Sri Lanka Institute of Information Technology

Specialized in Cyber Security

Year 2, Semester 1

Group 02.01

IE2042 - Database Management Systems for Security

Group Assignment 1

Group Details

Registration No	Name	Contact
IT21170720	K.D.S.P. Jayawikrama	it21170720@my.sliit.lk
IT21176388	H.C.K. Ariyarathna	it21176388@my.sliit.lk
IT21184758	Liyanage P.P.	it21184758@my.sliit.lk
IT21180316	Samaranayake Y.C	it21180316@my.sliit.lk

Table of Contents

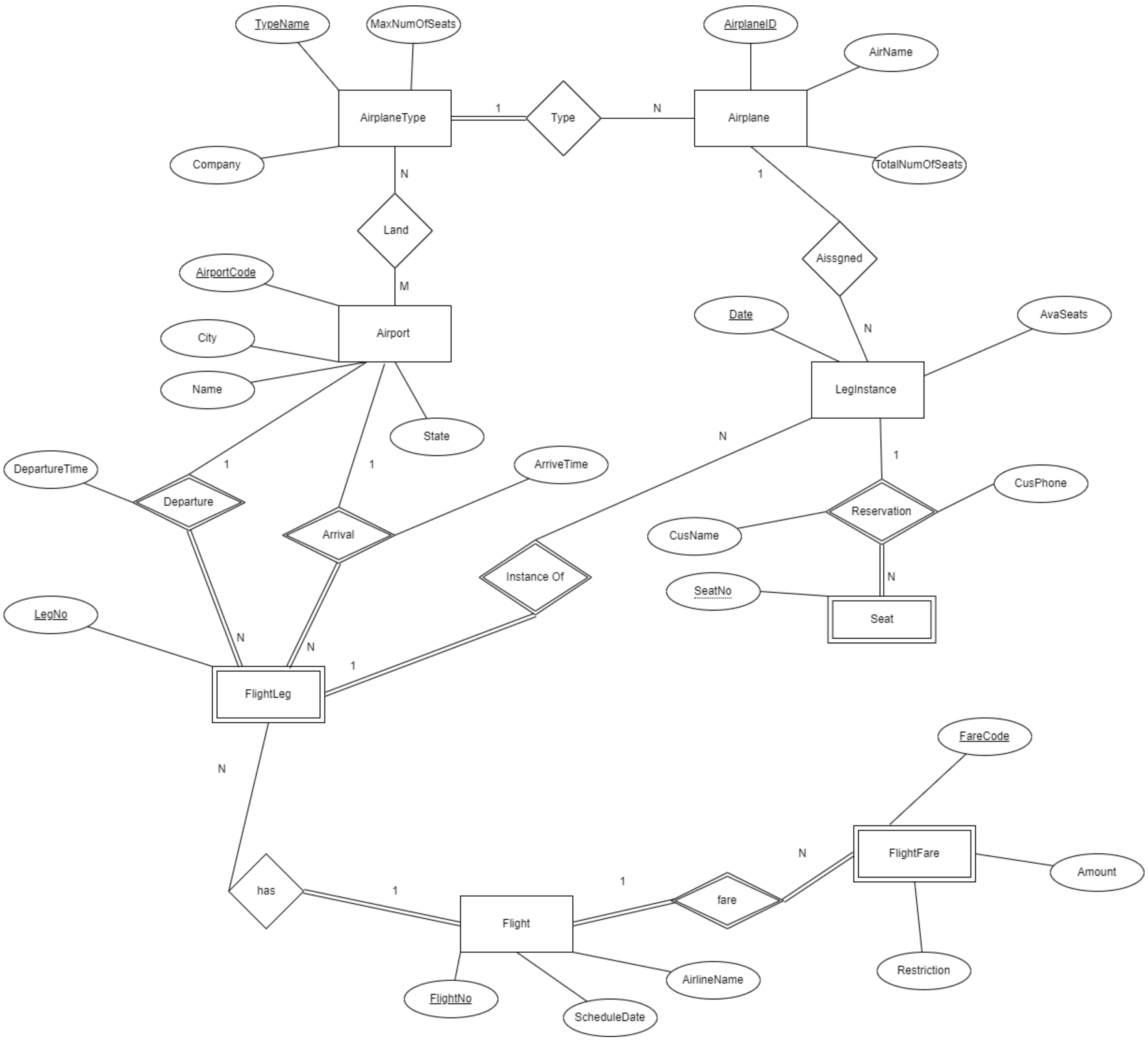
Part 1

Assumptions

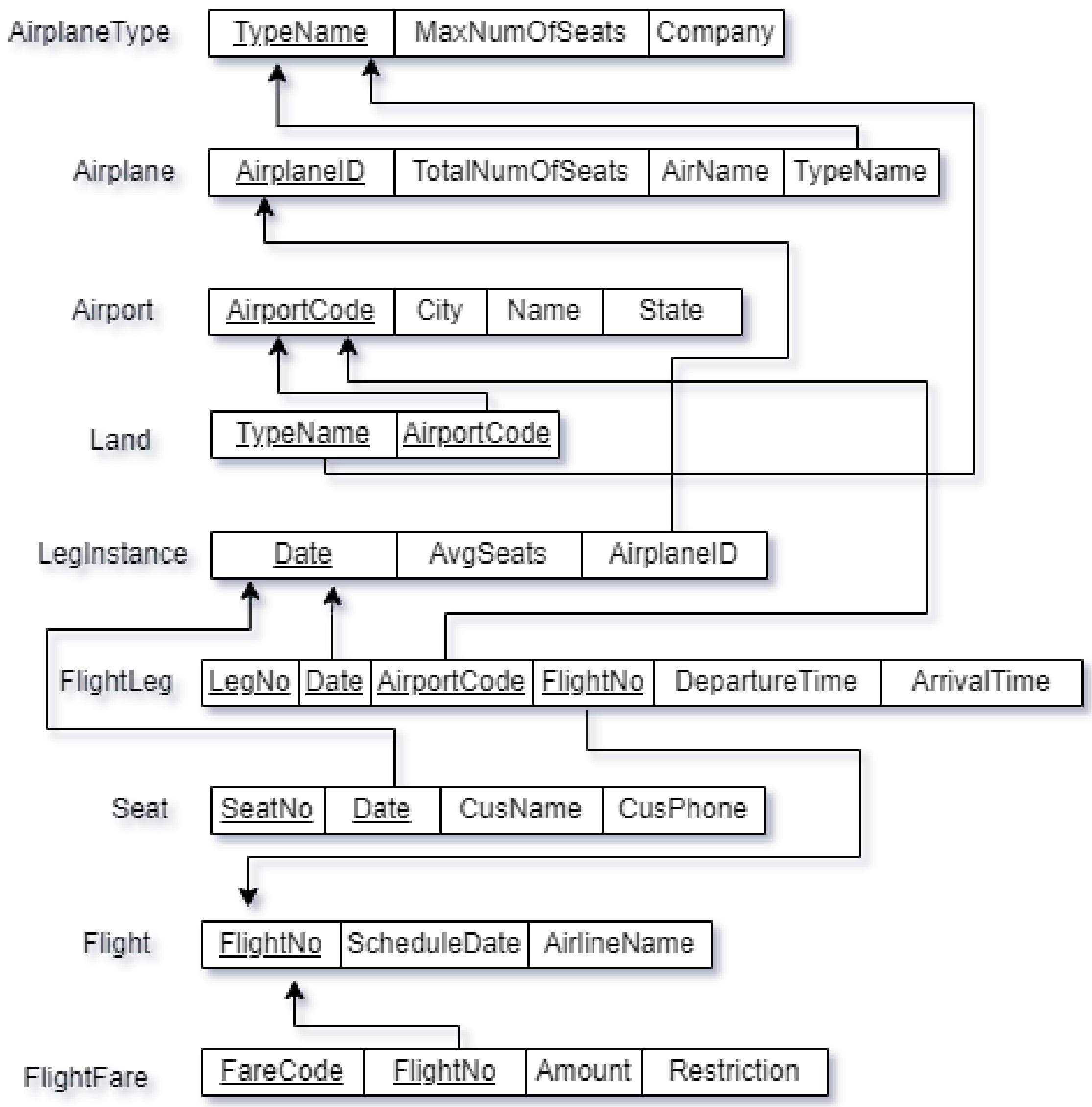
- AirplaneType table has not any repeating rows. So, this table does not violate the first normal form. This prime attribute (TypeName) does not represent any non-prime attribute. So, this table does not violate the second normal form. This non-prime attribute (MaxNumOfSeats and Company) does not represent any prime attribute. So, this table does not violate the third normal form. Because of that all the tables are in the third normal form.

