sales Using a NOT IN Subquery.

```
select * from event
where id not in (select distinct event_id
                 from booking);
```

7. Find Events with Ticket Prices Higher Than the Average Ticket Price Using a Subquery in the WHERE Clause.

```
select id,event_name from event where
ticket_price > (select
            avg(ticket_price) from event);
```

9. List Users Who Have Booked Tickets for Events in a Given Venue Using a Subquery in the WHERE Clause.

```
select * from customer
where id in(select customer_id from booking
                    where event_id in(select id from event
                                      where venue_id in (select id from venue
                                                         where venue_name='chennai')));
```

10. Calculate the Total Number of Tickets Sold for Each Event Category Using a Subquery with GROUP BY.

```
select event_type, sum(b.num_tickets)as total_tickets_booked from event e,booking b where b.event_id=e.id
group by event_type;
```

11. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with DATE_FORMAT.

```
select * from customer
where id in(select customer_id
        from booking where event_id in(select id
                                        from event where month(event_date)=5));
```

12. Calculate the Average Ticket Price for Events in Each Venue Using a Subquery

```
select id,venue_name,(select avg(ticket_price)
                        from event where venue.id=event.venue_id) as Avg_ticket_price from venue;
```