ER DESIGN F1 COMPETITION

(Table-Ticket Schedule simulations)

Vlad Bogdantsev
https://github.com/vladproduction

Task: ER Design F1 competition (Table or Ticket Schedule simulations);

Link for this project: https://github.com/vladproduction/MySql-projects/tree/main/formula_one

Requirements:

- to sketch out the design of a database;

- defining the entities, their attributes and showing their relationships;

- give a high-level overview of functions and use cases are relevant for the ER Design F1 competition;

- design entities and their attributes, relationships between them and describe them;

- draw an ER diagram depicting the designed entities, attributes (with data types), and relationships;

Expectation the completed task by criteria:

- A high-level overview of functions and use provided;

- Entities, their attributes, and relations are designed and fully described;

- An ER diagram depicting the designed entities, their attributes with data types, and relationships are drawn;

High-Level Overview of Functions and Use Cases

The system will manage various aspects of an F1 competition, including event scheduling, ticket sales, and participant management. The primary functions may include:

1)Event Management: Schedule and manage races, qualifying sessions, and practice sessions;

2)Ticket Management: Allow users to buy, view, and manage tickets for different events;

3)Driver Management: Track drivers, their teams, and their participation in events;

4)Reporting and Analytics: Generate reports on ticket sales, event performance, and driver statistics;

5)Venue Management: Manage the venues where events are held;

Entities and their Attributes

1)Event

Attributes:

- EventID (Integer, Primary Key)
- EventName (Varchar)
- EventDate (Date)
- Location (Varchar)
- Type (Enum: Race, Qualifying, Practice)
- VenueID (Integer, Foreign Key, references Venue)

2)Ticket

Attributes:

- TicketID (Integer, Primary Key)
- EventID (Integer, Foreign Key, references Event)
- SeatNumber (Varchar)
- Price (Decimal)
- CustomerID (Integer, Foreign Key, references Customer)

3)Customer

Attributes:

- CustomerID (Integer, Primary Key)
- FirstName (Varchar)
- LastName (Varchar)
- Email (Varchar)
- PhoneNumber (Varchar)

4)Team

Attributes:

- TeamID (Integer, Primary Key)
- TeamName (Varchar)
- Country (Varchar)

5)Driver

Attributes:

- DriverID (Integer, Primary Key)
- FirstName (Varchar)
- LastName (Varchar)
- TeamID (Integer, Foreign Key, references Team)

6)Venue

Attributes:

- VenueID (Integer, Primary Key)
- VenueName (Varchar)
- Location (Varchar)

7)Participation

Attributes:

- ParticipationID (Integer, Primary Key)
- DriverID (Integer, Foreign Key, references Driver)
- EventID (Integer, Foreign Key, references Event)
- Position (Integer) – the finishing position of the driver in the event

Relationships

Event

- Primary Key (PK): EventID
- Foreign Key (FK): VenueID (references Venue)
- Relationships: An event takes place at a venue.

Ticket

- Primary Key (PK): TicketID
- Foreign Keys (FK): EventID (references Event), CustomerID (references Customer)
- Relationships: A ticket is associated with one event and purchased by one customer.

Customer

- Primary Key (PK): CustomerID
- Relationships: A customer can purchase multiple tickets.

Team

- Primary Key (PK): TeamID
- Relationships: A team can have multiple drivers.

Driver

- Primary Key (PK): DriverID
- Foreign Key (FK): TeamID (references Team)
- Relationships: A driver represents one team in an event.

Venue

- Primary Key (PK): VenueID
- Relationships: A venue can host multiple events.

Participation

- Primary Key (PK): ParticipationID
- Foreign Keys (FK): DriverID (references Driver), EventID (references Event)
- Relationships: Participation links drivers to their respective events.

