

Assessment 1: Practical coding assignments using SQL.

Data Analytics Report for Qatar 2022 World Cup

29/09/2023

Student: Huynh Quang Minh NGUYEN

Student ID: 46863885

Lecturer: Geng Sun

Class: Friday 5 pm - 8 pm

Unit: Data and Visualization for Business (BUSA8090)

1. How the database was created

Database Creation Process

The development of the Qatar 2022 Football World Cup ticketing system database involved a systematic approach to ensure the efficient management of ticket sales, transportation, and fan accommodations throughout the World Cup period. There are 2 main parts in creating the completed database:

a) Part 1: Creating Data Model

Step 1: Identify Entities:

Initial planning involved understanding the essential entities, their attributes, and the relationships between them. In this step, firstly, I start to identify the main entities, which I consider as a “Noun”.

Step 2: Identify business rules:

I began to identify business rules, which is a “Noun-verb-Noun” relationship in the narrative. Having defined the relationship between each entity followed by “Noun – verb – Noun” rules.

Step 3: Define relationships and represent cardinality:

I then represent the cardinality for each relationship, this can be: one to one Relationship, one to many Relationships. These two relationships can be considered as Binary relationships or Unary relationships. The main difference between these two relationships is the entity involved in the process, in which Binary will have two distinct entities whereas Unary will have a relationship between two different instances of an entity.

However, in some cases, some business objects are relationships, therefore, we will have an associative entity (Many to Many Relationships).

Step 4: Identify attributes:

The following step is to identify the attributes within the entity, representing the information that I want to store about entities. However, before choosing which attributes should be put into the entity, I have started to define the main audience of this database use for, which is the Chief Operating Officer (COO) of the ticket-selling system and his/her main concern is to: minimizing the cost of operations on managing the “transportation system” and “fan camps” for the ticketholder.

In this part, to visualize the relationships and lay the foundation for the database schema, the ER Diagram is crucial to help the audience more easily understand the database. I use Chen’s Notation Diagram to visualize my database as the picture below:

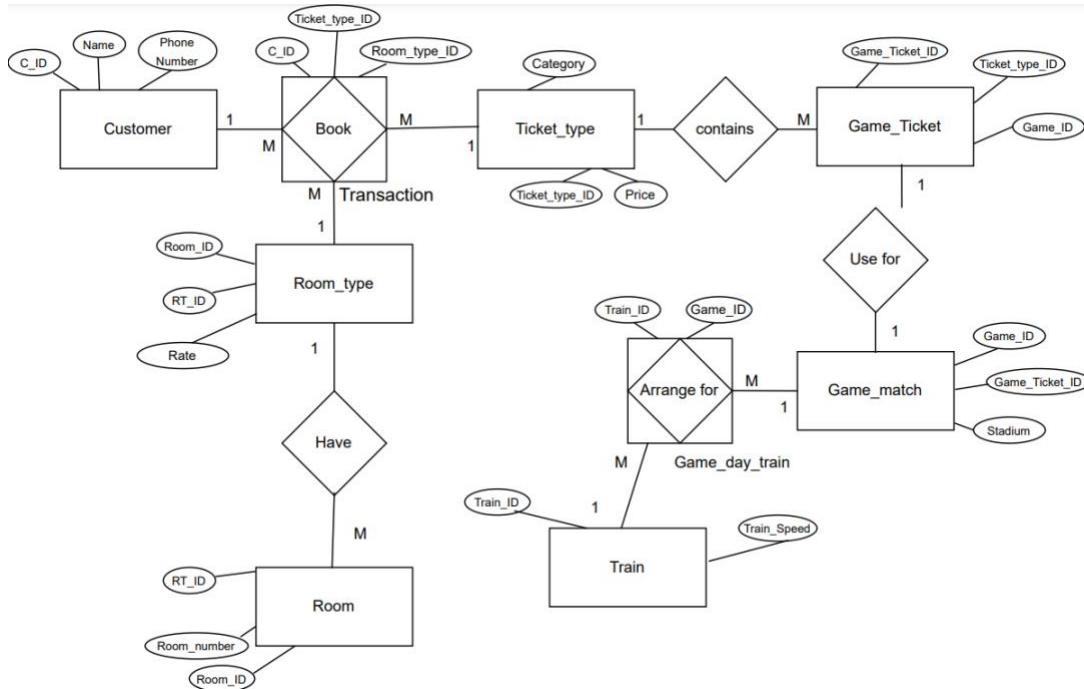


Figure 1: Chen's Notation Diagram

b) Part 2: Transforming a Data Model into a Database design.

