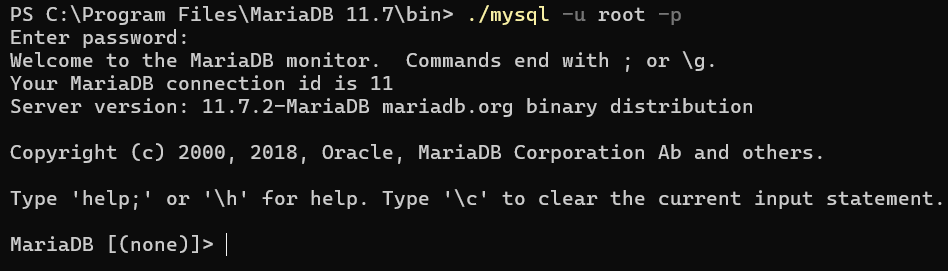
**Homework 6**

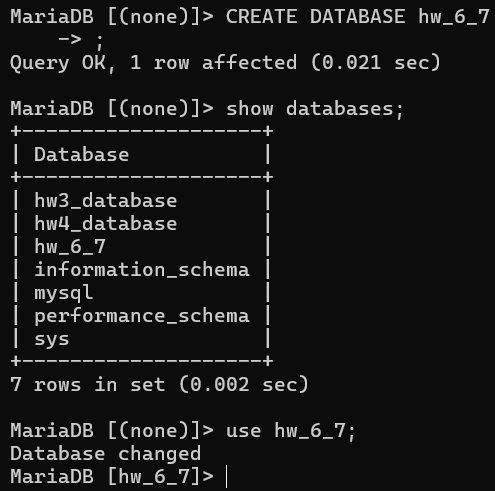
**Problem Statement:** This Assignment uses the following Travel Agency database schema with the specified primary and foreign key constrains:

* Booking(agents: String, traveler\_ssn: integer, trip\_id: integer)
  + *Foreign Keys*: traveler\_ssn references Traveler(ssn), trip\_id references Trip(id), agent references TravelAgent(name)
* GoesOn(ssn: integer, id: integer)
  + *Foreign Keys*: ssn references Traveler(ssn), id references Trip(id)
* Leg(Trip\_id: integer, startLocation: String, endLocation: String, startDate: Date, endDate: Date)
  + *Foreign Keys*: trip\_id references Trip(id)
* Owns(ssn: integer, passport number: integer, country: String)
  + *Foreign keys*: ssn references Traveler(ssn)
* Passport(passport number: integer, country: String, expirationDate: Date, holderName: String)
* TravelAgent(name: String, years experience: integer, phone: String)
* Traveler(name: String, ssn: integer, dob: Date)
* Trip(id: integer, start location: String, end location: String, start date: Date, end date: Date)

**Part 0: Start MariaDB**



**Part 1: Create a new Database**

****

**Part 2: Create Tables with Constraints**

1. **Primary Keys and Foreign Keys:** Include primary keys for each table and establish foreign key constrains as specified in the schema
2. **Attribute-Based Constraints:** Define the following two attribute based constraints –
   1. years\_experience in TravelAgent must be greater than or equal to 1.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE TravelAgent(  name VARCHAR(50),  years\_experience INTEGER CHECK(years\_experience >= 1),  phone VARCHAR(50),  PRIMARY KEY(name)  );  DESCRIBE TravelAgent;  SHOW TABLES; |  |

* 1. expirationDate in Passport must be after January 1, 2020.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE Passport(  passport\_number INTEGER,  country VARCHAR(50),  expirationDate DATE CHECK(expirationDate > '2020-01-01'),  holderName VARCHAR(50),  PRIMARY KEY(passport\_number, country)  );  DESCRIBE Passport;  SHOW TABLES; |  |

1. **Tuple-Based Constraints:** Define the following tuple-based constraints:
   1. In Leg, ensure endDate is not earlier than startDate

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE Leg(  trip\_id INTEGER,  startLocation VARCHAR(50),  endLocation VARCHAR(50),  startDate DATE,  endDate DATE,  CHECK (endDate >= startDate),  PRIMARY KEY(trip\_id, startLocation, endLocation),  FOREIGN KEY(trip\_id) REFERENCES Trip(id)  );  DESCRIBE Leg;  SHOW TABLES; |  |