- We assume that Arrival time and Departure time are separate relationship between Airport and FlightLeg entities.

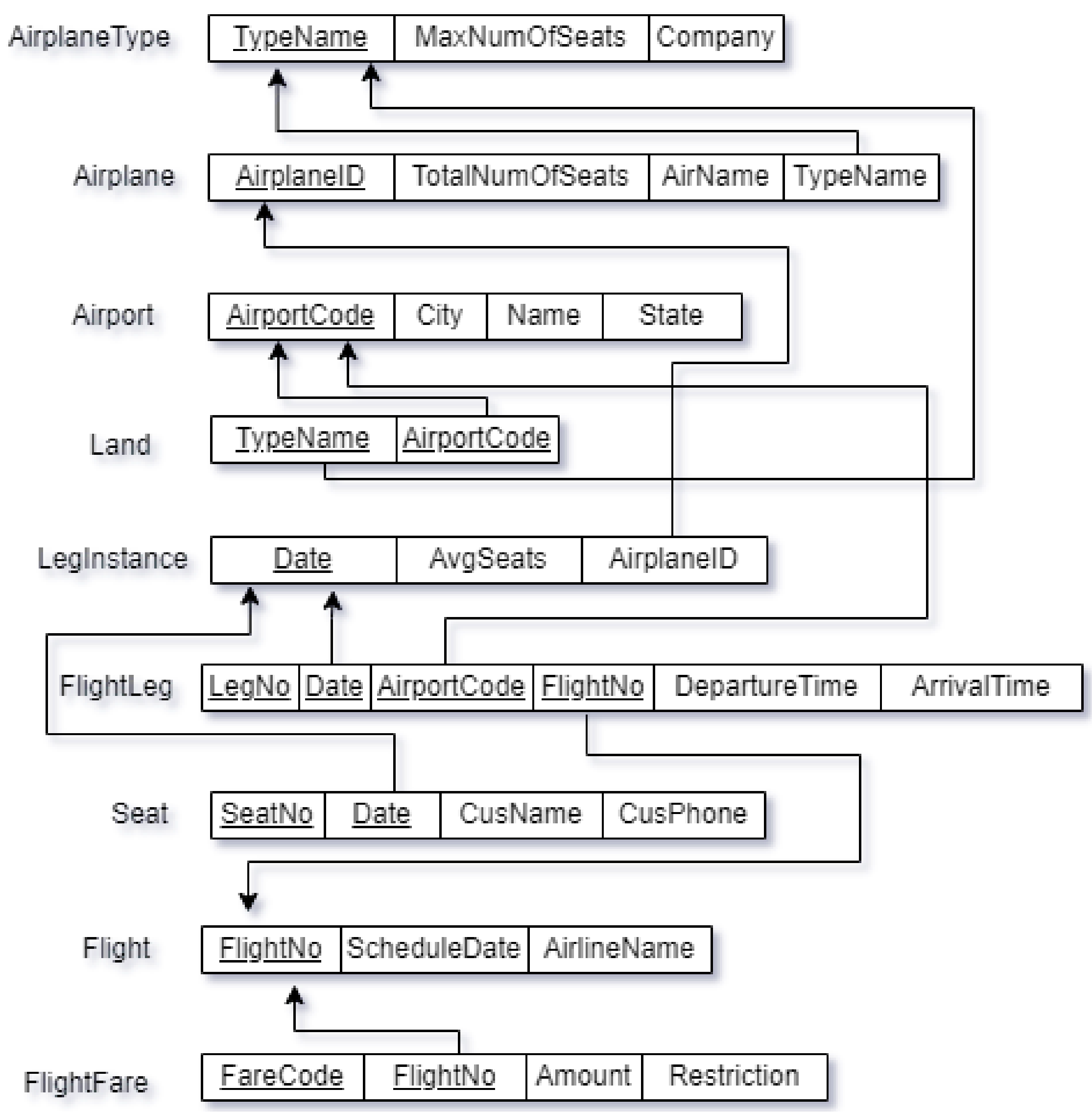
ER diagram


TypeName
MaxNumOfSeats
AirplaneID
AirName
AirplaneType
1
Type
N
Airplane
Company
TotalNumOfSeats
N
1
Land
Aissgned
AirportCode
M
Date
AvaSeats
City
Airport
N
Name
LegInstance
State
N
DepartureTime
1
1
ArriveTime
1
CusPhone
Departure
Reservation
Arrival
CusName
Instance Of
SeatNo
N
Seat
LegNo
N
N
1
FlightLeg
FareCode
N
N
FlightFare
Amount
has
1
1
fare
Flight
Restriction
AirlineName
FlightNo
ScheduleDate