In this part, each entity identified in Chen's Notation Diagram was translated into a table in the database schema with the attributes carefully chosen to capture relevant information for each entity to solve the COO's concern. Transforming a data model into a database design involves a systematic process to represent entities, relationships, and cardinalities in a structured and efficient manner.

Step 1: Create a table for each entity.

First, appropriate data types and constraints need to be assigned to each attribute to maintain data integrity and consistency. I use the "Create table" Code to create a table for every entity I have in Chen's Notation Diagram. Each table will represent a distinct entity and will contain attributes that characterize that entity in which attributes have been carefully selected and defined.

Step 2: Create the relationship by placing foreign keys.

In this step, relationships between tables were established based on foreign keys, ensuring referential integrity. A foreign key in a table refers to the primary key of another table, establishing a connection between the two entities. The foreign key represents the relationship and ensures referential integrity. I use the delete/update/alter actions set to maintain data integrity.

Step 3: Specify the logic for enforcing cardinality.

Cardinality defines the number of occurrences of one entity for a single occurrence in another entity within a relationship. Specify cardinality by considering the business rules and requirements. Common cardinalities include one-to-one, one-to-many, and many-to-many. Utilize constraints and logical rules to enforce the defined cardinalities within the database schema.

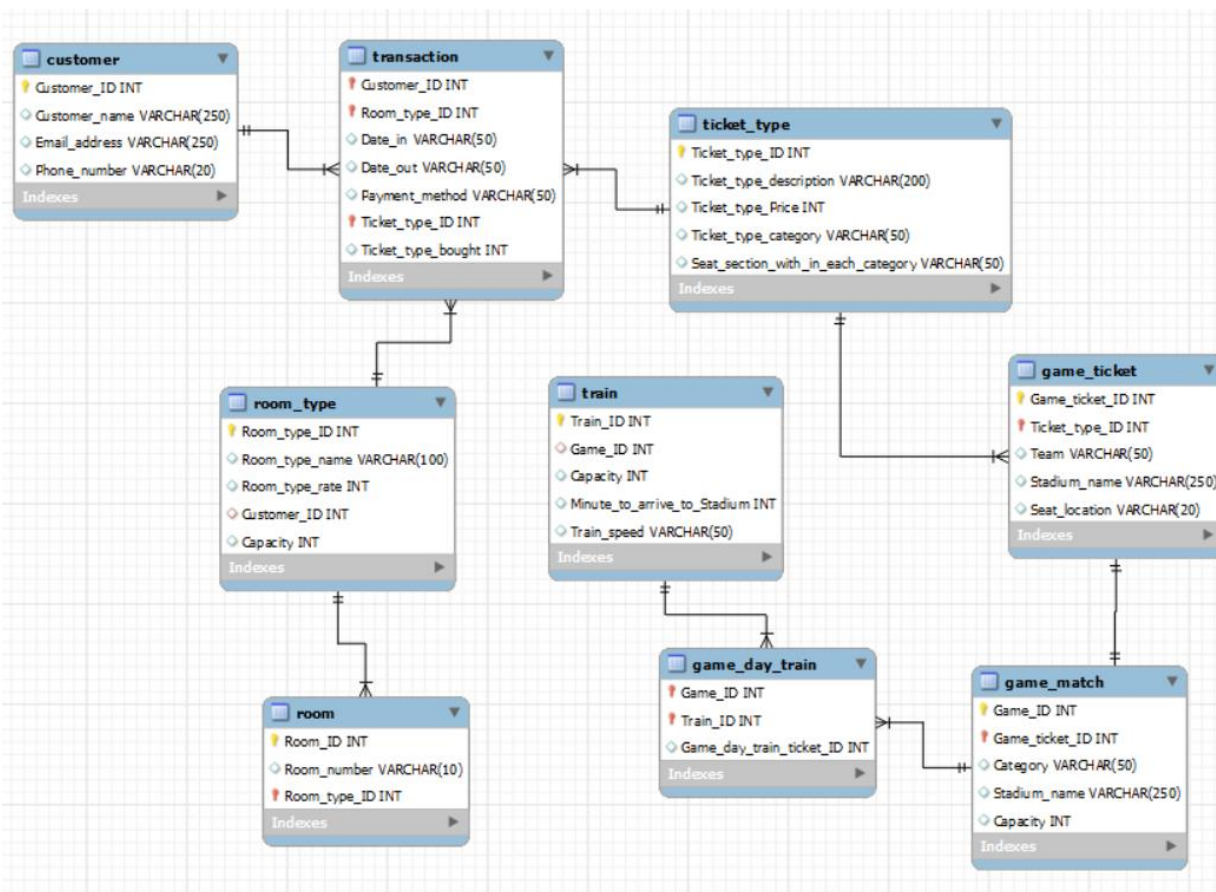


Figure 2: ER Diagram created by MySQL Workbench

This step-by-step process ensures a smooth transformation of the data model into a structured database design, capturing the relationships and cardinalities essential for effective data representation and management.

2. The elaborations of how the queries were selected and formulated, and suggestions of the return results.

