

# **Sri Lanka Institute of Information Technology**



## **Online Airline Reservation System**

**Group Assignment**

**IE2042- Database Management Systems for  
Security**

## **Group Details:**

	<b>Student Registration Number</b>	<b>Student Name</b>
<b>1</b>	IT21170416	Pemarathna I.R.C
<b>2</b>	IT21163340	Peiris H.R.T
<b>3</b>	IT21175602	Nilakshana H.A.N
<b>4</b>	IT21188572	Fernando S.D.K.

## **Table of Contents**

1. Assumptions.....	4
2. ERD Model .....	5
3. Logical Model .....	6
4. Normalized Logical Model .....	7
5. MS SQL server and Suitable sample data .....	9
i. SQL Screenshots .....	9
ii. SQL codes as a separate script.....	13
iii. Triggers.....	17
iv. Views .....	19
v. Indexes .....	21
vi. Stored Procedures .....	22
6. Database Vulnerabilities .....	26
i. SQL Injection (SQLi) Attack.....	26
ii. Denial Of Service Attack (DOS) .....	30
7. References .....	32

## **Part 01**

### **1. Assumptions**

- 1) Flight\_Leg entity is a weak entity of the Flight entity.
- 2) Leg\_instance entity is a weak entity of the Flight Leg entity.
- 3) Seat entity is a weak entity of the Leg instance entity.
- 4) Flight\_fare entity is a weak entity of the flight.
- 5) Flight\_ No can determined the schedule\_arr\_time and schedule\_dep\_time .

Flight\_No → schedule\_arr\_time,schedule\_dep\_time

- 6) AirplaneID can determine the arr\_time and dep\_time

AirplaneID → arr\_time, dep\_time

- 7) The AirplaneID in the Airplane table is the same as the AirplaneID in the Leg\_instance table, the AirplaneID in the Leg\_instance Table is the same as the AirplaneID in the Airplane\_Schedule Table, so the AirplaneID in the Airplane Table is the same as the AirplaneID in the Airplane\_Schedule table. Airplane A, Leg\_instance L, Airplane\_Schedule AS

A.AirplaneID = ~~L.AirplaneID~~

~~L.AirplaneID~~ = AS.AirplaneID

A. AirplaneID = AS.AirplaneID

- 8) The Flight\_ No in the Flight table is the same as the Flight\_ No in the Flight\_Leg table, the Flight\_ No in the Flight\_Leg Table is the same as the Flight\_ No in the Flight\_Schedule Table, so the Flight\_ No in the Flight Table is the same as the Flight\_ No in the Flight\_Schedule.

