



-- MySQL Script generated by MySQL Workbench

-- Tue Mar 5 08:48:10 2024

-- Model: New Model Version: 1.0

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE,

SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-----

-- Schema TicketBookingSystem

-----

-----

-- Schema TicketBookingSystem

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

```
-----  
-- Table `TicketBookingSystem`.`venue`  
-----
```

```
CREATE TABLE IF NOT EXISTS `TicketBookingSystem`.`venue` (  
  `venue_id` INT NOT NULL AUTO_INCREMENT,  
  `venue_name` VARCHAR(45) NOT NULL,  
  `venue_address` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`venue_id`))  
ENGINE = InnoDB;
```

```
-----  
-- Table `TicketBookingSystem`.`event`  
-----
```

```
CREATE TABLE IF NOT EXISTS `TicketBookingSystem`.`event` (  
  `event_id` INT NOT NULL AUTO_INCREMENT,  
  `event_name` VARCHAR(255) NOT NULL,  
  `event_date` DATE NOT NULL,  
  `event_time` TIME NOT NULL,  
  `venue_id` INT NULL,  
  `total_seats` INT NOT NULL,  
  `available_seats` INT NOT NULL,  
  `ticket_price` DOUBLE NOT NULL,  
  `event_type` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`event_id`),  
  INDEX `venue_id_idx` (`venue_id` ASC) ,  
  CONSTRAINT `venue_id`
```

```
FOREIGN KEY (`venue_id`)
REFERENCES `TicketBookingSystem`.`venue` (`venue_id`)
ON DELETE NO ACTION
ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----
-- Table `TicketBookingSystem`.`customer`
-- -----
```

```
CREATE TABLE IF NOT EXISTS `TicketBookingSystem`.`customer` (
  `customer_id` INT NOT NULL AUTO_INCREMENT,
  `customer_name` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `phone_number` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`customer_id`))
ENGINE = InnoDB;
```

```
-- -----
-- Table `TicketBookingSystem`.`booking`
-- -----
```

```
CREATE TABLE IF NOT EXISTS `TicketBookingSystem`.`booking` (
  `booking_id` INT NOT NULL AUTO_INCREMENT,
  `customer_id` INT NULL,
  `event_id` INT NULL,
  `num_tickets` INT NOT NULL,
  `total_cost` DOUBLE NOT NULL,
  `booking_date` DATE NOT NULL,
  PRIMARY KEY (`booking_id`),
  INDEX `event_id_idx` (`event_id` ASC),
```

```
INDEX `customer_id_idx` (`customer_id` ASC),  
CONSTRAINT `event_id`  
  FOREIGN KEY (`event_id`)  
  REFERENCES `TicketBookingSystem`.`event` (`event_id`)  
  ON DELETE NO ACTION  
  ON UPDATE NO ACTION,  
CONSTRAINT `customer_id`  
  FOREIGN KEY (`customer_id`)  
  REFERENCES `TicketBookingSystem`.`customer` (`customer_id`)  
  ON DELETE NO ACTION  
  ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;  
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;  
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

```
show tables;
```

```
insert into customer(customer_name,email,phone_number)  
values('ram','ram@gmail.com','9988776655'),  
('bhim','bhim@gmail.com','9988776655'),  
('krishna','krishna@gmail.com','9988776655'),  
('venky','venky@gmail.com','9988776655'),  
('govind','govind@gmail.com','9988776655'),  
('hare','hare@gmail.com','7788996655');
```

```
select * from customer;
```

```
insert into venue(venue_name,venue_address)
```

```
values('chennai','it park'),  
( 'mumbai','beach road'),  
( 'chennai','beach road'),  
( 'hyderabad','it park');
```

```
select * from venue;
```

```
insert into  
event(event_name,event_date,event_time,venue_id,total_seats,available_seats,ticket_price,event_t  
ype)
```

```
values
```

```
( 'CSK vs RR','2024-03-30','07:00',1,24000,50,3000,'sports'),  
( 'music concert','2024-04-22','01:00',3,25000,5,4000,'sports'),  
( 'MI vs RCB','2024-04-10','07:00',2,26000,50,6000,'sports');
```

```
insert into  
event(event_name,event_date,event_time,venue_id,total_seats,available_seats,ticket_price,event_t  
ype)
```