In the realm of the ticket-selling system, the Chief Operating Officer (COO) of the ticket-selling system faces a multitude of organizational challenges that directly influence strategic decision-making on how to minimize the cost of managing the transportation and fans camp. To help the COO make better decisions through the data generated, firstly, we need to define the potential problems that can impact and raise the cost of operations, leading to the ticket-selling system becoming less and less effective and efficient. There are some potential problems as:

1. **Capacity Optimization in Transportation:** Insufficient data or inaccurate forecasting leads to either overestimation or underestimation of transportation needs. Issue: Balancing the capacity of trains and other transportation modes to match ticket sales, preventing underutilization, or overcrowding.

2. **Ticket Holder Priority:** Ambiguities in defining criteria for ticket holder priority, potentially leading to customer dissatisfaction or unequal treatment. Ticket holders may have varying levels of priority in

terms of accommodation, transportation, or other event-related services, and lack of clear guidelines can lead to confusion and discontent.

3. Integration of Systems: Issues in integrating various information systems and databases that manage transportation, fan camp bookings, and ticket sales, impacting the efficiency of operations. Solution: Utilize an integrated ticketing platform that consolidates sales channels, enabling customers to access tickets seamlessly and efficiently across channels.

4. Number of Tickets Sold: Inadequate Sales Tracking: Insufficient mechanisms to accurately track and monitor ticket sales, hindering the ability to evaluate sales performance and plan accordingly.

5. Room Management: Room Overbooking and Underutilization: Difficulty in accurately estimating the number of rooms needed, resulting in either overbooking or underutilization, impacting operational efficiency and customer satisfaction.

These organizational issues underscore the necessity for a well-structured and integrated information system, robust data governance, and a strategic approach to optimize operations and enhance the overall event experience. To solve these issues, based on the database that has been built, we will construct the SQL Query to extract data to interpret the result and give the recommendation to help the COO enhance the whole operation's performance.