Flight F, Flight\_Leg FL, Flight\_Schedule FS

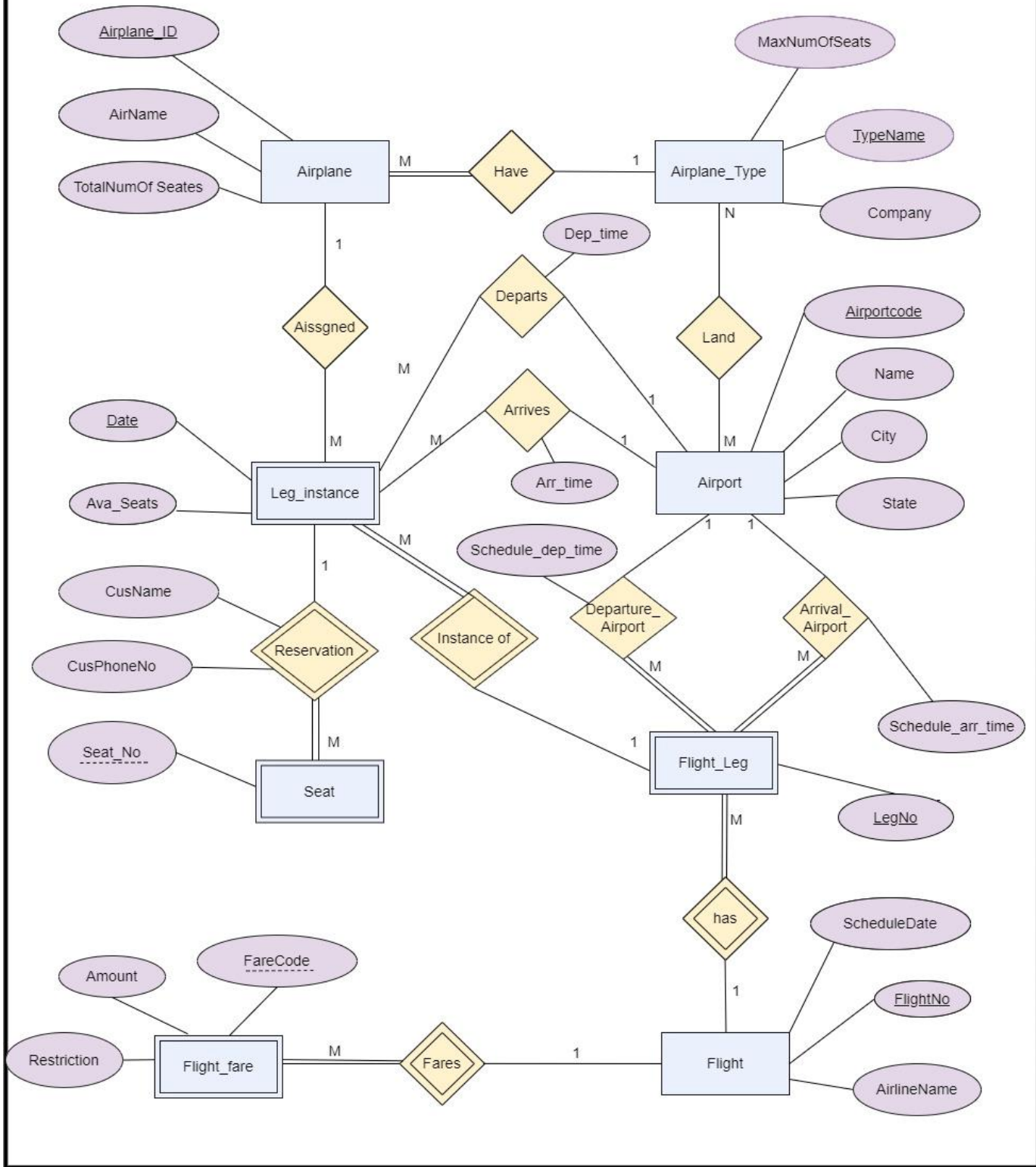
F. Flight\_ No = ~~FL. Flight\_ No~~

~~FL. Flight\_ No~~ = FS. Flight\_ No

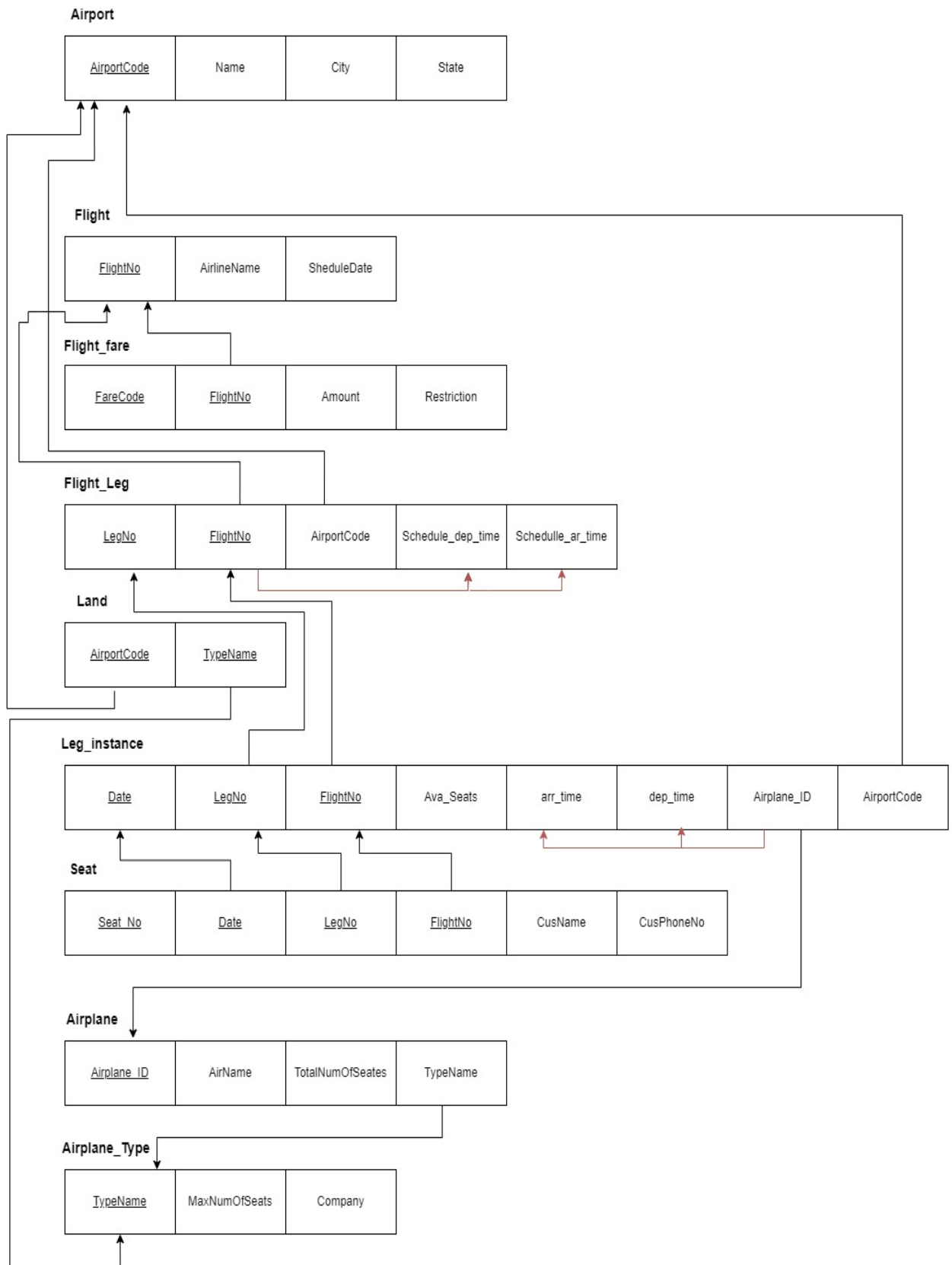
F. Flight\_ No = FS. Flight\_ No

## 2. ERD Model

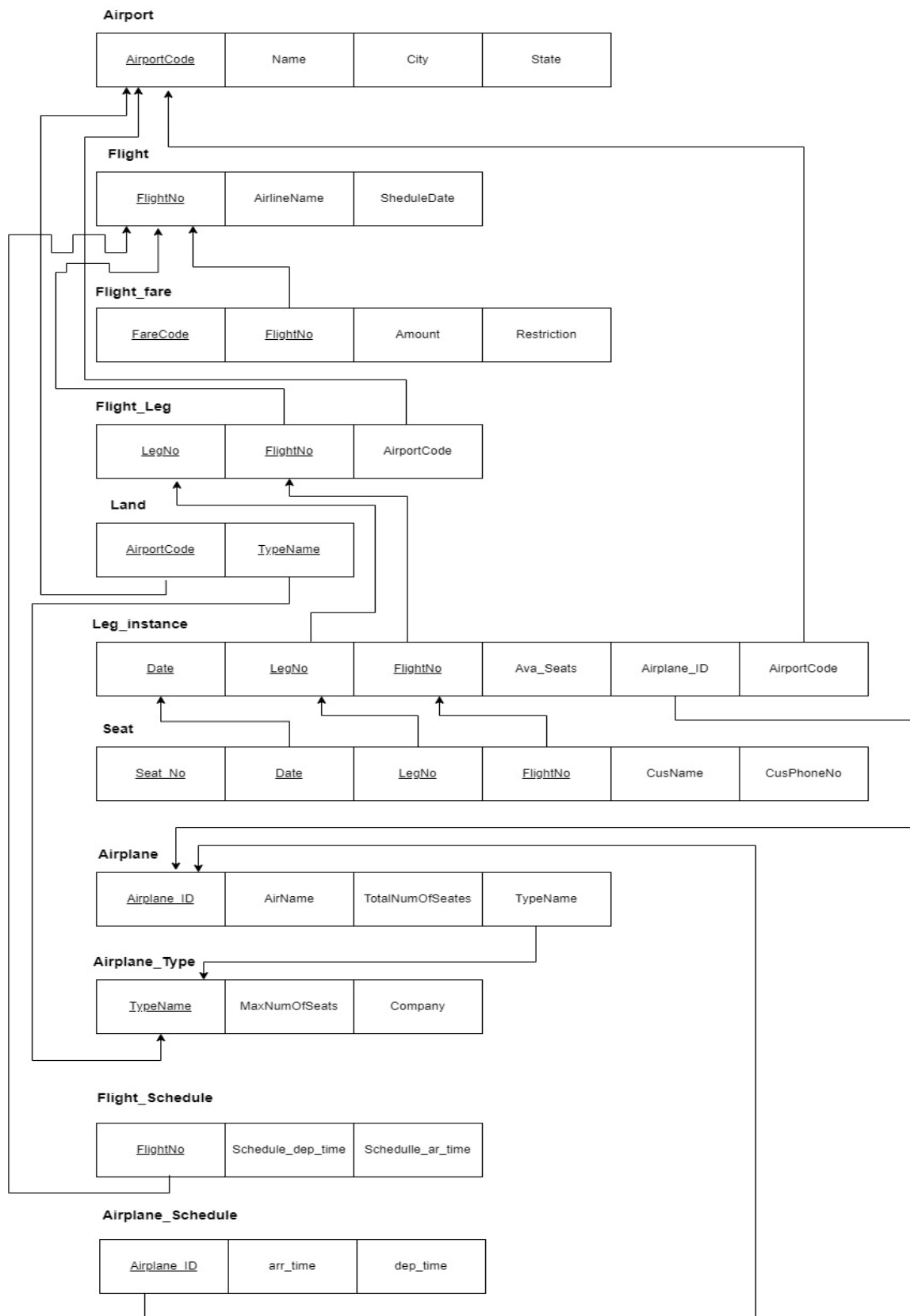
### Online Airline Reservation System



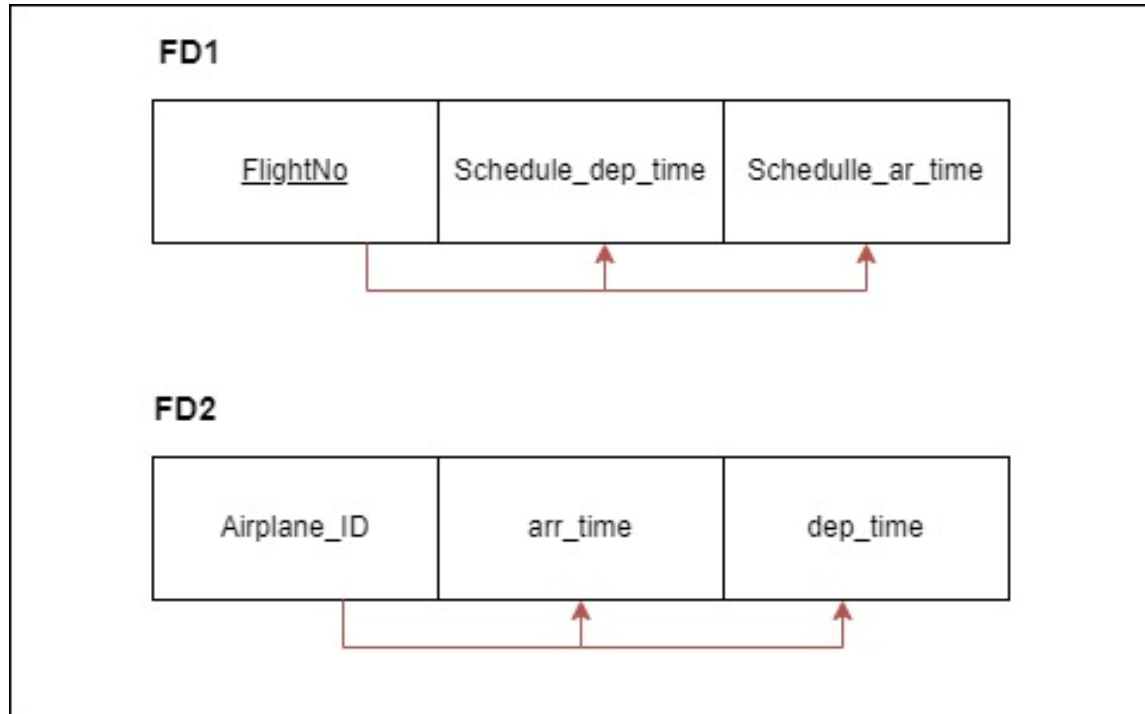
### 3. Logical Model



## 4. Normalized Logical Model



## Functional Dependencies





## 5. MS SQL server and Suitable sample data

### i. SQL Screenshots

```
2  --Creating Airport Table--
3
4  Create table Airport(
5      Airportcode varchar(10) NOT NULL,
6      Name varchar(50) NOT NULL,
7      City varchar(50) NOT NULL,
8      State varchar(50) NOT NULL,
9      constraint Airport_pk primary key (Airportcode),
10     CONSTRAINT checkAirport_Airportcode check (Airportcode LIKE '[A-Z][0-9][0-9][0-9][0-9][0-9][0-9]')
11 );
12
13 --Creating Airplane_Type Table--
14
15 Create table Airplane_Type(
16     TypeName varchar(50) NOT NULL,
17     MaxNumOfSeats varchar(50) NOT NULL,
18     Company varchar(50) NOT NULL,
19     constraint Airplane_Type_pk primary key (TypeName),
20 );
21
22 --Creating Land Table--
23
24 Create table Land(
25     Airportcode varchar(10) NOT NULL,
26     TypeName varchar(50) NOT NULL,
27     constraint Land_pk primary key (Airportcode,TypeName),
28     constraint Land_fk1 foreign key (Airportcode) REFERENCES Airport (Airportcode),
29     constraint Land_fk2 foreign key (TypeName) REFERENCES Airplane_Type (TypeName),
30 );
31
32 --Creating Airplane Table--
33
34 Create table Airplane(
35     AirplaneID varchar(10) NOT NULL,
36     AirName varchar(50) NOT NULL,
37     TotalNumOfSeats integer,
38     TypeName varchar(50) NOT NULL,
39     constraint airplane_pk primary key (AirplaneID),
40     constraint airplane_fk foreign key (TypeName) REFERENCES Airplane_Type (TypeName),
41     constraint checkAirplane_AirplaneID check (AirplaneID LIKE '[A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
42 );
43
44 --Creating Flight Table--
45
46 Create table Flight(
47     FlightNo varchar(50) NOT NULL,
48     AirlineName varchar(50) NOT NULL,
49     ScheduleDate varchar(50) NOT NULL,
50     constraint Flight_pk primary key (FlightNo),
51     constraint checkFlight_FlightNo check (FlightNo LIKE '[A-Z][A-Z][0-9][0-9][0-9]')
52 );
53
54 --Creating Flight_Leg Table--
55
56 Create table Flight_Leg(
57     LegNo varchar(10) NOT NULL,
58     FlightNo varchar(50) NOT NULL,
59     Airportcode varchar(10) NOT NULL,
60     constraint Flight_Leg_pk primary key (LegNo,FlightNo),
61     constraint Flight_Leg_fk1 foreign key (Airportcode) REFERENCES Airport (Airportcode),
62     constraint Flight_Leg_fk2 foreign key (FlightNo) REFERENCES Flight (FlightNo)
63 );
64
```

```

66 --Creating Leg_instance Table--
67
68 Create table Leg_instance(
69     Date date NOT NULL,
70     LegNo varchar(10) NOT NULL,
71     FlightNo varchar(50) NOT NULL,
72     Ava_Seats varchar(50) NOT NULL,
73     AirplaneID varchar(10) NOT NULL,
74     Airportcode varchar(10) NOT NULL,
75     constraint Leg_instance_pk primary key (Date,LegNo,FlightNo),
76     constraint Leg_instance_fk1 foreign key (LegNo,FlightNo) REFERENCES Flight_Leg (LegNo,FlightNo),
77     constraint Leg_instance_fk2 foreign key (AirplaneID) REFERENCES Airplane (AirplaneID),
78     constraint Flight_Leg_fk3 foreign key (Airportcode) REFERENCES Airport (Airportcode),
79 );
80
81
82 --Creating Seat Table--
83
84 Create table Seat(
85     Seat_No varchar(10) NOT NULL,
86     Date date NOT NULL,
87     LegNo varchar(10) NOT NULL,
88     FlightNo varchar(50) NOT NULL,
89     CusName varchar(50) NOT NULL,
90     CusPhoneNo varchar(50) NOT NULL,
91     constraint Seat_pk primary key (Seat_No,Date,LegNo,FlightNo),
92     constraint check_Seat_CusPhoneNo check (CusPhoneNo LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
93     constraint Seat_fk foreign key (Date,LegNo,FlightNo) REFERENCES Leg_instance (Date,LegNo,FlightNo)
94 );
95

```

```

95 --Creating Flight_fare Table--
96
97 Create table Flight_fare(
98     Farecode varchar(20) NOT NULL,
99     FlightNo varchar(50) NOT NULL,
100     Amount varchar(20) NOT NULL,
101     Restriction varchar(50) NOT NULL,
102     constraint Flight_fare_pk primary key (Farecode,FlightNo),