* 1. In Trip, ensure end\_date is not earlier than start\_date

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE Trip(  id INTEGER,  start\_location VARCHAR(50),  end\_location VARCHAR(50),  start\_date DATE,  end\_date DATE,  CHECK(end\_date >= start\_date),  PRIMARY KEY(id)  );  DESCRIBE Trip;  SHOW TABLES; |  |

1. **Cascade on Update and Delete:** Set up a foreign key constrain such that updating and deleting a Traveler will automatically update and delete all related records in Booking, Owns, and GoesOn tables.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE Owns(  ssn INTEGER,  passport\_number INTEGER,  country VARCHAR(50),  PRIMARY KEY(passport\_number, country),  FOREIGN KEY(ssn) REFERENCES Traveler(ssn)  ON DELETE CASCADE  ON UPDATE CASCADE  );  DESCRIBE Owns;  SHOW TABLES; |  |

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE GoesOn(  ssn INTEGER,  id INTEGER,  PRIMARY KEY(ssn, id),  FOREIGN KEY(ssn) REFERENCES Traveler(ssn)  ON DELETE CASCADE  ON UPDATE CASCADE,  FOREIGN KEY(id) REFERENCES Trip(id)  );  DESCRIBE GoesOn;  SHOW TABLES; |  |

1. **Cascade on Delete:** Implement a foreign key constraint in Booking such that if an agent in TravelAgent is deleted the agent attribute in any related Booking records is deleted.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE Booking(  agent VARCHAR(50),  traveler\_ssn INTEGER,  trip\_id INTEGER,  PRIMARY KEY(agent, traveler\_ssn, trip\_id),  FOREIGN KEY(traveler\_ssn) REFERENCES Traveler(ssn)  ON DELETE CASCADE  ON UPDATE CASCADE,  FOREIGN KEY(trip\_id) REFERENCES Trip(id),  FOREIGN KEY(agent) REFERENCES TravelAgent(name)  ON DELETE CASCADE  );  DESCRIBE Booking;  SHOW TABLES; |  |

1. **NOT NULL Constraints:** Add a NOT NULL constrain to the name column in Traveler.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE Traveler(  name VARCHAR(50) NOT NULL,  ssn INTEGER,  dob DATE,  PRIMARY KEY(ssn)  );  DESCRIBE Traveler;  SHOW TABLES; |  |

**Part 3: Insert Data**

1. \*\* Insert Into ‘Traveler’, ‘TravelAgent’, ‘Trip’, and ‘Passport’ \*\*:

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO Traveler (name, ssn, dob)  VALUES  ('John Doe', 101, '1985-06-12'),  ('Alice Brown', 102, '1992-03-05'),  ('Mike Johnson', 103, '1998-09-17'),  ('Lisa Turner', 104, '2000-12-22'),  ('Sarah Connor', 105, '2003-11-01');  SELECT \*  FROM Traveler; |  |

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO TravelAgent (name, years\_experience, phone)  VALUES  ('Emily Clark', 12, '123-456-7890'),  ('Robert Smith', 8, '234-567-8901'),  ('Anna Wilson', 15, '345-678-9012'),  ('Michael Davis', 10, '456-789-0123'),  ('Mary Johnson', 3, '567-890-1234');  SELECT \*  FROM TravelAgent; |  |

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO Trip (id, start\_location, end\_location, start\_date, end\_date)  VALUES  (201, 'New York', 'Paris', '2023-07-10', '2023-07-20'),  (202, 'Tokyo', 'Sydney', '2023-08-01', '2023-08-15'),  (203, 'London', 'Rome', '2023-09-05', '2023-09-15'),  (204, 'Berlin', 'Tokyo', '2023-10-02', '2023-10-12'),  (205, 'Miami', 'New York', '2023-10-22', '2023-10-29');  SELECT \*  FROM Trip; |  |

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO Passport (passport\_number, country, expirationDate, holderName)  VALUES  (3001, 'USA', '2025-11-30', 'John Doe'),  (3002, 'Canada', '2026-08-20', 'Alice Brown'),  (3003, 'UK', '2024-09-15', 'Mike Johnson'),  (3004, 'Australia', '2027-02-10', 'Lisa Turner'),  (3005, 'France', '2023-12-05', 'Sarah Connor');  SELECT \*  FROM Passport; |  |