```
values('CSK vs RCB','2024-03-30','07:00',1,24000,50,7000,'sports');
```

```
insert into  
event(event_name,event_date,event_time,venue_id,total_seats,available_seats,ticket_price,event_t  
ype)
```

```
values
```

```
( 'RCB vs RR','2024-04-10','07:00',1,27000,50,7000,'sports');
```

```
select * from event;
```

```
insert into booking(customer_id,event_id,num_tickets,total_cost,booking_date)
```

```
values(2,1,3,12000,'2024-03-01'),
```

```
(1,1,2,8000,'2024-03-01'),
```

```
(2,4,5,30000,'2024-03-15'),
```

```
(3,3,3,12000,'2024-03-01');
```

```
select * from booking;
```

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

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

-- done above

-- 2. Write a SQL query to list all Events.

```
select * from event;
```

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

```
select event_name,available_seats from event where available_seats>0;
```

-- 4. Write a SQL query to select events name partial match with 'cup'.

```
select event_name 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 between 1000 and 2500;
```

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

```
select * from event where event_date between '2024-03-01' and '2024-03-30';
```

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

```
select event_name,available_seats from event where available_seats>0 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 * from booking where num_tickets > 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 desc;
```

-- 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%";
```

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

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

```
select v.venue_id,v.venue_name,v.venue_address,avg(e.ticket_price) as average_ticket_price  
from venue v inner join event e  
on v.venue_id=e.venue_id  
group by v.venue_id;
```

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

```
select sum((total_seats-available_seats)*ticket_price) as total_revenue from event;
```

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

```
select event_name from event order by (total_seats-available_seats) 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 seats_sold from event;
```

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

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

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

```
select c.customer_name,sum(b.num_tickets) as booked_tickets
from booking b inner join customer c
on b.customer_id=c.customer_id
group by b.customer_id
order by booked_tickets desc limit 0,1;
```

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

```
select b.event_id,e.event_name,month(b.booking_date),sum(b.num_tickets)
from booking b inner join event e
on e.event_id=b.event_id
group by b.event_id,month(b.booking_date);
```



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

```
select v.venue_id,v.venue_name,avg(e.ticket_price)
from venue v inner join event e
on v.venue_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) from event group by event_type;
```

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

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

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

```
select c.customer_name
from customer c join booking b
on c.customer_id=b.customer_id
group by b.customer_id
having count(b.event_id)>1;
```

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

```
select b.customer_id,c.customer_name ,sum(b.total_cost) as total_revenue
from customer c inner join booking b
on c.customer_id=b.customer_id
group by b.customer_id;
```

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

```
select v.venue_name,avg(e.ticket_price) ,e.event_type
from venue v inner join event e
on v.venue_id=e.venue_id
group by e.venue_id,e.event_type;
```

-- 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(b.num_tickets) as Number_Of_tickets
from booking b JOIN customer c ON c.customer_id = b.customer_id
where b.booking_date between DATE_SUB('2024-03-30',INTERVAL 30 DAY) and '2024-04-30'
group by c.customer_id;
```

-- 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) from event where venue_id in (select venue_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 available_seats*2<total_seats;
```

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

```
select event_name,total_seats-available_seats as seats_sold from event;
```

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

```
select customer_id,customer_name from customer where customer_id not in (select customer_id
from booking);
```

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

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

-- 6. Calculate the Total Number of Tickets Sold for Each Event Type Using a Subquery in the FROM Clause.

```
select event_type,sum(total_seats-available_seats) from event group by event_type;
```

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

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

-- 8. Calculate the Total Revenue Generated by Events for Each User Using a Correlated Subquery.

```
select customer_id,sum(total_cost) as total_revenue
from booking where customer_id in
(select customer_id from customer) group by customer_id;
```

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

```
select customer_id
from booking where event_id in
(select event_id from event where venue_id in
(select venue_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_id,sum(num_tickets) as total_tickets
from booking
where event_id in (select event_id from event)
group by event_id;
```

-- 11. Find Users Who Have Booked Tickets for Events in each Month Using a Subquery with DATE\_FORMAT.

```
select * from customer
where customer_id in (select customer_id from booking);
```

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

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