103     constraint Flight_fare_fk foreign key (FlightNo) REFERENCES Flight (FlightNo)
104 );
105
106 --Creating Airplane_Schedule Table--
107
108 CREATE TABLE Airplane_Schedule (
109     AirplaneID varchar (10),
110     arr_time time NOT NULL,
111     dep_time time NOT NULL,
112     constraint Airplane_Schedule_PK primary key (AirplaneID),
113     constraint Airplane_Schedule_fk foreign key (AirplaneID) REFERENCES Airplane (AirplaneID)
114 );
115
116 --Creating Flight_Schedule Table--
117
118 CREATE TABLE Flight_Schedule (
119     FlightNo varchar (50) NOT NULL,
120     schedule_arr_time time(0) NOT NULL,
121     schedule_dep_time time(0) NOT NULL,
122     constraint Flight_schedule_PK primary key (FlightNo),
123     constraint Flight_Schedule_fk foreign key (FlightNo) REFERENCES Flight (FlightNo)
124 );

```

```

125 /*insert airport table*/
126 insert into Airport values('A020171','Sydney','San Mateo','California');
127 insert into Airport values('B020221','Frankfurt Airport','Frankfurt','Hesse');
128 insert into Airport values('C020271','Singapore','Salt Lake City','Utah');
129 insert into Airport values('D020251','Helsinki Airport','Ivalo','Kemi-Tornio');
130 insert into Airport values('E020230','Brisbane International Airport','Brisbane','Victoria');
131 insert into Airport values('F171223','Bandaranaike International Airport','katunayaka','Colombo');
132
133
134 /*insert Airplane_Type*/
135 insert into Airplane_Type values('Airlines 787','300','Rolls Royce');
136 insert into Airplane_Type values('Airbus A330','600','Pratt & Whitney');
137 insert into Airplane_Type values('Boeing 747','400','Pratt & Whitney');
138 insert into Airplane_Type values('Boeing 757','500','Miami-based Eastern Airlines');
139 insert into Airplane_Type values('Douglas 80','200','Pratt & Whitney JT8-D');
140 insert into Airplane_Type values('Antonov AN28','350','LET Aircraft Industries');
141
142
143 /*insert Land*/
144 insert into Land values('A020171','Airlines 787');
145 insert into Land values('B020221','Airbus A330');
146 insert into Land values('C020271','Boeing 747');
147 insert into Land values('D020251','Boeing 757');
148 insert into Land values('E020230','Douglas 80');
149 insert into Land values('F171223','Antonov AN28');
150
151
152 /*insert Airplane*/
153 insert into Airplane values('R200171223','Douglas DC-3',295,'Boeing 757');
154 insert into Airplane values('N987654321','Airbus A321XLR ',467,'Boeing 747');
155 insert into Airplane values('Q224466889','COMAC C919',440,'Airbus A330');
156 insert into Airplane values('D779955331','Universal Hydrogen ATR 72',174,'Antonov AN28');

```

```

164 /*insert Flight*/
165 insert into Flight values('AL987','Air Berlin','2022-12-23');
166 insert into Flight values('TY456','Belair','2022-10-19');
167 insert into Flight values('KL203','Paramount','2022-11-04');
168 insert into Flight values('QE333','Oman Air','2022-10-26');
169 insert into Flight values('BH912','IndiGo','2023-01-02');
170 insert into Flight values('LK729','Jetstar Asia','2022-10-23');
171 insert into Flight values('PR914','SL Airways','2022-12-01');
172
173
174 /*insert Flight_leg*/
175 insert into Flight_leg values('2','AL987','A020171');
176 insert into Flight_leg values('3','PR914','B020221');
177 insert into Flight_leg values('4','AL987','C020271');
178 insert into Flight_leg values('1','QE333','D020251');
179 insert into Flight_leg values('2','LK729','E020230');
180 insert into Flight_leg values('3','BH912','F171223');
181
182
183 /*insert Leg_instance*/
184 insert into Leg_instance values('2022-11-14','2','AL987','74','R200171223','A020171');
185 insert into Leg_instance values('2022-12-23','3','PR914','52','N987654321','B020221');
186 insert into Leg_instance values('2023-01-01','2','LK729','158','D779955331','E020230');
187 insert into Leg_instance values('2022-12-14','1','QE333','74','Q224466889','D020251');
188
189
190 /*insert seat*/
191 insert into seat values('11F','2022-11-14','2','AL987','Kaun Perera','0859632147');
192 insert into seat values('55JH','2022-12-23','3','PR914','Niki watson','0856329741');
193 insert into seat values('14LA','2023-01-01','2','LK729','Mary Ann','1253698521');
194 insert into seat values('13HA','2022-12-14','1','QE333','Sadun perera','9476966518');
195