SQL Query to solve these above organizational issues

1. Capacity Optimization in Transportation

1.1. Query to calculate and analyze the ticket sales for each train in the context of the Qatar 2022 World Cup

```
47 • SELECT
48     T.Train_ID,
49     T.Capacity AS Train_Capacity,
50     COUNT(GDT.Game_day_train_ticket_ID) AS Game_Day_Train_Ticket,
51     (T.Capacity - COUNT(GDT.Game_day_train_ticket_ID)) AS Remaining_Capacity
52 FROM
53     Train T
54 LEFT JOIN
55     Game_day_train GDT ON T.Train_ID = GDT.Train_ID
56 GROUP BY
57     T.Train_ID, T.Capacity
58 HAVING
59     (T.Capacity - COUNT(GDT.Game_day_train_ticket_ID)) >= 0;
```

SELECT

Clause: The SELECT clause defines the columns to be included in the result set. Columns Selected:

+ T.Train_ID: This is the ID of the train.

+ T.Capacity AS Train_Capacity: It represents the maximum capacity of the train.

+ COUNT (GDT.Game_day_train_ticket_ID) AS Game_Day_Train_Ticket: This calculates the number of game day train tickets complement for each game ticket and name as Game_Day_Train_Ticket

+ (T.Capacity - COUNT(GDT.Game_day_train_ticket_ID)) AS Remaining_Capacity: This calculates the remaining capacity of each train.

- FROM Clause: The FROM clause specifies the tables involved in the query. Table Used:

+ Train T: This is the main table representing the trains.

- JOIN Clause: LEFT JOIN: This is a left join between the Train table and the Game_day_train table based on the Train_ID.

- GROUP BY Clause: The GROUP BY clause is used to arrange identical data into groups in which columns for Grouping as

+ T.Train_ID and T.Capacity will be the results are grouped based on train ID and train capacity.

- HAVING Clause: The HAVING clause filters the groups after the GROUP BY phase with the condition: (T.Capacity - COUNT(GDT.Game_day_train_ticket_ID)) >= 0. This is to ensure that we only include groups where the remaining capacity is greater than or equal to zero, meaning the train is not overbooked.

Elaborations:

The query calculates the remaining capacity for each train by subtracting the number of game day train tickets sold from the maximum train capacity in which it uses a LEFT JOIN to include all trains, even if they have not sold any tickets. The GROUP BY clause is used to group the results by train, and the HAVING clause filters out trains that have exceeded their capacity.

Suggestions for Return Results:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
Train_ID	Train_Capacity	Game_Day_Train_Ticket	Remaining_Capacity
1	1	1	0
2	2	1	1
3	3	1	2
4	1	1	0
5	2	1	1
6	3	1	2

The return results will include the Train_ID, maximum capacity (Train_Capacity), the number of game day train tickets sold (Game_Day_Train_Ticket), and the remaining capacity of each train (Remaining_Capacity).

These results provide valuable insights into train occupancy, helping in optimizing transportation logistics for the Qatar 2022 Football World Cup. The COO can use this information to manage train schedules, allocate resources efficiently, and ensure a smooth transportation experience for fans attending the event.

1.2. Query to know the average capacity of the train.

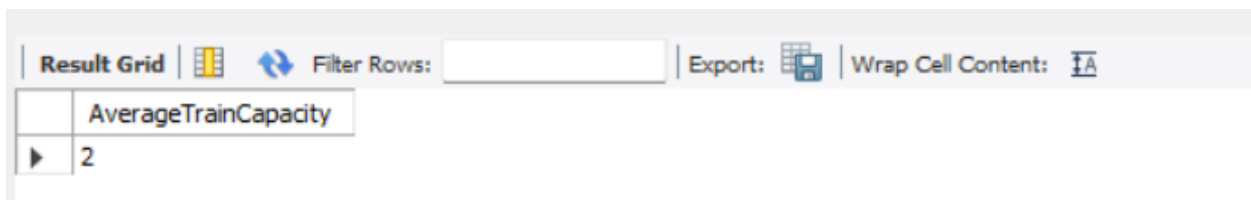
```
-- Query to know the average capacity of train
• SELECT Round(AVG(MaxTrainCapacity)) AS AverageTrainCapacity
FROM (
    SELECT Train_ID, MAX(Capacity) AS MaxTrainCapacity
    FROM Train
    GROUP BY Train_ID
) AS MaxCapacities;
```

- SELECT Clause: The SELECT clause specifies the result to be returned: the average train capacity.
- Subquery: The subquery calculates the maximum capacity for each train and aliases it as MaxTrainCapacity.
- Aggregate Function Average (AVG): The AVG function calculates the average of the maximum train capacities.
- GROUP BY Clause: The GROUP BY clause is used to arrange identical data into group

Elaborations:

The query calculates the average capacity of trains based on the maximum capacity for each train.