1. \*\* Insert into ‘Owns’ \*\*:

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| SELECT \*  FROM Traveler;  INSERT INTO Owns (ssn, passport\_number, country)  VALUES  (101, 3001, 'USA'),  (102, 3002, 'Canada'),  (103, 3003, 'UK'),  (104, 3004, 'Australia'),  (105, 3005, 'France');  SELECT \*  FROM Owns;  SELECT \*  FROM Traveler; |  |

1. \*\* Insert into ‘Booking’ \*\*:

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| SELECT \*  FROM Traveler;  SELECT \*  FROM Trip;  SELECT \*  FROM TravelAgent;  INSERT INTO Booking (agent, traveler\_ssn, trip\_id)  VALUES  ('Emily Clark', 101, 201),  ('Robert Smith', 102, 202),  ('Anna Wilson', 103, 203),  ('Michael Davis', 104, 204),  ('Emily Clark', 105, 205);  SELECT \*  FROM Booking; |  |

1. \*\* Inserts into ‘GoesOn’ \*\*:

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| SELECT \*  FROM Traveler;  SELECT \*  FROM Trip;  INSERT INTO GoesOn (ssn, id)  VALUES  (101, 201),  (102, 202),  (103, 203),  (104, 204),  (105, 205);  SELECT \*  FROM GoesOn; |  |

1. \*\* Insert into ‘Leg’ \*\*:

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| SELECT \*  FROM Trip;  INSERT INTO Leg (trip\_id, startLocation, endLocation, startDate, endDate)  VALUES  (201, 'New York', 'Paris', '2023-07-10', '2023-07-20'),  (202, 'Tokyo', 'Sydney', '2023-08-01', '2023-08-15'),  (203, 'London', 'Rome', '2023-09-05', '2023-09-15'),  (204, 'Berlin', 'Tokyo', '2023-10-02', '2023-10-12'),  (205, 'Miami', 'New York', '2023-10-22', '2023-10-29');  SELECT \*  FROM Leg; |  |

**Part 4: Triggers**

1. Trigger 1:
   1. First, create the DeletedTravelerLog table to store logs of deleted travelers. This table should include columns for name and ssn to record the details of each deleted traveler.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE DeletedTravelerLog(  name VARCHAR(50),  ssn INTEGER,  PRIMARY KEY(ssn)  );  DESCRIBE DeletedTravelerLog;  SHOW TABLES; |  |

* 1. Next, create a trigger on the Traveler table. Before a record is deleted, this trigger should log the name and ssn of the traveler to the DeletedTravelerLog table.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TRIGGER DeletedTravelerLogTrigger  BEFORE DELETE ON Traveler  FOR EACH ROW  INSERT INTO DeletedTravelerLog(name, ssn)  VALUES(OLD.name, OLD.ssn); |  |

1. Trigger 2:
   1. First, create the TravelerStats table. This table should include a column for the traveler’s ssn and a column for the total trip count to store the number of trips each traveler has taken.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TABLE TravelerStats(  ssn INTEGER,  trip\_count INTEGER DEFAULT 1,  PRIMARY KEY(ssn)  );  DESCRIBE TravelerStats;  SHOW TABLES; |  |

* 1. Next, create a trigger on the GoesOn table. After a new record is inserted into GoesOn, this trigger should update the trip count in the TravelerStats table for the traveler’s ssn to reflect the total number of trips.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| CREATE TRIGGER TravelerStatsTrigger  AFTER INSERT ON GoesOn  FOR EACH ROW  INSERT INTO TravelerStats(ssn)  VALUES(NEW.ssn)  ON DUPLICATE KEY UPDATE  trip\_count = trip\_count + 1; |  |

**Part 5: Constraint Scenarios**

1. **Delete a Trip ID in Trip (3 points):** Attempt to delete a Trip entry where the id is 201. Observe and describe the violation that occurs due to the foreign key constraints.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| DELETE FROM Trip  WHERE id = 201; |  |

The violation that occurs due to foreign key constraints is because there are multiple references to other child tables such as Booking, GoesOn, and Leg from which the parent table Trip references with no specification on how to handle deletions, so if there’s no specification foreign keys will reject the command if the value is being referenced in a child table. In the specific command, it says that the constraint is from the Booking table which indicates that the data that is attempted to be deleted is 201 which is used in the child table Booking.