```

```

192  /*insert Flight_fare*/
193  insert into Flight_fare values('AF1015','AL987','65000','plastic');
194  insert into Flight_fare values('CDIJ17','TY456','275000','bring metal');
195  insert into Flight_fare values('GNQS03','KL203','74000','bring pets and plants');
196  insert into Flight_fare values('PW4589','QE333','65000','eat fish and meats');
197  insert into Flight_fare values('YBHKL1','PR914','54000','bring over 50KG travel bags');
198  insert into Flight_fare values('NQS144','BH912','41000','bring pets');
199  insert into Flight_fare values('NOQS51','LK729','41265','bring gold,spicies');
200
201
202  /*insert Airplane_Schedule*/
203  insert into Airplane_Schedule values('D779955331','09:28:48','15:45:47');
204  insert into Airplane_Schedule values('N987654321','01:11:18','08:15:27');
205  insert into Airplane_Schedule values('R200171223','11:31:48','19:45:47');
206  insert into Airplane_Schedule values('Q224466889','13:07:09','23:55:47');
207
208
209  /*insert Flight_Schedule*/
210  insert into Flight_Schedule values('AL987','08.07.09','14:58:14');
211  insert into Flight_Schedule values('TY456','00:14:15','07:13:17');
212  insert into Flight_Schedule values('KL203','10:27:26','18:08:16');
213  insert into Flight_Schedule values('QE333','12:48:17','00:05:05');
214  insert into Flight_Schedule values('BH912','07:22:26','14:08:16');
215  insert into Flight_Schedule values('PR914','17:48:17','04:05:05');

```

```

218
219  select *
220  from Airport;
221
222  select *
223  from Airplane_Type;
224
225  select *
226  from Land;
227
228  select *
229  from Airplane;
230
231  select *
232  from Flight;
233
234  select *
235  from Flight_leg;

```

100 %

Results				
	Airportcode	Name	City	State
1	A020171	Sydney	San Mateo	California
2	B020221	Frank...	Frankfurt	Hesse
3	C020271	Sing...	Salt Lake...	Utah
4	D020251	Helsi...	Ivalo	Kemi-T...
5	E020230	Brisb...	Brisbane	Victoria
6	F171223	Band...	katunaya...	Colombo

	TypeName	MaxNumOfSeats	Company
1	Airbus A330	600	Pratt & Whitney
2	Airlines 787	300	Rolls Royce
3	Antonov A...	350	LET Aircraft I...
4	Boeing 747	400	Pratt & Whitney
5	Boeing 757	500	Miami-based...
6	Douglas 80	200	Pratt & Whin...

	Airportcode	TypeName
1	A020171	Airlines 787

## ii. SQL codes as a separate script

--Creating Airport Table--

```
Create table Airport(  
    Airportcode varchar(10) NOT NULL,  
    Name varchar(50) NOT NULL,  
    City varchar(50) NOT NULL,  
    State varchar(50) NOT NULL,  
    constraint Airport_pk primary key (Airportcode),  
    CONSTRAINT checkAirport_Airportcode check (Airportcode LIKE '[A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')  
);
```

--Creating Airplane\_Type Table--

```
Create table Airplane_Type(  
    TypeName varchar(50) NOT NULL,  
    MaxNumOfSeats varchar(50) NOT NULL,  
    Company varchar(50) NOT NULL,  
    constraint Airplane_Type_pk primary key (TypeName),  
);
```

--Creating Airplane\_Type Table--

```
Create table Land(  
    Airportcode varchar(10) NOT NULL,  
    TypeName varchar(50) NOT NULL,  
    constraint Land_pk primary key (Airportcode,TypeName),  
    constraint Land_fk1 foreign key (Airportcode) REFERENCES Airport (Airportcode),  
    constraint Land_fk2 foreign key (TypeName) REFERENCES Airplane_Type (TypeName),  
);
```

--Creating Airplane Table--

```
Create table Airplane(  
    AirplaneID varchar(10) NOT NULL,  
    AirName varchar(50) NOT NULL,  
    TotalNumOfSeats integer,  
    TypeName varchar(50) NOT NULL,  
    constraint airplane_pk primary key (AirplaneID),  
    constraint airplane_fk foreign key (TypeName) REFERENCES Airplane_Type  
(TypeName),  
    constraint checkAirplane_AirplaneID check (AirplaneID LIKE '[A-Z][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')  
);
```

--Creating Flight Table--

```
Create table Flight(  
    FlightNo varchar(50) NOT NULL,  
    AirlineName varchar(50) NOT NULL,  
    ScheduleDate varchar(50) NOT NULL,  
    constraint Flight_pk primary key (FlightNo),  
    constraint checkFlight_FlightNo check (FlightNo LIKE '[A-Z][A-Z][0-9][0-9][0-9][0-9][0-9][0-9]')  
);
```

--Creating Flight\_Leg Table--

```
Create table Flight_Leg(  
    LegNo varchar(10) NOT NULL,  
    FlightNo varchar(50) NOT NULL,  
    Airportcode varchar(10) NOT NULL,  
    constraint Flight_Leg_pk primary key (LegNo,FlightNo),  
    constraint Flight_Leg_fk1 foreign key (Airportcode) REFERENCES Airport  
(Airportcode),  
    constraint Flight_Leg_fk2 foreign key (FlightNo) REFERENCES Flight (FlightNo)  
);
```

--Creating Leg\_instance Table--

```
Create table Leg_instance(  
    Date date NOT NULL,  
    LegNo varchar(10) NOT NULL,  
    FlightNo varchar(50) NOT NULL,  
    Ava_Seats varchar(50) NOT NULL,  
    AirplaneID varchar(10) NOT NULL,  
    Airportcode varchar(10) NOT NULL,  
    constraint Leg_instance_pk primary key (Date,LegNo,FlightNo),  
    constraint Leg_instance_fk1 foreign key (LegNo,FlightNo) REFERENCES Flight_Leg  
(LegNo,FlightNo),  
    constraint Leg_instance_fk2 foreign key (AirplaneID) REFERENCES Airplane  
(AirplaneID),  
    constraint Flight_Leg_fk3 foreign key (Airportcode) REFERENCES Airport  
(Airportcode),  
);
```

--Creating Seat Table--

```
Create table Seat(  
    Seat_No varchar(10) NOT NULL,  
    Date date NOT NULL,  
    LegNo varchar(10) NOT NULL,  
    FlightNo varchar(50) NOT NULL,  
    CusName varchar(50) NOT NULL,  
    CusPhoneNo varchar(50) NOT NULL,  
    constraint Seat_pk primary key (Seat_No,Date,LegNo,FlightNo),  
    constraint check_Seat_CusPhoneNo check (CusPhoneNo LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),  
    constraint Seat_fk foreign key (Date,LegNo,FlightNo) REFERENCES Leg_instance  
(Date,LegNo,FlightNo)  
);
```

--Creating Flight\_fare Table--

```
Create table Flight_fare(  
    Farecode varchar(20) NOT NULL,  
    FlightNo varchar(50) NOT NULL,  
    Amount varchar(20) NOT NULL,  
    Restriction varchar(50) NOT NULL,  
    constraint Flight_fare_pk primary key (Farecode,FlightNo),  
    constraint Flight_fare_fk foreign key (FlightNo) REFERENCES Flight (FlightNo)  
);
```

--Creating Airplane\_Schedule Table--

```
CREATE TABLE Airplane_Schedule (  
    AirplaneID varchar (10),  
    arr_time time NOT NULL,  
    dep_time time NOT NULL,  
    constraint Airplane_Schedule_PK primary key (AirplaneID),  
    constraint Airplane_Schedule_fk foreign key (AirplaneID) REFERENCES Airplane  
(AirplaneID)  
);
```

--Creating Flight\_Schedule Table--

```
CREATE TABLE Flight_Schedule (  
    FlightNo varchar (50) NOT NULL,  
    schedule_arr_time time(0) NOT NULL,  
    schedule_dep_time time(0) NOT NULL,  
    constraint Flight_schedule_PK primary key (FlightNo),  
    constraint Flight_Schedule_fk foreign key (FlightNo) REFERENCES Flight  
(FlightNo)  
);
```

/\*insert airport table\*/

```
insert into Airport values('A020171','Sydney','San Mateo','California');  
insert into Airport values('B020221','Frankfurt Airport','Frankfurt','Hesse');  
insert into Airport values('C020271','Singapore','Salt Lake City','Utah');  
insert into Airport values('D020251','Helsinki Airport','Ivalo','Kemi-Tornio');  
insert into Airport values('E020230','Brisbane International  
Airport','Brisbane','Victoria');  
insert into Airport values('F171223','Bandaranaike International  
Airport','katunayaka','Colombo');
```

/\*insert Airplane\_Type\*/

```
insert into Airplane_Type values('Airlines 787','300','Rolls Royce');  
insert into Airplane_Type values('Airbus A330','600','Pratt & Whitney');  
insert into Airplane_Type values('Boeing 747','400','Pratt & Whitney');  
insert into Airplane_Type values('Boeing 757','500','Miami-based Eastern Airlines');  
insert into Airplane_Type values('Douglas 80','200','Pratt & Whitney JT8-D');  
insert into Airplane_Type values('Antonov AN28','350','LET Aircraft Industries');
```