Suggested Return Results:



AverageTrainCapacity
2

The return result will be a single value representing the average train capacity. If we execute this query, we will get the average train capacity based on the data in the Train table. This information can help the COO have a general picture of the train's capacity, helping the COO to make better decision on how many train need for each game match.

2. Ticket Holder Priority

2.1. Query to know how many ticketholders.

```
-- Query to know how many ticketholder
SELECT COUNT(DISTINCT Customer_ID) AS NumberOfTicketHolders
FROM transaction
WHERE Ticket_type_bought > 0;
```

SELECT Clause: The SELECT clause specifies the result to be returned: the sum of Ticket_type_bought as TotalTicketsSold.

- Distinct Clause help to remove duplicate values from Customer_ID.

- FROM Clause: The FROM clause specifies the table involved in the query: transaction.
- Aggregate Function (SUM): SUM(Ticket_type_bought) calculates the total sum of the Ticket_type_bought column.

Elaborations:

The query calculates the total number of ticketholders by summing up the values in the Ticket_type_bought column.

Suggested Return Results:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
NumberOfTicketHolders			
59			

The return result will be a single value, representing the total number of ticket holders, helping the COO know how many people will be attending each match, and the need to book FanCamp Rooms while using train transportation to travel to the Stadium.

2.2. Query to know how many people have tickets as a customer.

```
-- Query to know how many people have ticket as a customer
SELECT 'Ticket Holders' AS Category, COUNT(Customer_ID) AS TotalCount
FROM transaction
WHERE Ticket_type_bought > 0
UNION
SELECT 'Non-Ticket Holders' AS Category, COUNT(Customer_ID) AS TotalCount
FROM transaction
WHERE Ticket_type_bought = 0;
```

- SELECT Clause: The SELECT clause specifies the result to be returned: 'Ticket Holders' and 'Non-Ticket Holders' as categories and the respective counts.

- UNION Clause: The UNION operator is used to combine the results of two SELECT statements.

- WHERE Clause: The WHERE clause filters the records based on the condition specified: Ticket_type_bought > 0 for Ticket Holders and Ticket_type_bought = 0 for Non-Ticket Holders.

Elaborations:

The query calculates the count of customers who are ticket holders and non-ticket holders, helping to determine who has priority when booking the FanCamp Rooms, including the duplicate values.

Suggested Return Results:

Result Grid			Filter Rows:	Export:	Wrap Cell Content:
	Category	TotalCount			
▶	Ticket Holders	64			
	Non-Ticket Holders	0			

The return result will be two rows, one indicating the count of ticket holders and the other indicating the count of non-ticket holders. If we execute this query, we will get a result showing the count of customers who are ticket holders and non-ticket holders based on the data in the transaction table. This query will help the decision-makers know the conversion ratio of changing customers into ticketholders. If the conversion ratio is low, the COO needs to look back at the selling system to identify whether they are not convenient for customer to buy a ticket or if there is a problem with the marketing department as not many customers know about the ticket_selling_system.

3. Integration of Systems

3.1. Query to know how to solve issues in integrating various information systems and databases that manage transportation, fan camp bookings, and ticket sales, impacting the efficiency of operations.

```
-- Query to know to solve issues in integrating various information systems and databases
SELECT *
FROM transaction T
JOIN Game_ticket GT ON T.Ticket_type_ID = GT.Ticket_type_ID;
```

- SELECT Clause: The SELECT * clause indicates that I want to select all columns from the resulting dataset.
- FROM Clause: The FROM clause specifies the tables involved in the query. In this case will be:
 - + Transaction T: This table represents transactions.
 - + Game_ticket GT: This table represents game tickets.
- JOIN Clause: This is a join between the transaction table and the Game_ticket table based on the Ticket_type_ID.