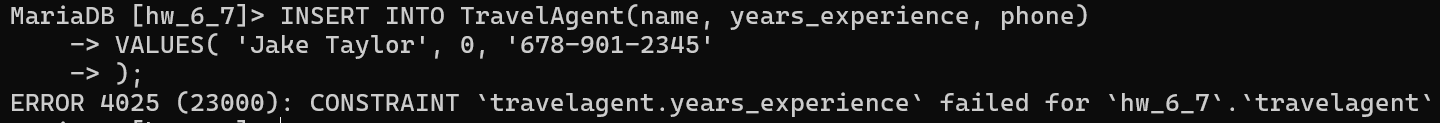
1. **Update an SSN in Owns (3 points):** Attempt to update the ssn in Owns that has the value 101 to 999. Describe the violation and explain how the foreign key constraint prevents this update.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| UPDATE Owns  SET ssn = 999  WHERE ssn = 101; |  |

The foreign key violation is triggered because the foreign key ssn in the child table Owns is connected to the parent table Traveler(ssn) and it rejected because the update attempts to change the value on a child row, which it rejects because there is not an equivalent value that is already in the parent table Traveler.

1. **Insert a Travel Agent with 0 Years of Experience (3 points):** Try to insert a new TravelAgent (’Jake Taylor’, 0, ’678-901-2345’). Describe the result and explain how the attribute-based constraint enforces data validity.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO TravelAgent(name, years\_experience, phone)  VALUES( 'Jake Taylor', 0, '678-901-2345'  ); |  |



The insertion into TravelAgent fails because of the constraint CHECK(years\_experience >= 1) on the TravelAgent table. This attribute based constraint enforces data validity because if a travel agent has no experience, there’s a good chance they aren’t a travel agent. It essentially insures that invalid or nonsensical data isn’t stored. The error shows that it was due to the constraint travelagent.years\_experience since it make sure there is a least 1 year od experience for each entry, but the attempt was for 0.

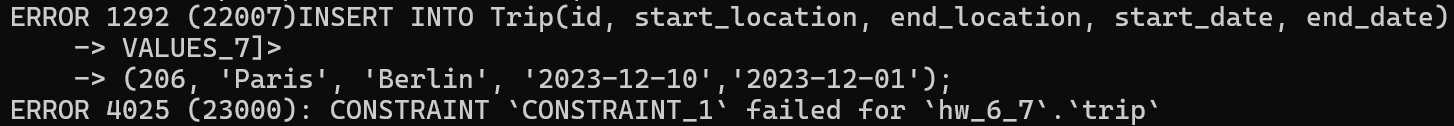
1. **Update Passport Expiration Date to an Invalid Date (3 points):** Attempt to update the Passport with the number 3001 to have an expiration date ’2018-12- 01’. Describe the result and explain the purpose of the attribute-based constraint on expiration dates.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| UPDATE Passport  SET expirationDate = '2018-12-01'  WHERE passport\_number = 3001; |  |

The reason why the Update Failed is because of the Constraint CHECK(expirationDate > '2020-01-01'), this is important because it ensure only passports that are valid unexpired passports are being included in the data. The data in the error message said it was from the value 2018, which is less than the checked value 2020.

1. **Insert a Trip with an Invalid Date Range (3 points):** Try to insert a new Trip with the following values (206, ’Paris’, ’Berlin’, ’2023-12-10’, ’2023-12-01’). Describe the violation and explain how the tuple-based constraint ensures logical date ranges.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO Trip(id, start\_location, end\_location, start\_date, end\_date)  VALUES  (206, 'Paris', 'Berlin', '2023-12-10','2023-12-01'); |  |



The violation occurred because of the constraint CHECK(end\_date >= start\_date) which indicates that the end date has to be after the start date, which makes sense because there’s no reason for a start date to be after an end date, so this can prevent nonsensical information from accidentally being entered.