Summary of Relationships:

Ticket → Customer: One ticket is associated with one customer and One Customer might have many tickets (Many-to-One);

Event → Venue: One event is associated with one venue and one venue could have many events (Many-to-One);

Ticket → Event: One ticket is associated with one event and one event might have many tickets (Many-to-One);

Driver → Team: One driver is associated with one team and team can have many drivers (Many-to-One);

Event and Driver Tables have many-to-many relationships; One Event can have many Drivers and one Driver can participate in many Events; To reflect this relationship in terms of database Participation table was created; Participation table has two FK`s mapped to ID`s of Event and Driver tables (Many-to-Many);

ER Diagram Creation

1. Diagramming Tool (draw.io);

2. Start with Entities and create a rectangle for each entity (Event, Ticket, Customer, Team, Driver, Venue, Participation); Primary keys (PK) and Foreign keys (FK);

3. Attributes inside each rectangle, documented with their data types;

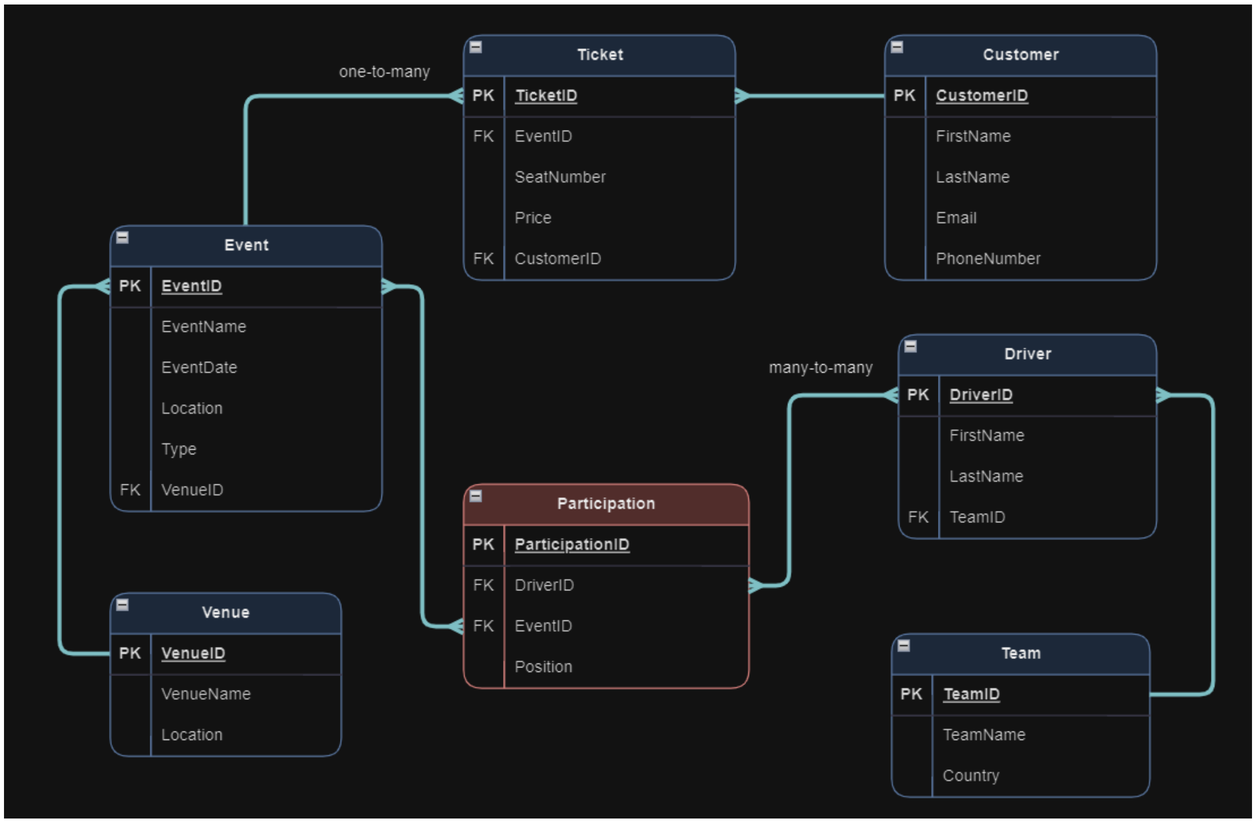
- Event (PK: EventID, EventName, EventDate, Location, Type, FK: VenueID);
- Ticket (PK: TicketID, FK: EventID, SeatNumber, Price, FK: CustomerID);
- Customer (PK: CustomerID, Name, Email, Phone);
- Team (PK: TeamID, TeamName, Country);
- Driver (PK: DriverID, FirstName, LastName, FK: TeamID);
- Venue (PK: VenueID, VenueName, Address, Capacity);
- Participation (PK: ParticipationID, FK: DriverID, FK: EventID, Position);

4. Draw Relationships (connect entities using lines, labeling lines to indicate relationship types as one-to-many and many-to-many)

Specific Relationship Guidelines:

- Event to Venue: One-to-many
- Event to Ticket: One-to-many
- Ticket to Customer: One-to-many
- Team to Driver: One-to-many
- Driver to Event: Many-to-many (via Participation)

Diagram (ER DESIGN F1 COMPETITION)



//some SQL scripts examples:

create database:

```
CREATE DATABASE formula_one;
```

create table:

```
CREATE TABLE Event (
    EventID INT AUTO_INCREMENT PRIMARY KEY,
    EventName VARCHAR(255) NOT NULL,
    EventDate DATE NOT NULL,
    Location VARCHAR(255),
    Type ENUM('RACE', 'QUALIFYING', 'PRACTICE') NOT NULL,
    VenueID INT,
    FOREIGN KEY (VenueID) REFERENCES Venue(VenueID)
);
```

insert data into table:

```
START TRANSACTION;

INSERT INTO customer (FirstName, LastName, Email, PhoneNumber) VALUES
('John', 'Doe', 'jdoe@gmail.com', '100-155655');
INSERT INTO customer (FirstName, LastName, Email, PhoneNumber) VALUES
('Henry', 'Martinez', 'hmartinez@hotmail.com', '555-357890');
INSERT INTO customer (FirstName, LastName, Email, PhoneNumber) VALUES
('Jack', 'Clark', 'jclark@yahoo.com', '777-579012');
INSERT INTO customer (FirstName, LastName, Email, PhoneNumber) VALUES
('Noah', 'Lewis', 'nlewis@email.com', '999-791234');
INSERT INTO customer (FirstName, LastName, Email, PhoneNumber) VALUES
('Lily', 'Lee', 'llee@hotmail.com', '101-802345');

-- Add more INSERT statements as needed

-- If an error occurs, use the following command to roll back
-- ROLLBACK;

COMMIT;
```

read data from table:

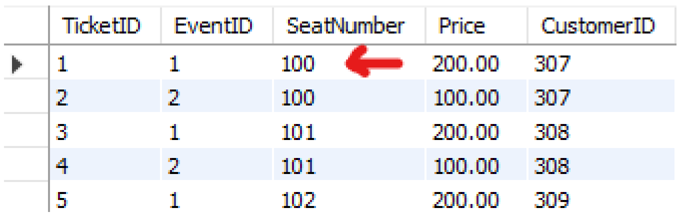
```
-- insert team
-- insertion data into team /*insert_teams.sql*/
/*read from team after insertion*/
select * from team;
```

update data:

//customer change place:

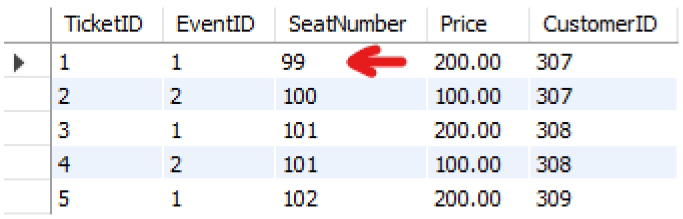
```
-- update operation;
-- let say customer want to change his seat place;
select * from ticket; -- check for all tickets
select * from ticket where CustomerID = 307; -- find concrete customer by ID
from ticket table
UPDATE ticket
SET SeatNumber = '99'
WHERE TicketID = 1 AND EventID = 1 AND SeatNumber = '100' AND Price = 200.00
AND CustomerID = 307; -- updating place from 100 --> 99
select * from ticket where TicketID = 1; -- check for update
```

before



TicketID	EventID	SeatNumber	Price	CustomerID
1	1	100	200.00	307
2	2	100	100.00	307
3	1	101	200.00	308
4	2	101	100.00	308
5	1	102	200.00	309

after



TicketID	EventID	SeatNumber	Price	CustomerID
1	1	99	200.00	307
2	2	100	100.00	307
3	1	101	200.00	308
4	2	101	100.00	308
5	1	102	200.00	309

delete data:

//ticket deletion

```
-- delete operation;
-- let say customer want to decline his payment for Qualification and to
refuse ticket;
select * from ticket; -- check for all tickets
select * from ticket where CustomerID = 307; -- find concrete customer by ID
from ticket table
DELETE FROM ticket
WHERE TicketID = 2 AND CustomerID = 307;
select * from ticket where CustomerID = 307; -- check for deletion data
-- 38 records were before removing, and 37 stands after
-- as well amount of tickets for this customer changed properly
```