/\*insert Land\*/

```
insert into Land values('A020171','Airlines 787');  
insert into Land values('B020221','Airbus A330');  
insert into Land values('C020271','Boeing 747');  
insert into Land values('D020251','Boeing 757');  
insert into Land values('E020230','Douglas 80');  
insert into Land values('F171223','Antonov AN28');
```

/\*insert Airplane\*/

```
insert into Airplane values('R200171223','Douglas DC-3',295,'Boeing 757');  
insert into Airplane values('N987654321','Airbus A321XLR ',467,'Boeing 747');  
insert into Airplane values('Q224466889','COMAC C919',440,'Airbus A330');  
insert into Airplane values('D779955331','Universal Hydrogen ATR 72',174,'Antonov  
AN28');
```

```

/*insert Flight*/
insert into Flight values('AL987','Air Berlin','2022-12-23');
insert into Flight values('TY456','Belair','2022-10-19');
insert into Flight values('KL203','Paramount','2022-11-04');
insert into Flight values('QE333','Oman Air','2022-10-26');
insert into Flight values('BH912','IndiGo','2023-01-02');
insert into Flight values('LK729','Jetstar Asia','2022-10-23');
insert into Flight values('PR914','SL Airways','2022-12-01');

/*insert Flight_leg*/
insert into Flight_leg values('2','AL987','A020171');
insert into Flight_leg values('3','PR914','B020221');
insert into Flight_leg values('4','AL987','C020271');
insert into Flight_leg values('1','QE333','D020251');
insert into Flight_leg values('2','LK729','E020230');
insert into Flight_leg values('3','BH912','F171223');

/*insert Leg_instance*/
insert into Leg_instance values('2022-11-14','2','AL987','74','R200171223','A020171');
insert into Leg_instance values('2022-12-23','3','PR914','52','N987654321','B020221');
insert into Leg_instance values('2023-01-01','2','LK729','158','D779955331','E020230');
insert into Leg_instance values('2022-12-14','1','QE333','74','Q224466889','D020251');

/*insert seat*/
insert into seat values('11F','2022-11-14','2','AL987','Kaun Perera','0859632147');
insert into seat values('55JH','2022-12-23','3','PR914','Niki watson','0856329741');
insert into seat values('14LA','2023-01-01','2','LK729','Mary Ann','1253698521');
insert into seat values('13HA','2022-12-14','1','QE333','Sadun perera','9476966518');

/*insert Flight_fare*/
insert into Flight_fare values('AF1015','AL987','650000','plastic');
insert into Flight_fare values('CDIJ17','TY456','275000','bring metal');
insert into Flight_fare values('GNQS03','KL203','74000','bring pets and plants');
insert into Flight_fare values('PW4589','QE333','65000','eat fish and meats');
insert into Flight_fare values('YBKL1','PR914','54000','bring over 50KG travel bags');
insert into Flight_fare values('NQS144','BH912','41000','bring pets');
insert into Flight_fare values('NOQS51','LK729','41265','bring gold,spicies');

/*insert Airplane_Schedule*/
insert into Airplane_Schedule values('D779955331','09:28:48','15:45:47');
insert into Airplane_Schedule values('N987654321','01:11:18','08:15:27');
insert into Airplane_Schedule values('R200171223','11:31:48','19:45:47');
insert into Airplane_Schedule values('Q224466889','13:07:09','23:55:47');

/*insert Flight_Schedule*/
insert into Flight_Schedule values('AL987','08.07.09','14:58:14');
insert into Flight_Schedule values('TY456','00:14:15','07:13:17');
insert into Flight_Schedule values('KL203','10:27:26','18:08:16');
insert into Flight_Schedule values('QE333','12:48:17','00:05:05');
insert into Flight_Schedule values('BH912','07:22:26','14:08:16');
insert into Flight_Schedule values('PR914','17:48:17','04:05:05');

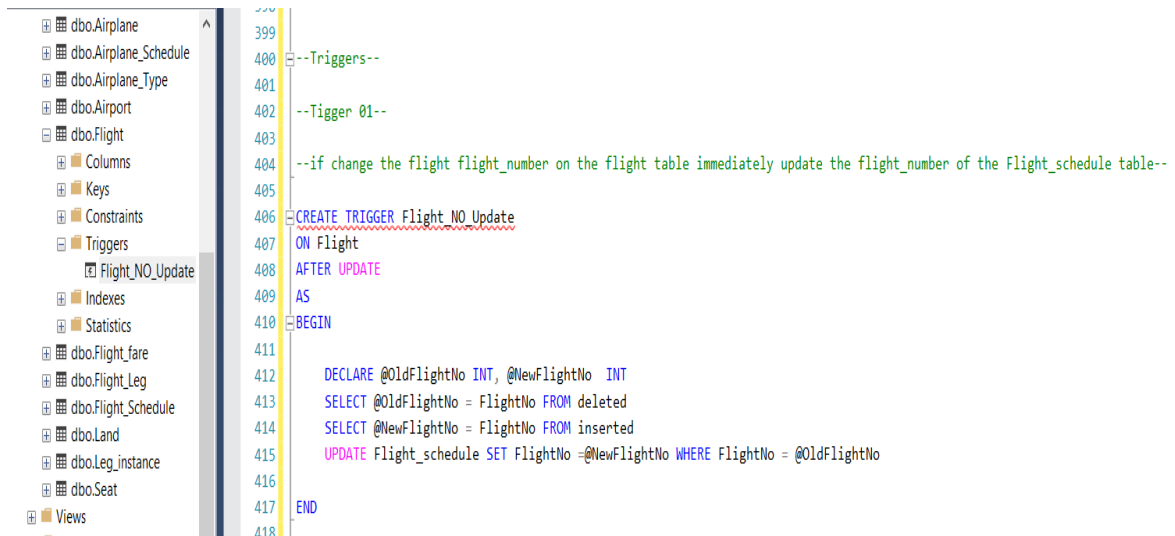
```



### iii. Triggers

#### i. Trigger For Flight Schedule Table

If change the flight number on the flight table immediately update the flight number on the Flight schedule table



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Triggers' folder is expanded under the 'dbo' schema, showing a trigger named 'Flight\_NO\_Update'. On the right, the SQL script for creating this trigger is displayed. The script includes comments and the following T-SQL code:

```
399
400 --Triggers--
401
402 --Tigger 01--
403
404 --if change the flight flight_number on the flight table immediately update the flight_number of the Flight_schedule table--
405
406 CREATE TRIGGER Flight_NO_Update
407 ON Flight
408 AFTER UPDATE
409 AS
410 BEGIN
411
412     DECLARE @OldFlightNo INT, @NewFlightNo INT
413     SELECT @OldFlightNo = FlightNo FROM deleted
414     SELECT @NewFlightNo = FlightNo FROM inserted
415     UPDATE Flight_schedule SET FlightNo = @NewFlightNo WHERE FlightNo = @OldFlightNo
416
417 END
418
```

--Tigger 01--

--if change the flight\_number on the flight table immediately update the flight\_number of the Flight\_schedule table--

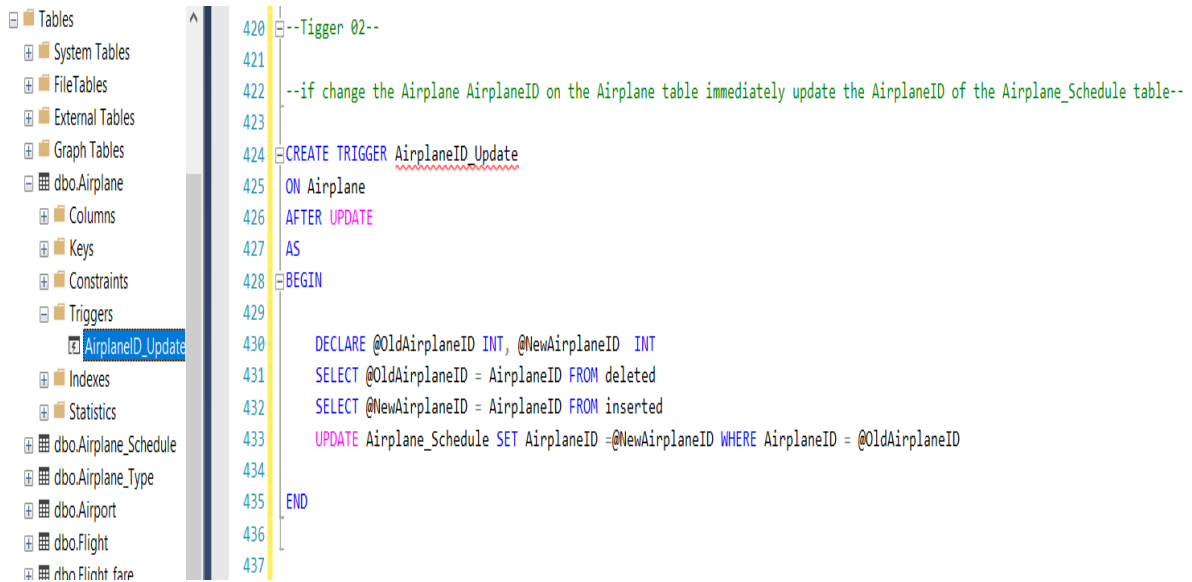
```
CREATE TRIGGER Flight_NO_Update
ON Flight
AFTER UPDATE
AS
BEGIN
```

```
    DECLARE @OldFlightNo INT, @NewFlightNo INT
    SELECT @OldFlightNo = FlightNo FROM deleted
    SELECT @NewFlightNo = FlightNo FROM inserted
    UPDATE Flight_schedule SET FlightNo = @NewFlightNo WHERE FlightNo = @OldFlightNo
```

```
END
```

ii. Trigger For Airplane Schedule table

If change the Airplane ID on the Airplane table immediately update the Airplane ID of the Airplane Schedule table