Elaborations:

The query is performing join between transaction and Game_ticket based on the common attribute Ticket_type_ID. This join will combine records from both tables where the Ticket_type_ID matches.

Suggested Return Results:

Result Grid												Filter Rows:	Export:	Wrap Cell Contents:
Customer_ID	Room_type_ID	Date_in	Date_out	Payment_method	Ticket_type_ID	Ticket_type_bought	Game_ticket_ID	Ticket_type_ID	Team	Stadium_name	Seat_location			
10012	142501	20-Nov-22	21-Nov-22	Visa	14565132	2	13462	14565132	QATAR vs ECUADOR	Lusail Stadium	4C			
10012	142501	20-Nov-22	21-Nov-22	Visa	14565132	2	133462	14565132	SWITZERLAND vs CAMEROON	Khalifa International Stadium	27U			
10012	142501	20-Nov-22	21-Nov-22	Visa	14565132	2	293462	14565132	UNITED STATES vs WALES	Al Bayt Stadium	31A			
10012	142501	20-Nov-22	21-Nov-22	Visa	14565132	2	413462	14565132	ARGENTINA vs MEXICO	Al Bayt Stadium	21C			
10012	142501	20-Nov-22	21-Nov-22	Visa	14565132	2	573462	14565132	Winner A vs Winner B	Education City Stadium	52D			

The return results will include all columns from both Transaction and Game_ticket for rows where the Ticket_type_ID matches in both tables. The combined results provide comprehensive information about transactions, including details about the associated game tickets. This can be useful for analyzing ticket

sales and related game ticket information, helping the COO to identify the pattern in customer's purchasing behavior. Therefore, it will be useful for making further predictions for the future.

3.2. Query to know the total number of games.

```
-- Query to known total number of games
• SELECT COUNT(*) AS TotalGames
  FROM Game_match;
```

- SELECT Clause: The SELECT clause specifies the result to be returned: the total number of games.

- Aggregate Function (COUNT): The COUNT function is an aggregate function that counts the number of rows.

Elaborations:

The query counts the total number of games available in the Game_match table.

Suggested Return Results:

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
TotalGames			
64			

The return result will be a single value representing the total number of games. If we execute this query, we can get the total number of games based on the data in the Game_match table, together with knowing the total ticket_sold, the COO can estimate the average number of customers for each game and, therefore, make a better decision to arrange the train for each game held.

4. Number of Tickets Sold

4.1. Query to estimate the total tickets sold for each game

```
---- Query to estimate total ticketsold for each game:
SELECT Team, COUNT(*) AS TicketCount
FROM Game_ticket
GROUP BY Team
Having Count(*) >2
Order by TicketCount asc;
```

- SELECT Clause: The SELECT clause specifies the columns to be included in the result set: Team and the count of tickets for each team as TicketCount.

- FROM Clause: the FROM clause specifies the table involved in the query: Game_ticket.
- GROUP BY Clause: set by the Team column.
- HAVING Clause: HAVING Count(*) > 2 filters the grouped results, showing only those teams with more than 2 tickets sold.
- ORDER BY Clause: ORDER BY TicketCount ASC orders the result set by TicketCount in ascending order.

Elaborations:

The query counts the number of tickets sold (TicketCount) for each team (Team). It filters out teams with fewer than 2 ticket sales (using HAVING). It presents the results in ascending order of ticket count.

Suggested Return Results:

Result Grid   Filter Rows: <input type="text"/>	
Team	TicketCount
▶ ENGLAND vs IRAN	4
QATAR vs ECUADOR	5
SENEGAL vs NETHERLANDS	5
SPAIN vs GERMANY	5
GHANA vs URUGUAY	5

The return results will include each team and the count of tickets sold for teams that have more than 2 ticket sales. This information is valuable for identifying popular teams or matches based on ticket sales, helping the COO know which type of team this season customers will pay more attention on, and which types of matches will receive more attention in comparison with other matches. Based on that, the COO can arrange suitable resources for each match held as a number of trains, the number of employees for each match, and the number of rooms for the ticketholder.