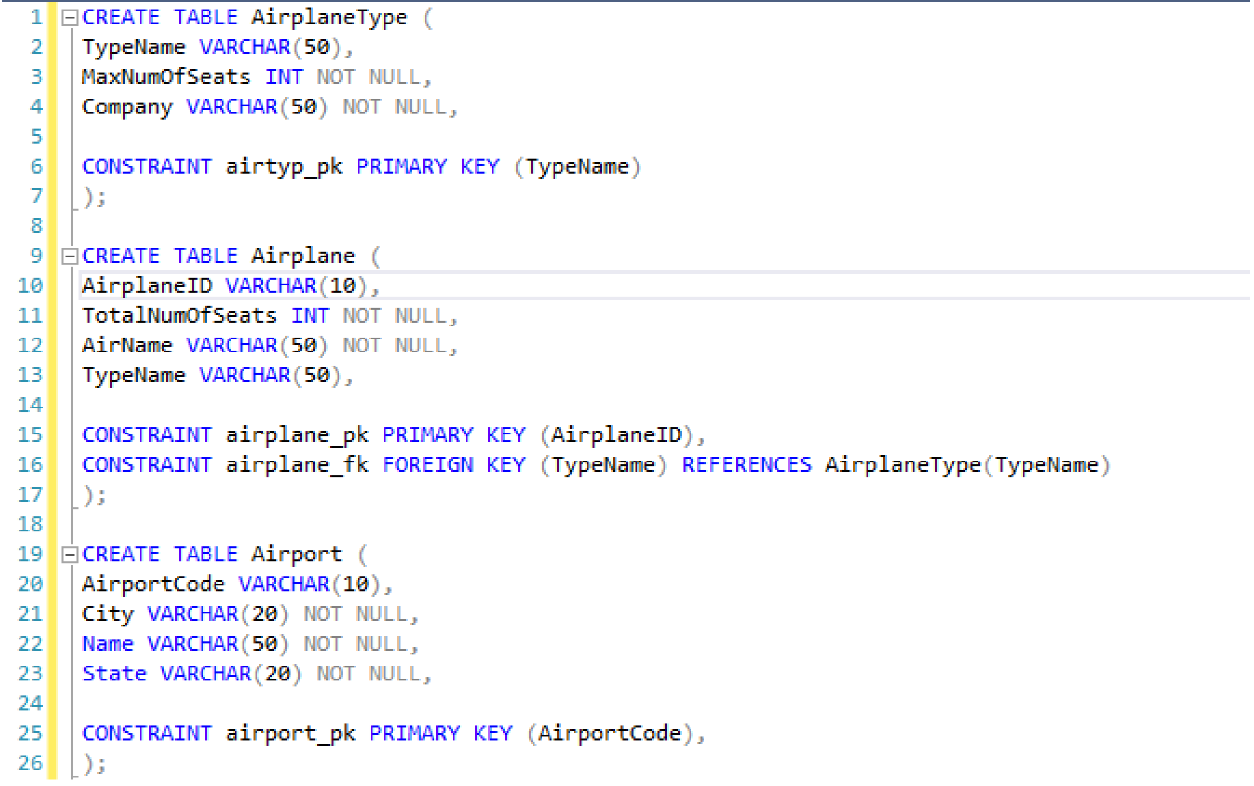
Logical Model



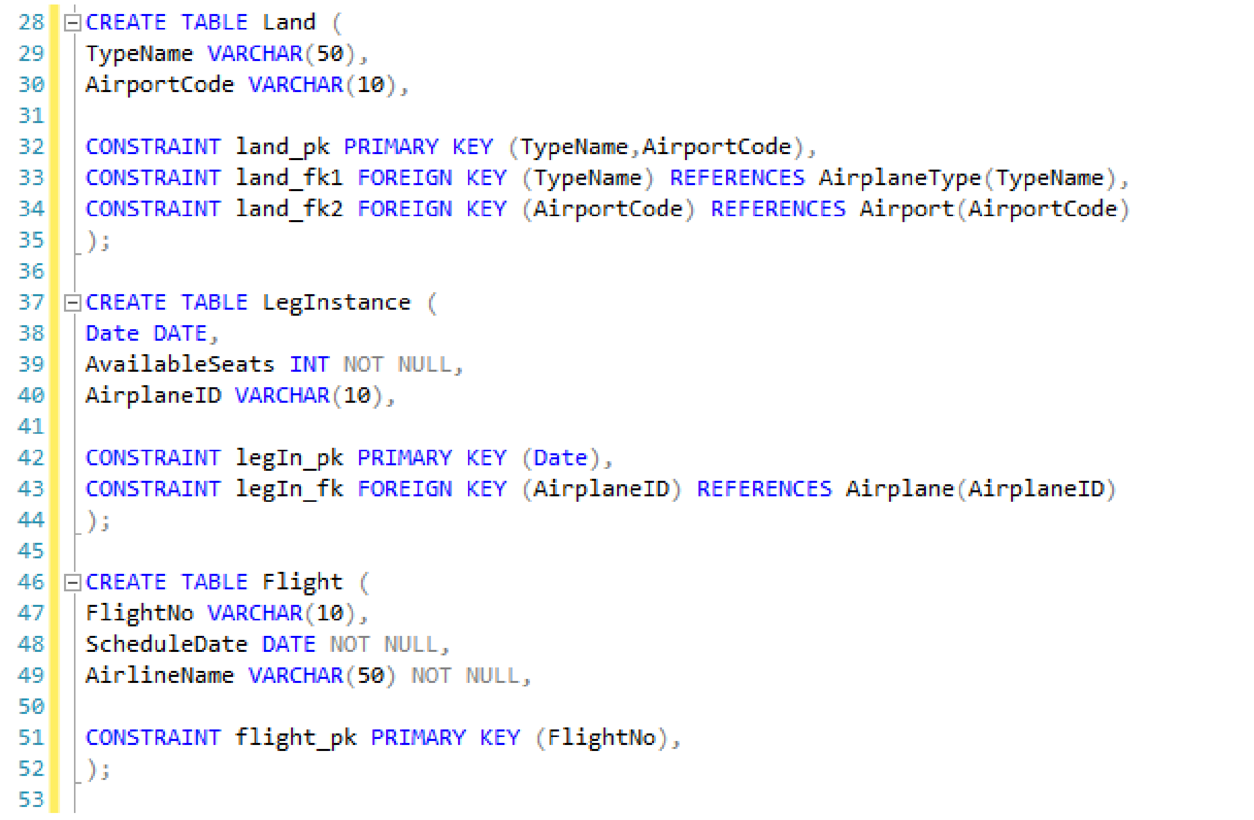
Logical Model (3RD Normalize)



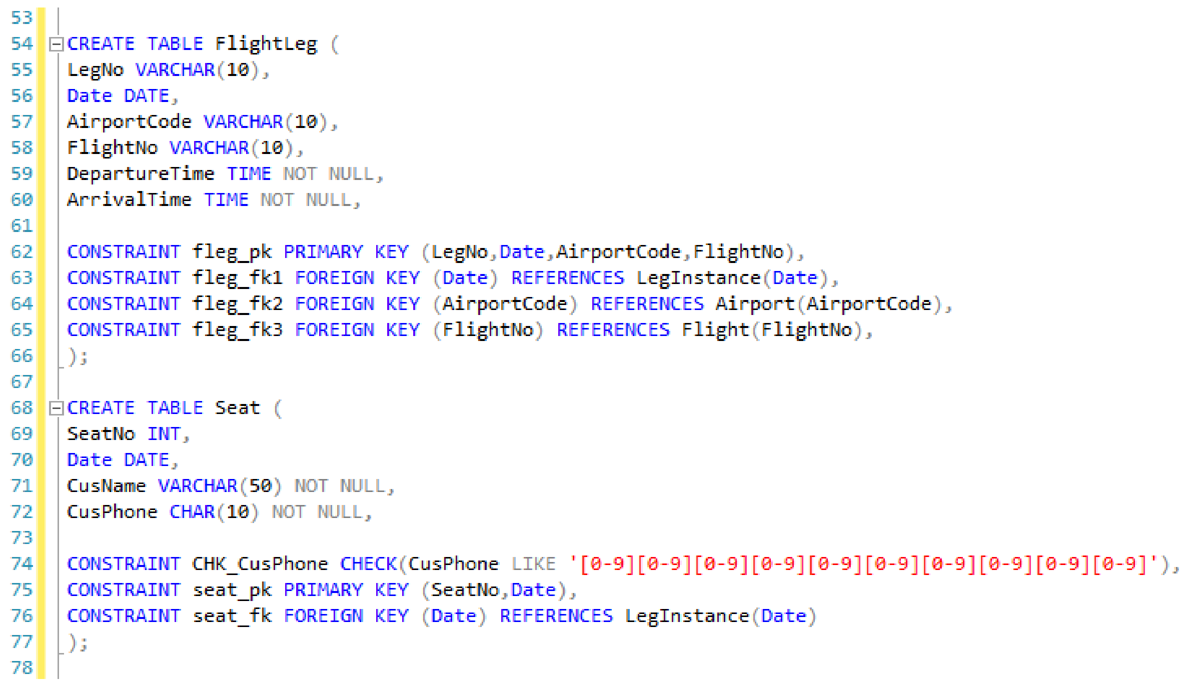
Creating Tables



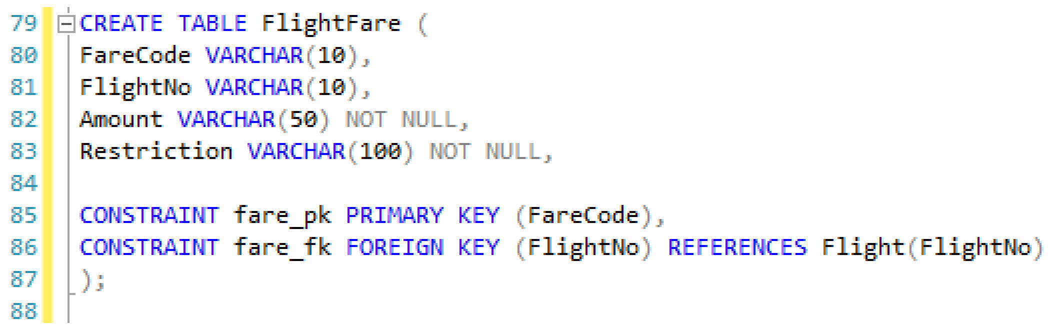
```sql
CREATE TABLE AirplaneType (
TypeName VARCHAR(50),
MaxNumOfSeats INT NOT NULL,
Company VARCHAR(50) NOT NULL,

CONSTRAINT airtyp_pk PRIMARY KEY (TypeName)
);

CREATE TABLE Airplane (
AirplaneID VARCHAR(10),
TotalNumOfSeats INT NOT NULL,
AirName VARCHAR(50) NOT NULL,
TypeName VARCHAR(50),

CONSTRAINT airplane_pk PRIMARY KEY (AirplaneID),
CONSTRAINT airplane_fk FOREIGN KEY (TypeName) REFERENCES AirplaneType(TypeName)
);

CREATE TABLE Airport (
AirportCode VARCHAR(10),
City VARCHAR(20) NOT NULL,
Name VARCHAR(50) NOT NULL,
State VARCHAR(20) NOT NULL,

CONSTRAINT airport_pk PRIMARY KEY (AirportCode),
);
```



```sql
CREATE TABLE Land (
TypeName VARCHAR(50),
AirportCode VARCHAR(10),

CONSTRAINT land_pk PRIMARY KEY (TypeName,AirportCode),
CONSTRAINT land_fk1 FOREIGN KEY (TypeName) REFERENCES AirplaneType(TypeName),
CONSTRAINT land_fk2 FOREIGN KEY (AirportCode) REFERENCES Airport(AirportCode)
);

CREATE TABLE LegInstance (
Date DATE,
AvailableSeats INT NOT NULL,
AirplaneID VARCHAR(10),

CONSTRAINT legIn_pk PRIMARY KEY (Date),
CONSTRAINT legIn_fk FOREIGN KEY (AirplaneID) REFERENCES Airplane(AirplaneID)
);

CREATE TABLE Flight (
FlightNo VARCHAR(10),
ScheduleDate DATE NOT NULL,
AirlineName VARCHAR(50) NOT NULL,

CONSTRAINT flight_pk PRIMARY KEY (FlightNo),
);
```