The screenshot shows the SQL Server Enterprise Manager interface on the left, with the 'Triggers' folder expanded under 'dbo'. The 'AirplaneID\_Update' trigger is selected. On the right, a SQL script is displayed with line numbers 420 to 437. The script is as follows:

```
420 --Tigger 02--
421
422 --if change the Airplane AirplaneID on the Airplane table immediately update the AirplaneID of the Airplane_Schedule table--
423
424 CREATE TRIGGER AirplaneID Update
425 ON Airplane
426 AFTER UPDATE
427 AS
428 BEGIN
429
430 DECLARE @OldAirplaneID INT, @NewAirplaneID INT
431 SELECT @OldAirplaneID = AirplaneID FROM deleted
432 SELECT @NewAirplaneID = AirplaneID FROM inserted
433 UPDATE Airplane_Schedule SET AirplaneID = @NewAirplaneID WHERE AirplaneID = @OldAirplaneID
434
435 END
436
437
```

--Tigger 02--

--if change the AirplaneID on the Airplane table immediately update the AirplaneID of the Airplane\_Schedule table--

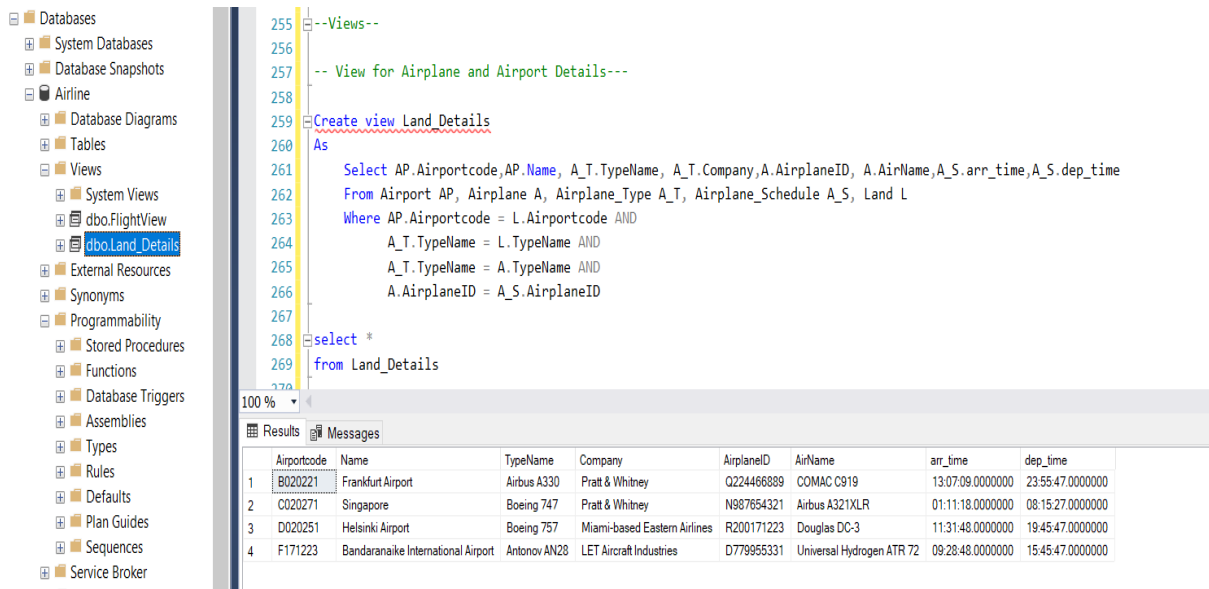
```
CREATE TRIGGER AirplaneID_Update
ON Airplane
AFTER UPDATE
AS
BEGIN
```

```
    DECLARE @OldAirplaneID INT, @NewAirplaneID INT
    SELECT @OldAirplaneID = AirplaneID FROM deleted
    SELECT @NewAirplaneID = AirplaneID FROM inserted
    UPDATE Airplane_Schedule SET AirplaneID = @NewAirplaneID WHERE AirplaneID =
    @OldAirplaneID
```

```
END
```

## iv. Views

### i. View for Airplane and Airport Details



```
255 --Views--
256
257 -- View for Airplane and Airport Details---
258
259 Create view Land_Details
260 As
261 Select AP.Airportcode,AP.Name, A_T.TypeName, A_T.Company,A.AirplaneID, A.AirName,A_S.arr_time,A_S.dep_time
262 From Airport AP, Airplane A, Airplane_Type A_T, Airplane_Schedule A_S, Land L
263 Where AP.Airportcode = L.Airportcode AND
264       A_T.TypeName = L.TypeName AND
265       A_T.TypeName = A.TypeName AND
266       A.AirplaneID = A_S.AirplaneID
267
268 select *
269 from Land_Details
```

	Airportcode	Name	TypeName	Company	AirplaneID	AirName	arr_time	dep_time
1	B020221	Frankfurt Airport	Airbus A330	Pratt & Whitney	Q224466889	COMAC C919	13:07:09.0000000	23:55:47.0000000
2	C020271	Singapore	Boeing 747	Pratt & Whitney	N987654321	Airbus A321XLR	01:11:18.0000000	08:15:27.0000000
3	D020251	Helsinki Airport	Boeing 757	Miami-based Eastern Airlines	R200171223	Douglas DC-3	11:31:48.0000000	19:45:47.0000000
4	F171223	Bandaraneike International Airport	Antonov AN28	LET Aircraft Industries	D779955331	Universal Hydrogen ATR 72	09:28:48.0000000	15:45:47.0000000

-- View for Airplane and Airport Details---

Create view Land\_Details

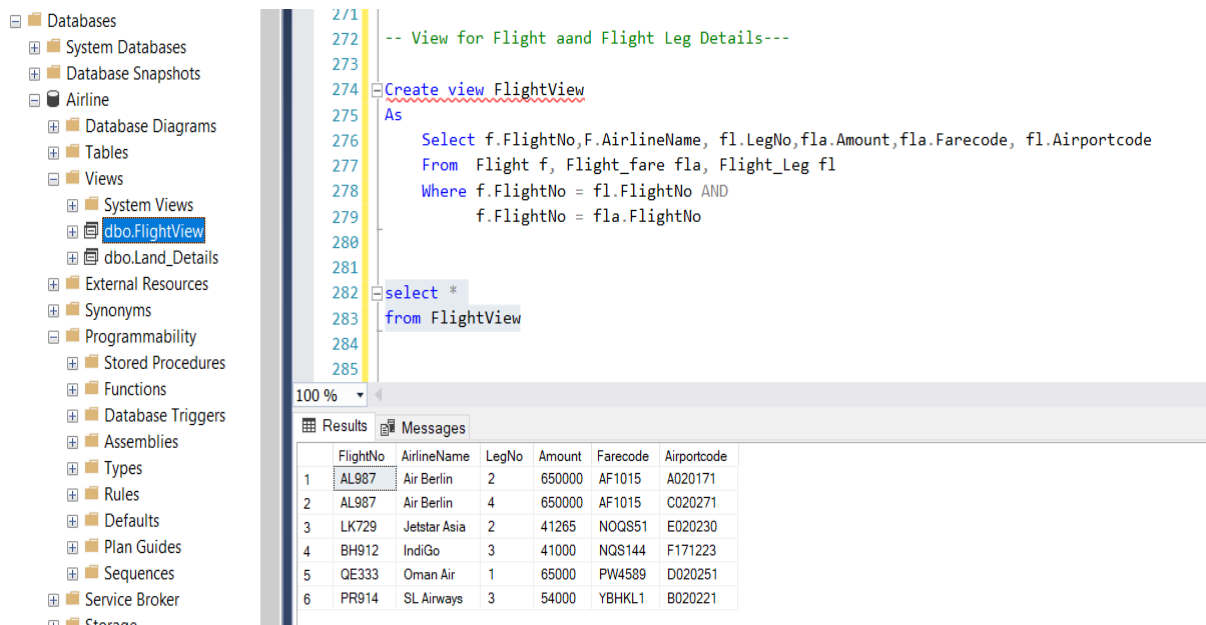
As

```
Select AP.Airportcode,AP.Name, A_T.TypeName, A_T.Company,A.AirplaneID,
A.AirName,A_S.arr_time,A_S.dep_time
From Airport AP, Airplane A, Airplane_Type A_T, Airplane_Schedule A_S, Land L
Where AP.Airportcode = L.Airportcode AND
      A_T.TypeName = L.TypeName AND
      A_T.TypeName = A.TypeName AND
      A.AirplaneID = A_S.AirplaneID
```

select \*

from Land\_Details

## ii. View for Flight and Flight Leg Details



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Databases' folder is expanded, showing 'Airline' > 'Views' > 'dbo.FlightView'. The central pane displays the following SQL script:

```
-- View for Flight aand Flight Leg Details---  
  
Create view FlightView  
As  
    Select f.FlightNo,F.AirlineName, fl.LegNo,fla.Amount,fla.Farecode, fl.Airportcode  
    From Flight f, Flight_fare fla, Flight_Leg fl  
    Where f.FlightNo = fl.FlightNo AND  
          f.FlightNo = fla.FlightNo  
  
select *  
from FlightView
```