4.2. Query to know how many tickets sold at 1 stadium.

```
-- Query to know how many ticket_sold at 1 stadium
SELECT Stadium_name, COUNT(*) AS TicketCount
FROM Game_ticket
WHERE Stadium_name = 'Lusail Stadium' OR Stadium_name = 'Ai Bayt Stadium'
GROUP BY Stadium_name;
```

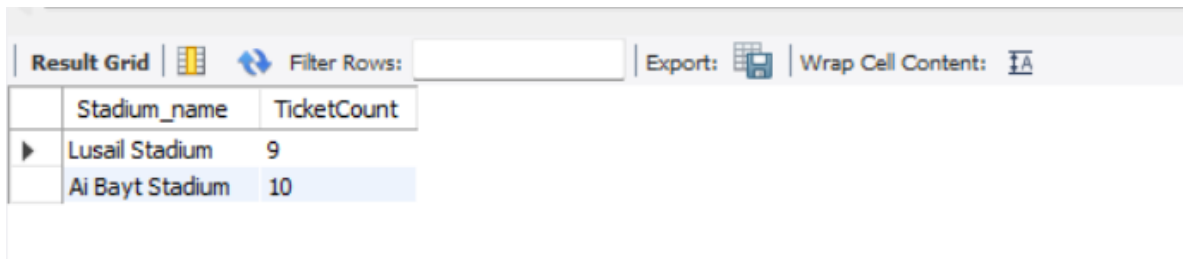
- SELECT Clause: The SELECT clause specifies the columns to be included in the result set: Stadium_name and the count of tickets for each stadium as TicketCount.
- FROM Clause: The FROM clause specifies the table involved in the query: Game_ticket.
- WHERE Clause: WHERE Stadium_name = 'Lusail Stadium' OR Stadium_name = 'Ai Bayt Stadium' filters the results to include only those rows where the stadium name is either 'Lusail Stadium' or 'Ai Bayt Stadium'.

- GROUP BY Clause: GROUP BY Stadium_name groups the result set by the Stadium_name column.

Elaborations:

The query counts the number of tickets sold (TicketCount) for each specified stadium (Stadium_name). It filters the results to include only tickets from 'Lusail Stadium' or 'Ai Bayt Stadium' and presents the results grouped by stadium.

Suggested Return Results:



	Stadium_name	TicketCount
▶	Lusail Stadium	9
	Ai Bayt Stadium	10

The return results will include the number of tickets sold for each specified stadium: 'Lusail Stadium' and 'Ai Bayt Stadium'. This information helps to understand ticket sales for specific stadiums. If the ticket sales for one stadium are much lower than other stadiums in contrast, then, there would be a problem behind that. This could come from inconvenient transportation, lack of infrastructure, availability of rooms for stay, etc. so that the COO could identify the main problem's root and, as a result, make better decisions to hold a successful World Cup Period.

4.3. Query to know the total number of tickets sold.

```
-- Query to know total number of ticket_sold
SELECT SUM(Ticket_type_bought) AS TotalTicketsSold
FROM transaction;
```

SELECT Clause: The SELECT clause specifies the result to be returned: the sum of Ticket_type_bought as TotalTicketsSold.

- FROM Clause: The FROM clause specifies the table involved in the query: transaction.

- Aggregate Function (SUM): SUM(Ticket_type_bought) calculates the total sum of the Ticket_type_bought column.

Elaborations:

The query calculates the total number of tickets sold by summing up the values in the Ticket_type_bought column.

Suggested Return Results:

Result Grid			Filter Rows: <input type="text"/>	Export:	Wrap Cell Content:
	TotalTicketsSold				
▶	207				

The return result will be a single value, representing the total number of tickets sold. If we execute this query, we will get a result showing the total number of tickets sold based on the data in the transaction table. Based on this information and combined with several ticketholders, the COO can know the average ticket held by 1 person. Then the COO can calculate the conversion rate of customers into ticketholders, helping to make a predictive analysis for the future and be well-prepared for the situation to come from the future.