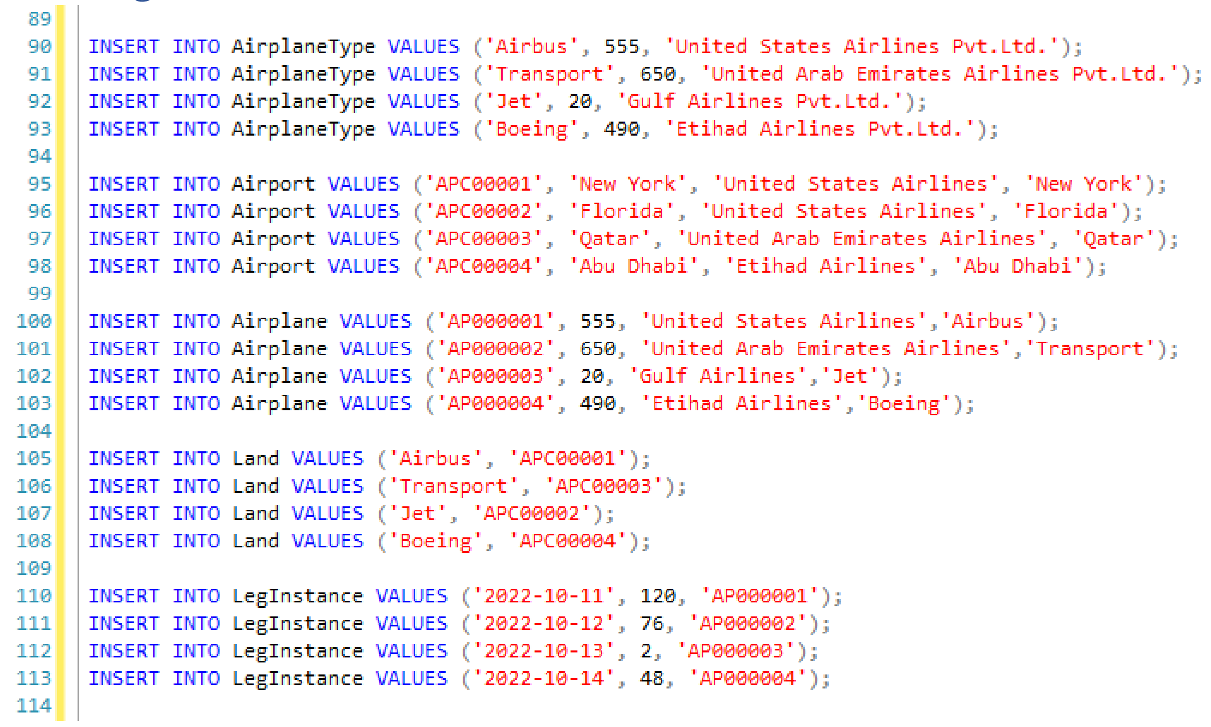


```sql

CREATE TABLE FlightLeg (
LegNo VARCHAR(10),
Date DATE,
AirportCode VARCHAR(10),
FlightNo VARCHAR(10),
DepartureTime TIME NOT NULL,
ArrivalTime TIME NOT NULL,

CONSTRAINT fleg_pk PRIMARY KEY (LegNo,Date,AirportCode,FlightNo),
CONSTRAINT fleg_fk1 FOREIGN KEY (Date) REFERENCES LegInstance(Date),
CONSTRAINT fleg_fk2 FOREIGN KEY (AirportCode) REFERENCES Airport(AirportCode),
CONSTRAINT fleg_fk3 FOREIGN KEY (FlightNo) REFERENCES Flight(FlightNo),
);

CREATE TABLE Seat (
SeatNo INT,
Date DATE,
CusName VARCHAR(50) NOT NULL,
CusPhone CHAR(10) NOT NULL,

CONSTRAINT CHK_CusPhone CHECK(CusPhone LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
CONSTRAINT seat_pk PRIMARY KEY (SeatNo,Date),
CONSTRAINT seat_fk FOREIGN KEY (Date) REFERENCES LegInstance(Date)
);

```

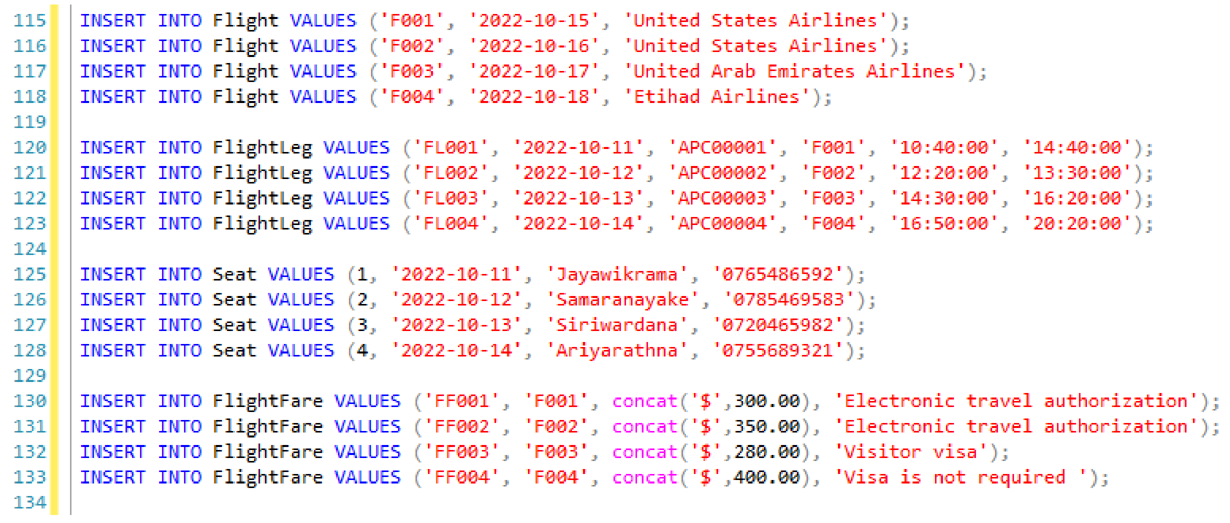


```sql
CREATE TABLE FlightFare (
FareCode VARCHAR(10),
FlightNo VARCHAR(10),
Amount VARCHAR(50) NOT NULL,
Restriction VARCHAR(100) NOT NULL,

CONSTRAINT fare_pk PRIMARY KEY (FareCode),
CONSTRAINT fare_fk FOREIGN KEY (FlightNo) REFERENCES Flight(FlightNo)
);