Below the script, the 'Results' tab shows a table with 6 rows and 6 columns:

	FlightNo	AirlineName	LegNo	Amount	Farecode	Airportcode
1	AL987	Air Berlin	2	650000	AF1015	A020171
2	AL987	Air Berlin	4	650000	AF1015	C020271
3	LK729	Jetstar Asia	2	41265	NOQS51	E020230
4	BH912	IndiGo	3	41000	NQS144	F171223
5	QE333	Oman Air	1	65000	PW4589	D020251
6	PR914	SL Airways	3	54000	YBHKL1	B020221

```
-- View for Flight and Flight Leg Details---
```

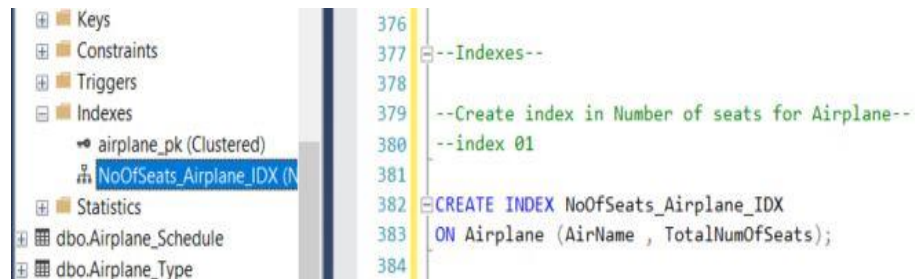
```
Create view FlightView  
As
```

```
    Select f.FlightNo,F.AirlineName, fl.LegNo,fla.Amount,fla.Farecode,  
    fl.Airportcode  
    From Flight f, Flight_fare fla, Flight_Leg fl  
    Where f.FlightNo = fl.FlightNo AND  
          f.FlightNo = fla.FlightNo
```

```
select *  
from FlightView
```

## v. Indexes

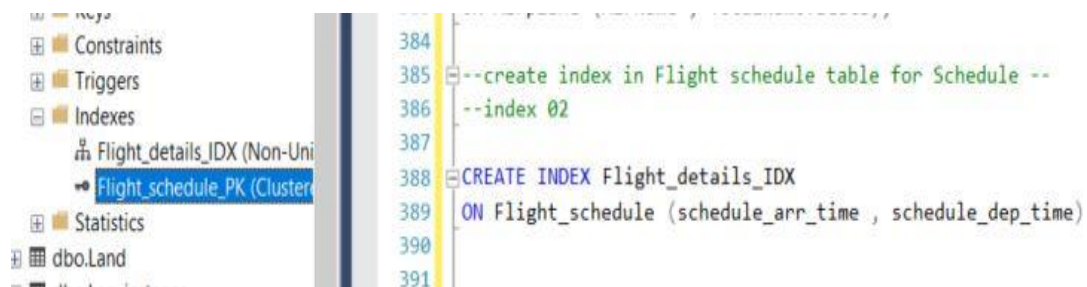
- i. Create an index in the Number of seats for an Airplane



```
--Create an index in the Number of seats for Airplane--  
--index 01--
```

```
CREATE INDEX NoOfSeats_Airplane_IDX  
ON Airplane (AirName , TotalNumOfSeats);
```

- ii. Create index in Flight schedule table for the Schedule



```
--create index in Flight schedule table for Schedule --  
--index 02--
```

```
CREATE INDEX Flight_details_IDX  
ON Flight_schedule (schedule_arr_time , schedule_dep_time)
```

### iii. Stored Procedures

1.

The screenshot shows the SQL Server Enterprise Manager (left pane) and the SQL Server Enterprise Query Editor (right pane). The left pane displays the database structure, including the 'Airline' database and its 'Stored Procedures' folder. The right pane shows the SQL code for creating a stored procedure named 'QS1'.

```
292
293 -- procedure Number 01 --
294
295 Create Procedure QS1 (@Airport varchar(6) , @leg varchar(20) output )
296 AS
297 begin
298     Select @leg = FL.LegNo
299     From Flight_Leg FL, Airport A
300     Where FL.Airportcode = A.Airportcode AND
301           A.Name = @Airport
302 End
303
304 Declare @LegN0 varchar(20)
305
306 Exec QS1 'Sydney' , @LegN0 output
307
```

Below the SQL code, the 'Messages' pane shows the execution results:

```
100 %
Messages
Leg No : 2
Completion time: 2022-10-25T11:21:59.3584154+05:30
```

```
-- procedure Number 01 --

Create Procedure QS1 (@Airport varchar(6) , @leg varchar(20) output )
AS
begin
    Select @leg = FL.LegNo
    From Flight_Leg FL, Airport A
    Where FL.Airportcode = A.Airportcode AND
          A.Name = @Airport
End

Declare @LegN0 varchar(20)

Exec QS1 'Sydney' , @LegN0 output

Print 'Leg No : ' + @LegN0
```

2.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Databases' tree is expanded to 'Airline' > 'Programmability' > 'Stored Procedures' > 'dbo.QS2'. The main pane shows the SQL code for the procedure. The Messages pane at the bottom shows the output of the procedure execution.

```
310
311 -- procedure Number 02 --
312
313 Create Procedure QS2 @AirportName varchar(20) , @Air_name varchar(50) output
314 AS
315 begin
316     Select @Air_name = A.airname
317     From Airport AP , Airplane_Type A_T , Land L , Airplane A
318     Where AP.Airportcode = L.Airportcode AND
319           A_T.TypeName = L.TypeName AND
320           A_T.TypeName = A.TypeName AND
321           AP.name = @AirportName
322 End
323
324 DECLARE @A_Name varchar(50)
325
326 EXEC QS2 'Singapore' , @A_Name output
327
328 print 'Airplane Name : ' + @A_Name
```

100 %

Messages

Airplane Name : Airbus A321XLR

Completion time: 2022-10-25T11:23:03.1459569+05:30

```
-- procedure Number 02 --
```

```
Create Procedure QS2 @AirportName varchar(20) , @Air_name varchar(50) output
AS
begin
    Select @Air_name = A.airname
    From Airport AP , Airplane_Type A_T , Land L , Airplane A
    Where AP.Airportcode = L.Airportcode AND
          A_T.TypeName = L.TypeName AND
          A_T.TypeName = A.TypeName AND
          AP.name = @AirportName
End

DECLARE @A_Name varchar(50)

EXEC QS2 'Singapore' , @A_Name output

print 'Airplane Name : ' + @A_Name
```

3.

The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Airline' database is expanded, showing the 'Stored Procedures' folder. The 'dbo.QS3' procedure is selected. The main pane displays the script for 'procedure Number 03'. The script includes a CREATE PROCEDURE statement, an UPDATE statement, and an EXEC statement. The 'Results' tab is active, showing a table with 7 rows and 4 columns: Farecode, FlightNo, Amount, and Restriction.

```

-- procedure Number 03 --
CREATE PROCEDURE QS3 @FlightNO VARCHAR(20) , @increase FLOAT
AS
BEGIN
    UPDATE Flight_fare
    SET amount = amount + amount * (@increase/100)
    WHERE FlightNo = @FlightNO
END

DECLARE @F_NO VARCHAR(20)

EXEC QS3 'KL203' , 20

```

	Farecode	FlightNo	Amount	Restriction
1	AF1015	AL987	650000	plastic
2	CDU17	TY456	275000	bring metal
3	GNQS03	KL203	2.52946e+007	bring pets and plants
4	NOQS51	LK729	41265	bring gold,spices
5	NQS144	BH912	41000	bring pets
6	PW4589	QE333	65000	eat fish and meats
7	YBHKL1	PR914	54000	bring over 50KG travel bags

```
-- procedure Number 03 --
```

```
CREATE PROCEDURE QS3 @FlightNO VARCHAR(20) , @increase FLOAT
AS
```

```
    BEGIN
```

```
        UPDATE Flight_fare
```

```
        SET amount = amount + amount * (@increase/100)
```

```
        WHERE FlightNo = @FlightNO
```

```
    END
```

```
DECLARE @F_NO VARCHAR(20)
```

```
EXEC QS3 'KL203' , 20
```

```
select *
from Flight_fare
```



4.