5. Room Management

5.1. Query to get details about room bookings and the respective customers to address the issue of room overbooking and underutilization.

```
SELECT
    room.Room_number,
    room_type.Room_type_name,
    room_type.Room_type_rate,
    customer.Customer_name,
    room_type.Capacity,
    room_type.Capacity - COUNT(room_type.Room_type_ID) AS RemainingCapacity
FROM
    room
JOIN
    Room_type ON room.Room_type_ID = room_type.Room_type_ID
LEFT JOIN
    transaction ON room_type.Room_type_ID = transaction.Room_type_ID
Left JOIN
    Customer ON transaction.Customer_ID =customer.Customer_ID
GROUP BY
    room.Room_number, room_type.Room_type_name, room_type.Room_type_rate, customer.Customer_name, room_type.Capacity;
```

- SELECT Clause: The SELECT clause specifies the columns to be included in the result set.

- FROM Clause: The FROM clause specifies the tables involved in the query, in which:

+ Room: This table represents rooms.

+ Room_type: This table represents room types.

+ Transaction: This table represents transactions.

+ Customer: This table represents customers.

- JOIN Clauses:

+ JOIN Room_type ON room.Room_type_ID = room_type.Room_type_ID: This is a join between the room table and the Room_type table based on the Room_type_ID.

+ LEFT JOIN transaction ON room_type.Room_type_ID = transaction.Room_type_ID: This is a left join between the Room_type table and the transaction table based on Room_type_ID.

+ LEFT JOIN Customer ON transaction.Customer_ID = customer.Customer_ID: This is a left join between the transaction table and the Customer table based on Customer_ID.


- GROUP BY Clause: GROUP BY room.Room_number, room_type.Room_type_name, room_type.Room_type_rate, customer.Customer_name, room_type.Capacity: This groups the result set by the specified columns.


Elaborations:


The query combines information from the room, Room_type, transaction, and Customer tables. It calculates the remaining capacity in each room type based on the total capacity and the count of associated transactions while grouping the results by room number, room type name, room type rate, customer name, and room type capacity.

Suggested Return Results:

Result Grid

 Filter Rows:

 Export:

 Wrap Cell Content:

	Room_number	Room_type_name	Room_type_rate	Customer_name	Capacity	RemainingCapacity
▶	101	Single	100	Joe	1	0
	102	Single	100	Ben	1	0
	103	Single	100	Zumara	1	0
	104	Double	160	Chris	2	1
	105	Share	200	Amelia	4	3

The return results will include columns from the specified tables, showing room number, room type details, customer names, room type capacity, and remaining capacity for each room type. This can be useful for room management and capacity analysis, helping the COO know how many numbers of room is still available if all the ticketholder has registered for booking so the FanCamp Room can let another visitor book for the room to bring additional revenues and in contrast.

5.2. Query to know how many people a ticketholder is and just hold 1 ticket.

```
-- Query to know how many people is a ticketholder and just hold 1 ticket
SELECT COUNT(Distinct Customer_ID) AS NumberOfCustomersWithOneTicket
FROM transaction
WHERE Ticket_type_bought = 1;
```

- SELECT Clause: The SELECT clause specifies the result to be returned: the total number of games.

- Aggregate Function (COUNT): The COUNT function is an aggregate function that counts the number of rows.

- Where Clause: indicate the conditions under which a row will be included in the result.

Elaborations:

The query counts the total number of games available in the Game_match table.

Suggested Return Results:



Result Grid			Filter Rows: <input type="text"/>	Export: 	Wrap Cell Content: 
	NumberOfCustomersWithOneTicket				
▶	17				

The return result will be a single value representing the total number of games based on the data in the Game_match table. This information can be utilized to predict the room_type that ticketholder will look for. For example, if the ticketholders have bought 1 game ticket, we can assume that they are single and want to look for a single room type, helping the COO to think of which type of room should be prioritized to build most and least.