```

Inserting values

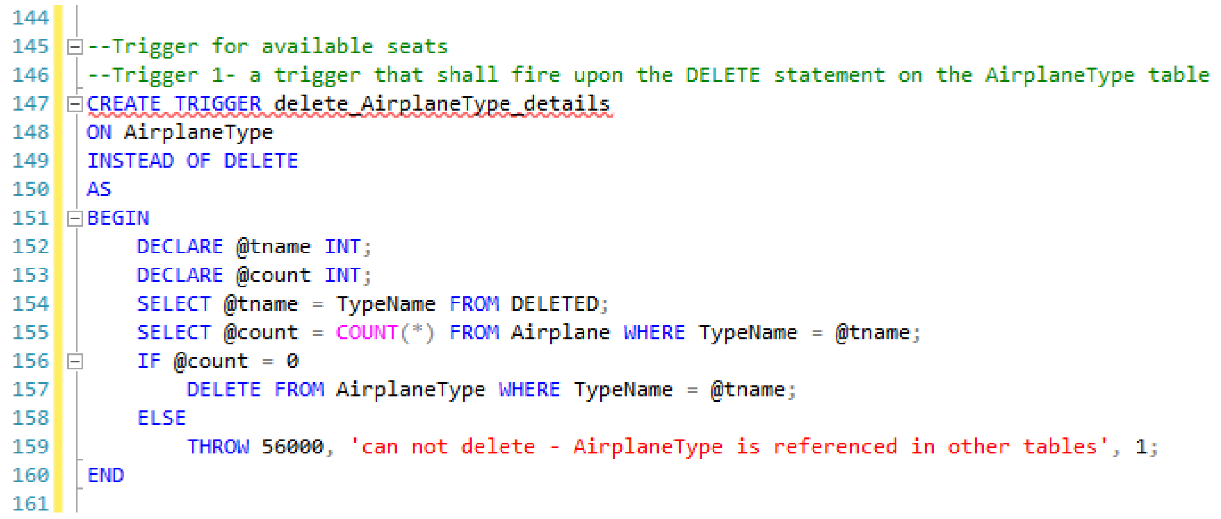


```sql

INSERT INTO AirplaneType VALUES ('Airbus', 555, 'United States Airlines Pvt.Ltd.');
INSERT INTO AirplaneType VALUES ('Transport', 650, 'United Arab Emirates Airlines Pvt.Ltd.');
INSERT INTO AirplaneType VALUES ('Jet', 20, 'Gulf Airlines Pvt.Ltd.');
INSERT INTO AirplaneType VALUES ('Boeing', 490, 'Etihad Airlines Pvt.Ltd.');

INSERT INTO Airport VALUES ('APC00001', 'New York', 'United States Airlines', 'New York');
INSERT INTO Airport VALUES ('APC00002', 'Florida', 'United States Airlines', 'Florida');
INSERT INTO Airport VALUES ('APC00003', 'Qatar', 'United Arab Emirates Airlines', 'Qatar');
INSERT INTO Airport VALUES ('APC00004', 'Abu Dhabi', 'Etihad Airlines', 'Abu Dhabi');

INSERT INTO Airplane VALUES ('AP000001', 555, 'United States Airlines','Airbus');
INSERT INTO Airplane VALUES ('AP000002', 650, 'United Arab Emirates Airlines','Transport');
INSERT INTO Airplane VALUES ('AP000003', 20, 'Gulf Airlines','Jet');
INSERT INTO Airplane VALUES ('AP000004', 490, 'Etihad Airlines','Boeing');

INSERT INTO Land VALUES ('Airbus', 'APC00001');
INSERT INTO Land VALUES ('Transport', 'APC00003');
INSERT INTO Land VALUES ('Jet', 'APC00002');
INSERT INTO Land VALUES ('Boeing', 'APC00004');

INSERT INTO LegInstance VALUES ('2022-10-11', 120, 'AP000001');
INSERT INTO LegInstance VALUES ('2022-10-12', 76, 'AP000002');
INSERT INTO LegInstance VALUES ('2022-10-13', 2, 'AP000003');
INSERT INTO LegInstance VALUES ('2022-10-14', 48, 'AP000004');

```

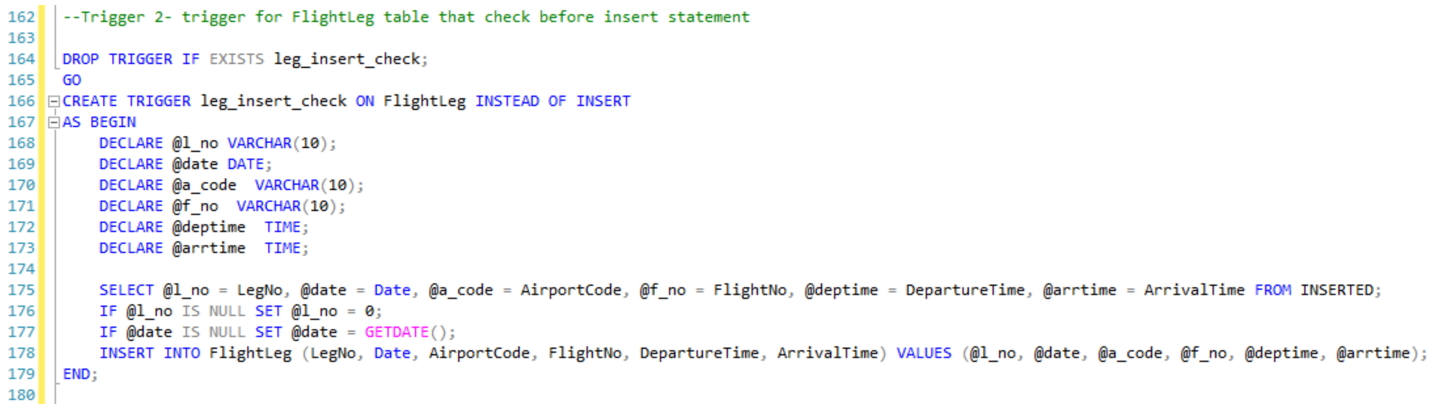


```sql
INSERT INTO Flight VALUES ('F001', '2022-10-15', 'United States Airlines');
INSERT INTO Flight VALUES ('F002', '2022-10-16', 'United States Airlines');
INSERT INTO Flight VALUES ('F003', '2022-10-17', 'United Arab Emirates Airlines');
INSERT INTO Flight VALUES ('F004', '2022-10-18', 'Etihad Airlines');

INSERT INTO FlightLeg VALUES ('FL001', '2022-10-11', 'APC00001', 'F001', '10:40:00', '14:40:00');
INSERT INTO FlightLeg VALUES ('FL002', '2022-10-12', 'APC00002', 'F002', '12:20:00', '13:30:00');
INSERT INTO FlightLeg VALUES ('FL003', '2022-10-13', 'APC00003', 'F003', '14:30:00', '16:20:00');
INSERT INTO FlightLeg VALUES ('FL004', '2022-10-14', 'APC00004', 'F004', '16:50:00', '20:20:00');

INSERT INTO Seat VALUES (1, '2022-10-11', 'Jayawikrama', '0765486592');
INSERT INTO Seat VALUES (2, '2022-10-12', 'Samaranayake', '0785469583');
INSERT INTO Seat VALUES (3, '2022-10-13', 'Siriwardana', '0720465982');
INSERT INTO Seat VALUES (4, '2022-10-14', 'Ariyarathna', '0755689321');

INSERT INTO FlightFare VALUES ('FF001', 'F001', concat('$',300.00), 'Electronic travel authorization');
INSERT INTO FlightFare VALUES ('FF002', 'F002', concat('$',350.00), 'Electronic travel authorization');
INSERT INTO FlightFare VALUES ('FF003', 'F003', concat('$',280.00), 'Visitor visa');
INSERT INTO FlightFare VALUES ('FF004', 'F004', concat('$',400.00), 'Visa is not required ');