The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Programmability' folder is expanded, showing the 'Stored Procedures' folder, which contains the 'dbo.QS4' procedure. The main window shows the code for 'procedure Number 04', which is a stored procedure named 'QS4' that takes a customer number as input and returns a flight number. The code is as follows:

```
-- procedure Number 04 --
CREATE PROCEDURE QS4 @Cus_Num VARCHAR(20) , @Flight_NO VARCHAR(20) OUTPUT
AS
BEGIN
    SELECT @Flight_NO = FL.FlightNo
    FROM Flight_leg FL , Seat S
    WHERE FL.FlightNo= S.FlightNo AND
           S.CusName = @Cus_Num
END
DECLARE @F_NO VARCHAR(20)
EXEC QS4 'Mary Ann' , @F_NO OUTPUT
PRINT 'Flight NO : ' + @F_NO
```

The Messages pane at the bottom shows the output of the procedure: 'Flight NO : LK729' and the completion time: '2022-10-25T11:27:25.7611990+05:30'.

```
-- procedure Number 04 --
```

```
CREATE PROCEDURE Find_Flight_Details @Cus_Num VARCHAR(20) , @Flight_NO VARCHAR(20)
OUTPUT
AS
    BEGIN
        SELECT @Flight_NO = FL.FlightNo
        FROM Flight_leg FL , Seat S
        WHERE FL.FlightNo= S.FlightNo AND
               S.CusName = @Cus_Num
    END

DECLARE @F_NO VARCHAR(20)

EXEC Find_Flight_Details 'Mary Ann' , @F_NO OUTPUT

PRINT 'Flight NO : ' + @F_NO
```

## **Part 02**

### **Database Vulnerabilities**

#### **i. SQL Injection (SQLi) Attack**

A vulnerability known as SQL injection (SQLi) allows an attacker to change a database and acquire potentially important information by using a piece of SQL (structured query language) code. Given that it may be applied to any web application or website that makes use of a SQL-based database, it is one of the most common and dangerous types of attack (which is most of them). It has been explored since the late 1990s, but they are still relevant today. Fortnite (2019), Cisco (2018), and Tesla(2014) are some examples of SQLi.

SQL injections can be divided into three groups based on how they access the back-end data and the degree of potential harm they can do.

- 1) In-band SQLi Since attackers share the same communication route, they can easily execute.
  - I. Error-based SQLi: The attacker's actions lead the database to output an error message. Based on the information produced by these error messages, the attacker compiles details about the database infrastructure.
  - II. Union-based SQLi: The attacker combines several select queries into a single HTTP response and utilizes the UNION SQL operator to retrieve the needed data.
- 2) Inferential SQLi (Blind SQLi): The attacker uses the server's behavior patterns to learn more about the server's architecture. The attacker cannot see the result of an attack in-band.
  - I. Time-based SQLi: Attackers send a SQL query to the database, making it wait a short while before returning a true or false answer.
  - II. Boolean SQLi: Attackers send a SQL query to the database, allowing the application to return a result that is either true or false.
- 3) Out-of-band SQLi: When attackers cannot use the same channel to launch the attack and gather information, or when a server is too sluggish or unstable to do so, they may turn to SQLi out-of-band. DNS and HTTP rely on a server's computing capacity to convey data to an attacker. [1]

## Tools associated with SQLI

- |           |                |         |
|-----------|----------------|---------|
| 1. SQLmap | 2. SQLSus      | 3. Mole |
| 4. Haviji | 5. BSQL Hacker | [2]     |

## How works SQLi

An attacker must first locate weak user inputs within the web page or web application to launch a SQL Injection attack. Such user input is used directly in a SQL query on a web page or web application that contains a SQL Injection vulnerability. The assailant can input content. SQL Injections can be used by attackers to discover other users' login credentials in the database. You can choose which data to output from the database using SQL.

- Common Vulnerable Login Query

**SELECT \* FROM users**

**WHERE login = 'abc'**

**AND password = '123'**

If it returns something, then log in.

- MS SQL Server Login Syntax

**VAR SQL = "SELECT \* FROM users**

**WHERE login = +formusr+**

**AND password = +formpwd+.**

**formusr = 'or 1=1**

**formpws = anything**

- Injecting through Strings

**SELECT \* FROM users**

**WHERE username = ' ' or 1=1**

**AND password = 'anything'**

- ✓ The argument for the string is closed.
- ✓ Subsequently, the SQL command is viewed as finished.
- ✓ In addition to string fields, there are also the date and numeric fields.

- ✓ The string contains data used to exploit systems. This series of characters make up SQL queries.
- ✓ A SQL query is sent from the program to the database.
- ✓ After a database query, including an attack, sends data back to the program
- ✓ The application sends data to the user.

### **Impacts of SQLi**

- Disclose sensitive information.
- Data integrity is compromised.
- Compromise the privacy of users.
- Give an attacker system admin and general access.
- Loss of customer trust and reputational harm.

## ➤ Countermeasures and Mitigation

Any website or web application that makes use of a SQL database, such as MySQL, Oracle, SQL Server, or another one, may be vulnerable to a SQL Injection flaw. Injections are ranked as the top danger to web application security by the OWASP organization (Open Web Application Security Project) in their list of the top 10 threats for 2017. It is difficult to stop SQL Injection vulnerabilities. Input validation and parametrized queries with prepared statements are the only effective defenses against SQL Injection attacks. The application code shouldn't ever make direct use of the input. Not just web form inputs like login forms must be sanitized by the developer; all input must be done so. Single quotes and other potentially harmful code components must be removed. On your production website, it's a good idea to disable the display of database problems. SQL Injection can be used to learn more about your database by exploiting database faults. You might not be able to patch a SQL Injection vulnerability right away if you find it, say through an Acunetix scan. For instance, the flaw could be in open-source code. In these circumstances, you can use a web application firewall to sanitize your input momentarily.

- 1) Validate Input
- 2) Prepare a query
- 3) Create a prepared statement
- 4) Bind the parameters
- 5) Execute query
- 6) Fetch the result

## **ii. Denial Of Service Attack (DOS)**

Attackers and online criminals utilize a sort of assault known as a denial-of-service (DoS) attack to take down a computer system or a network by overloading the target database with requests or traffic, rendering the system or network inaccessible to its intended users. DoS attacks can affect database management systems as well and refer to as network-based assaults. This malicious attempt aims to temporarily disable a networked system without permanently damaging it. The attacker employs specially crafted software to bombard the target machine with a torrent of data packets to tax the system's constrained resources. DoS attack costs victims' organizations the most money. [2]

### **1) Bandwidth Attacks**

Each website's hosting is given a set amount of bandwidth; if more people visit the site than allowed, the hosting is compelled to restrict the site. A website will fall offline because of the attacker refreshing it frequently and opening a lot of pages. All the network's available bandwidth will be consumed by attacks.

### **2) Protocol Attacks**

The system's resources, including the CPU and RAM, will be devoured when an assault is initiated. Protocols are required to move data across a network. The victim's system uses excessive amounts of resources because of these assaults due to a particular feature or implementation defect of a protocol installed there.

### **3) ICMP Flood Attacks**

A "ping flood" is when an attacker repeatedly sends ICMP echo requests to a victim's computer. The attacker expects the victim to transmit an ICMP "echo reply" packet in response to each ICMP request. It is most effective when the attacker has more bandwidth than the target.

### **4) UDP Flood**

Targets to overwhelm a targeted server with User Datagram Protocol (UDP) packets.

### **5) TCP SYN Flood Attacks**

An attacker can take advantage of TCP-SYN flooding to cause a server to allocate extra RAM and wait for a pending connection as a result. The sender is unable to receive the last handshake message due to TCP- SYN flooding. Other systems are unable to communicate with the target system due to its clogged buffer.

## **Impacts of DOS**

- Genuine users may not be able to find the information or take the necessary actions if they cannot access the resources.
- Businesses might be unable to complete time-sensitive tasks.
- They might have their reputation ruined.
- Customers can decide to utilize a rival.

## **➤ Countermeasures and Mitigation**

Vulnerability is hard to understand. Removing network software, and preventive and investigative processes are required for high-profile databases. Making the right decisions reduces vulnerability to a DOS assault.

### **1) Build redundancy into the infrastructure**

Split your servers across different data centers and employ a load-balancing system to distribute traffic among them.

### **2) Practice Basic Network Security**

Use security measures like anti-phishing tools and firewalls that only allow a limited amount of outside traffic to reach your system.

### **3) Deploy anti-DDoS hardware and software modules**

Network and web application firewalls are required for servers. The firewall or router can be set up to block some volumetric threats coming from the outside.

### **4) Check for security patches and keep updated.**

Updates keep safe from attacks.

## **References**

[1] "Imperva Learning Center," [Online]. Available: <https://www.imperva.com/learn/>.

[2] [Online]. Available: <https://www.serverwatch.com/>.