1. **Update a Traveler to Cascade Changes (3 points):** Update the ssn of Traveler 102 to have the value 106 and observe how the update cascades to the Booking , Owns and GoesOn tables. Describe the effect and explain how the cascade on update policy maintains referential integrity.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| SELECT \*  FROM Traveler;  SELECT \*  FROM Booking;  SELECT \*  FROM Owns;  SELECT \*  FROM GoesOn;  UPDATE Traveler  SET ssn = 106  WHERE ssn = 102;  SELECT \*  FROM Traveler;  SELECT \*  FROM Booking;  SELECT \*  FROM Owns;  SELECT \*  FROM GoesOn; |  |

The CASCADE ON UPDATE policy maintains referential integrity as if there is an update in the parent table, such as an update to the ssn in Traveler, then the child tables that use Traveler.ssn as a foreign key for their own ssn values is automatically updated if the corresponding ssn value is referenced in that table. For example, by changing ssn from 102 to 106 in the parent table Traveler, the children tbales Booking, Owns, and GoesOn also changes the ssn value where 102 was previously referenced to 106.

1. **Delete a Travel Agent to Cascade changes in Booking (3 points):** Delete the TravelAgent ’Emily Clark’ and observe how the delete cascades to the Booking. Describe the effect and explain how the cascade on update policy maintains referential integrity.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| DELETE FROM TravelAgent  WHERE name = 'Emily Clark';  SELECT \*  FROM TravelAgent;  SELECT \*  FROM Booking; |  |

When deleting the name Emily Clark from the parent table TravelAgent, it automatically deletes all the rows that reference TravelAgent.name for their own agent name attributes, so the rows that previously held information with the agent Emily Clark are now deleted from the children tables as well as the parent table, so it maintains consistency as a foreign key so there are no references in the children tables for the foreign key for the agent name attribute that is not in the parent table in order to maintain referential integrity.

1. **Insert a Traveler with a NULL Name (3 points):** Attempt to insert a Traveler with the following values (NULL, 107, ’2001-05-10’) to test the NOT NULL constraint. Describe the result and explain why a NOT NULL constraint is important for certain fields.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO Traveler(name, ssn, dob)  VALUES  (NULL, 107, '2001-05-10'); |  |

As the result of trying to insert a value in the Traveler Table that has NULL in the name it creates the error that name cannot be null, which make sense because in the Traveler creation there was a specification for name to be NOT NULL. NOT NULL is important for certain fields because it enforces data validity, for this scenario specifically, it wouldn’t make sense for a travler to have no name but they have a social security number and date of birth which is much more sensitive information than a name.

1. **Trigger 1: Deletion of Traveler (3 points):** Delete a Traveler with ssn 103 and verify that the trigger logs the deleted traveler’s name and ssn to the DeletedTravelerLog table. Describe the effect of the trigger.

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| SELECT \*  FROM Traveler;  DELETE FROM Traveler  WHERE ssn = 103;  SELECT \*  FROM Traveler;  SELECT \*  FROM DeletedTravelerLog; |  |

Due to the trigger, when the person with the ssn 103 is deleted from the Traveler table as instructed, but the added result of a trigger automates the process of adding the name and social security number of the deleted row to be added to a DeletedTravelerLog table.

1. **Trigger 2: Insert into GoesOn to Update TravelerStats (3 points):** Insert a new record with the values (104, 205) into GoesOn and check the TravelerStats table to confirm the trip count updates as expected. Describe the result and explain how the trigger maintains the TravelerStats table

|  |  |
| --- | --- |
| SQL Statements | SQL Implementation |
| INSERT INTO GoesOn(ssn, id)  VALUES(104, 204);  SELECT \*  FROM TravelerStats;  INSERT INTO GoesOn(ssn, id)  VALUES(104, 205);  SELECT \*  FROM TravelerStats; |  |

The when values are inserted into goes in, under the assumption that there is already a duplicated value in the TravelerStats table, when there is a value entered into the GoesOn table. I inserted the values for GoesOn(ssn, id) with (104, 204) to mimic the first time the ssn 104 with trip is being entered in order for TravelerStats to be set to 1 for the trip count, and then added the example trip (104, 205) for it to count another trip and bring the TravlerStats.trip\_count added to 2 when values are inserted into GoesOn.