```

Triggers

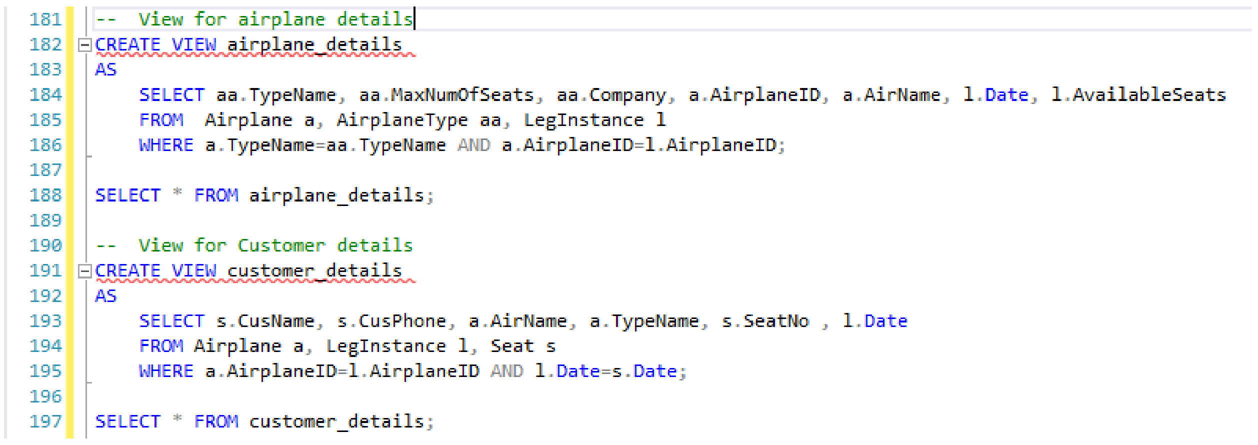


```sql

--Trigger for available seats
--Trigger 1- a trigger that shall fire upon the DELETE statement on the AirplaneType table
CREATE TRIGGER delete_AirplaneType_details
ON AirplaneType
INSTEAD OF DELETE
AS
BEGIN
    DECLARE @tname INT;
    DECLARE @count INT;
    SELECT @tname = TypeName FROM DELETED;
    SELECT @count = COUNT(*) FROM Airplane WHERE TypeName = @tname;
    IF @count = 0
        DELETE FROM AirplaneType WHERE TypeName = @tname;
    ELSE
        THROW 56000, 'can not delete - AirplaneType is referenced in other tables', 1;
END

```



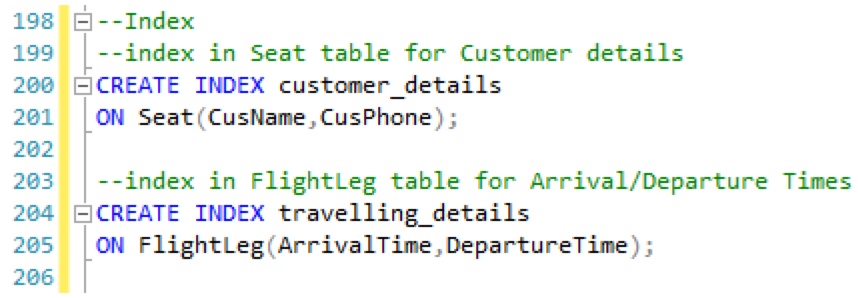
```sql
--Trigger 2- trigger for FlightLeg table that check before insert statement

DROP TRIGGER IF EXISTS leg_insert_check;
GO
CREATE TRIGGER leg_insert_check ON FlightLeg INSTEAD OF INSERT
AS BEGIN
    DECLARE @l_no VARCHAR(10);
    DECLARE @date DATE;
    DECLARE @a_code  VARCHAR(10);
    DECLARE @f_no  VARCHAR(10);
    DECLARE @deptime  TIME;
    DECLARE @arrtime  TIME;

    SELECT @l_no = LegNo, @date = Date, @a_code = AirportCode, @f_no = FlightNo, @deptime = DepartureTime, @arrtime = ArrivalTime FROM INSERTED;
    IF @l_no IS NULL SET @l_no = 0;
    IF @date IS NULL SET @date = GETDATE();
    INSERT INTO FlightLeg (LegNo, Date, AirportCode, FlightNo, DepartureTime, ArrivalTime) VALUES (@l_no, @date, @a_code, @f_no, @deptime, @arrtime);
END;

```

Views



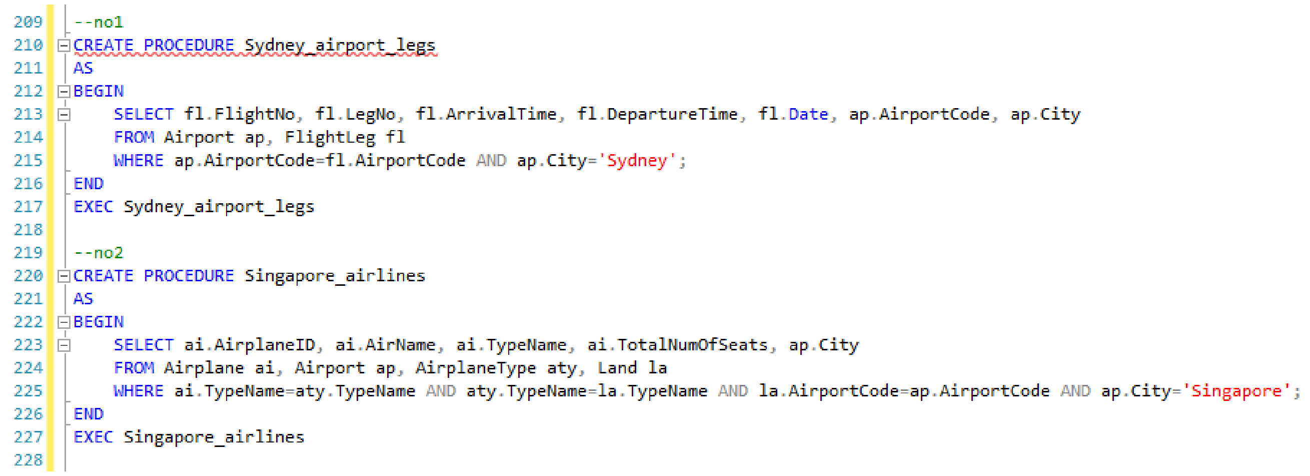
```
-- View for airplane details
CREATE VIEW airplane_details
AS
    SELECT aa.TypeName, aa.MaxNumOfSeats, aa.Company, a.AirplaneID, a.AirName, l.Date, l.AvailableSeats
    FROM  Airplane a, AirplaneType aa, LegInstance l
    WHERE a.TypeName=aa.TypeName AND a.AirplaneID=l.AirplaneID;

SELECT * FROM airplane_details;

-- View for Customer details
CREATE VIEW customer_details
AS
    SELECT s.CusName, s.CusPhone, a.AirName, a.TypeName, s.SeatNo , l.Date
    FROM Airplane a, LegInstance l, Seat s
    WHERE a.AirplaneID=l.AirplaneID AND l.Date=s.Date;

SELECT * FROM customer_details;
```

Indexes

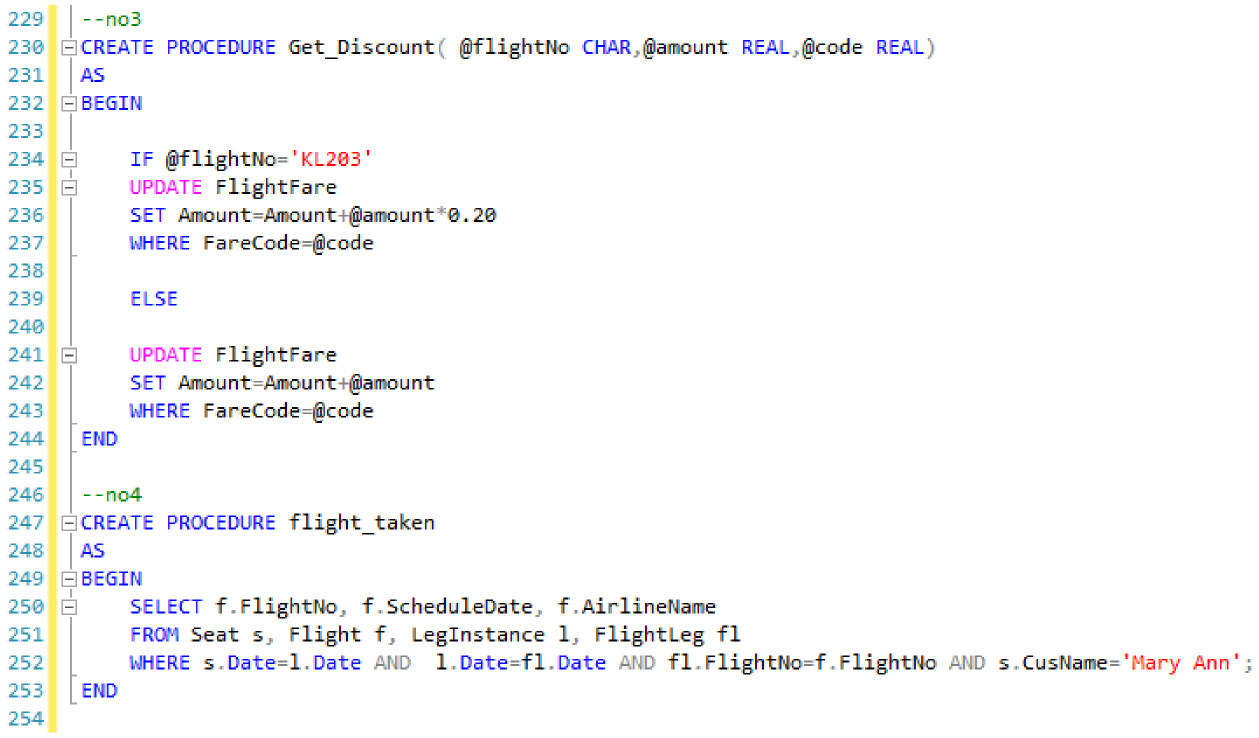


```
--Index
--index in Seat table for Customer details
CREATE INDEX customer_details
ON Seat(CusName,CusPhone);

--index in FlightLeg table for Arrival/Departure Times
CREATE INDEX travelling_details
ON FlightLeg(ArrivalTime,DepartureTime);
```

Procedures



```
--no1
CREATE PROCEDURE Sydney_airport_legs
AS
BEGIN
    SELECT fl.FlightNo, fl.LegNo, fl.ArrivalTime, fl.DepartureTime, fl.Date, ap.AirportCode, ap.City
    FROM Airport ap, FlightLeg fl
    WHERE ap.AirportCode=fl.AirportCode AND ap.City='Sydney';
END
EXEC Sydney_airport_legs

--no2
CREATE PROCEDURE Singapore_airlines
AS
BEGIN
    SELECT ai.AirplaneID, ai.AirName, ai.TypeName, ai.TotalNumOfSeats, ap.City
    FROM Airplane ai, Airport ap, AirplaneType aty, Land la
    WHERE ai.TypeName=aty.TypeName AND aty.TypeName=la.TypeName AND la.AirportCode=ap.AirportCode AND ap.City='Singapore';
END
EXEC Singapore_airlines
```



```
--no3
CREATE PROCEDURE Get_Discount( @flightNo CHAR,@amount REAL,@code REAL)
AS
BEGIN

    IF @flightNo='KL203'
    UPDATE FlightFare
    SET Amount=Amount+@amount*0.20
    WHERE FareCode=@code

    ELSE

    UPDATE FlightFare
    SET Amount=Amount+@amount
    WHERE FareCode=@code
END

--no4
CREATE PROCEDURE flight_taken
AS
BEGIN
    SELECT f.FlightNo, f.ScheduleDate, f.AirlineName
    FROM Seat s, Flight f, LegInstance l, FlightLeg fl
    WHERE s.Date=l.Date AND  l.Date=fl.Date AND fl.FlightNo=f.FlightNo AND s.CusName='Mary Ann';
END
```

Part 2

SQL injection

Introduction

Write about a SQL injection. An attack strategy that modifies backend SQL statements by controlling application input to exploit websites. SQL Injection happens when a designer doesn't as expected filter through potentially harmful characters and accepts user input that is put directly into a SQL Statement.
This enables an attacker to take, change, and delete data from your data set. Some SQL Servers, like Microsoft SQL Server, provide stored and extended procedures (information base server capacities). If an attacker is able to access these Procedures, they might be able to compromise the entire system. In-band SQL injections (Classic), inferential SQL injections (Blind), and out-of-band SQL injections are the three types of SQL injections. There are different types of SQL injection that can be characterized based on the techniques used to access the backend data and the potential harm they provide.

SQL Injection Types

In-band SQLI

The specific sort of SQL injection known as "classic SQL injection" is called "SQL injection." Anything that is "in-band" is a direct response that the attacker receives via the same communication channel that they employed to launch their attack. This is the case, for instance, if the attacker manually carries out the assault while utilizing a web browser.

In-band SQLI is 2 types as below,

1. Error based SQLI

Error-based SQL injection modifies the database's data by using error output. When doing an in-band injection, the attacker uses the same channel to acquire database data while conducting the attack. a common attack plan. Due to a programming fault, the server can instead send a SQL error rather than the requested data. Grep's extract function automates this process.

By only taking advantage of the database flaw, an attacker can frequently fully comprehend the database. [1]

2. Union based SQLI

The SQL union operator combines two statements into a single one. As part of the HTTP response, it is. The results of the combined select statements are then returned as one result.

There are two essential conditions that must be met for a UNION query to be effective:

1. The number of columns returned by each query must be the same.
2. There must be backward compatibility between each query and each of the data types in each respective column.

Inferential / blind SQLI

The attacker sends data payloads to the server and then waits to see how the server reacts and behaves. This technique, sometimes known as "blind SQLi," prevents an attacker from seeing in-band information about an attack since information is not provided to the hacker from a website database.

Blind SQL injections take longer to execute but could be just as hazardous because they rely on the server's response and behavior patterns. The following [2] categories can be used to classify blind SQL injections.

1. Boolean-based Blind SQLI

   By submitting a SQL query to the database, the attacker instructs the software to generate a value. The result will vary depending on whether the premise is accurate. The information in the HTTP response could change or stay the same depending on the result. The attacker can then decide if the message's conclusion was accurate or not. [2]

2. Time-based Blind SQLI

   In order to react to a SQL query issued by the attacker, the database is compelled to wait. Attackers may gain important knowledge about the legitimacy of a query from how long it takes the database to respond to a query. HTTP requests may receive immediate responses or ones that take some time to develop. The attacker can therefore ascertain if the message they used returned true or false without depending on the database. [2]

<u>Out-of-band SQLI</u>

Only if specific features on the database server utilized by the web application are enabled is this type of attack conceivable. Attacks using SQL injection frequently replace in-band and inferential SQLi approaches.

If an attacker is unable to launch and collect data via the same channel or if the server is too sluggish or unstable for them to do so, they may utilize SQLi out-of-band. Both DNS and HTTP depend on the server's processing capability to convey information to an intrusive party. [2]

How does a SQL injection attack work?

An SQL query is a command that gives specific instructions to an application database. Additionally, queries can be used to execute operating system commands. When a user executes a query, a set of parameters makes sure that just the desired records are returned. This gives attackers an opportunity to attempt a SQL injection and insert malicious code into the query's input form. Investigating the operation of the targeted database is the initial stage in a SQL injection attack. Different random values are given to the query in order to test the server's response.

After that, attackers create a query that the server will recognize and

Attackers then create a query that the server will interpret and execute as a SQL command using what they have learnt about the database. For instance, a database might contain details on consumers who have made purchases and have customer IDs. An attacker can use the input field to enter "CustomerID = 1000 OR 1=1" rather than looking for a specific customer ID. The SQL query would return all available customer IDs and any associated data because the statement 1=1 is always true. This gives the attacker access at the administrator level by bypassing authentication. SQL attacks can be created to erase an entire database, avoid the requirement for passwords, remove entries, or add undesired data, in addition to extracting illegal information.

Techniques and impact

An attacker can inject SQL Query/Commands through an information structure field if he knows that the system is defenseless against SQL Injection. Giving the attacker access to your information database and letting him execute any SQL command, including DROP TABLE to the data set, is the same thing!

On the vulnerable framework, an aggressor might carry out erratic SQL articulations. This could make you reevaluate how reliable your data set is and reveal sensitive information. There are different levels of information/framework access for the attacker depending on the back-end information base being used. It might be possible to modify current queries, UNION erratic information (used to choose related data from two tables), use sub selections, or affix more searches.

Your privacy may be violated by a SQL Injection, which can lead to the following effects:

confidentiality: As sensitive information is typically contained in SQL data bases, losing categorization is a problem with SQL Injection problems.

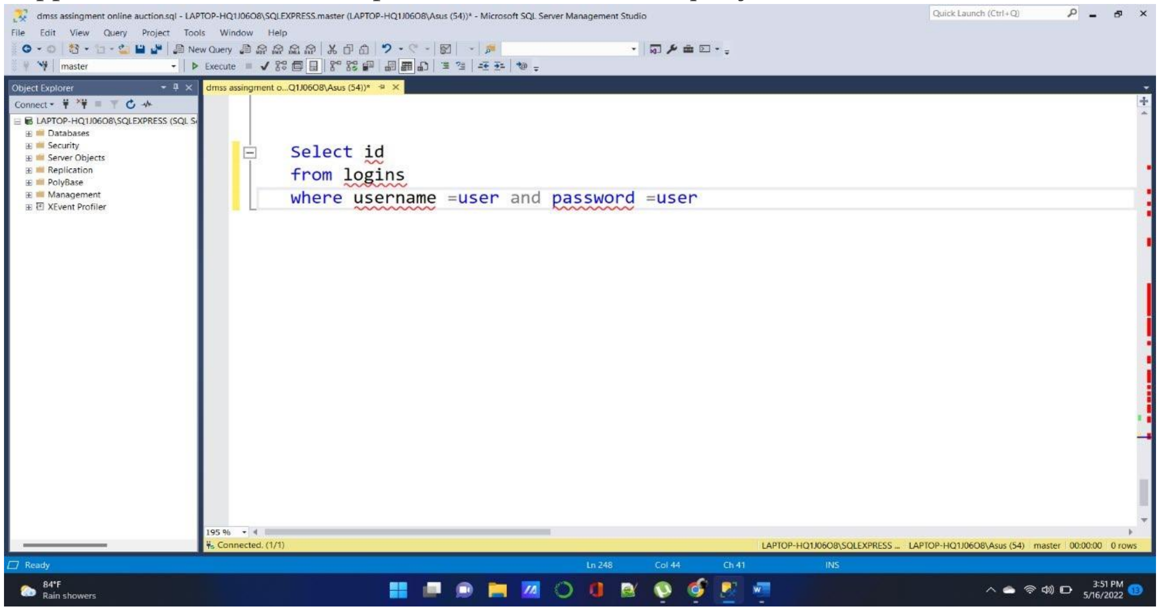
Verification: If weak SQL orders are used to validate client names and passwords, it may be possible to login to a system as a different client who is unaware of the secret phrase.

Authorization: A successful SQL Injection attack may be able to update permission data if it is stored in a SQL data set.

Integrity: A SQL Injection attack might potentially alter or even remove important data in addition to making it possible to browse it.

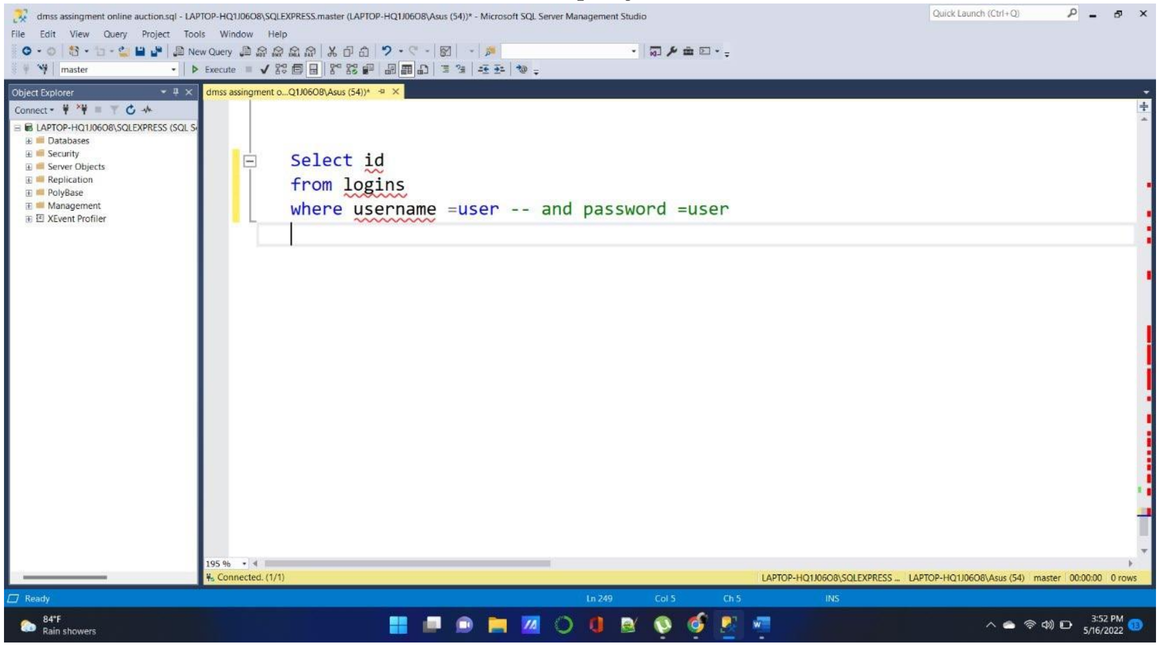
Let's now go into the straightforward process of SQL injection. Programmers may use SQL Injection to directly execute SQL instructions designed to go around the login structure border and examine what's going on there. This is only possible if the data sources are delivered directly with the SQL query to the data set without first being properly cleaned (i.e., made immune). SQL Injection flaws provide an attacker the ability to directly access the information database.

Suppose username is user & password is user So the query will be :



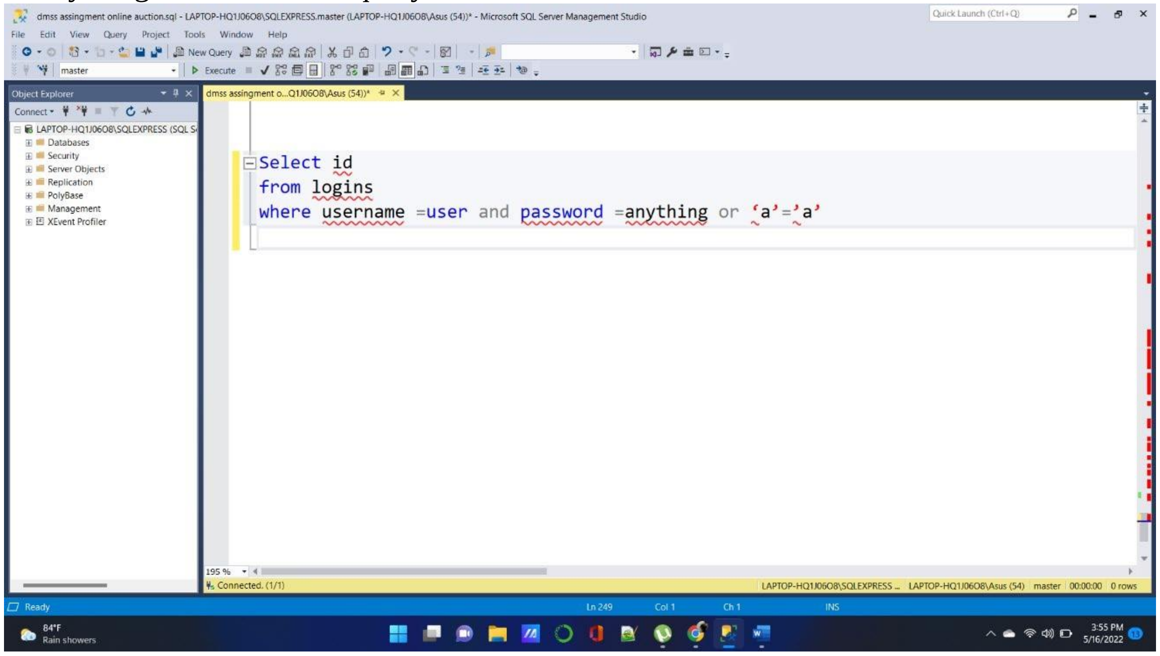
```
Select id
from logins
where username =user and password =user
```

So if hacker enters username as **user – -,** so query will like this :



```
Select id
from logins
where username =user -- and password =user
```

So SQL will just disregard inquiry after '- - ' and it will be consider as commented part. Or on the other hand, on the off chance that programmer enters secret phrase as anything or 'a'='a' so the query will be:



```
Select id
from logins
where username =user and password =anything or 'a'='a'
```

Due to the web application's contributions not being cleaned as planned, the use of single statements has changed the SQL order for "WHERE" into a two-part clause. No of what the first segment includes, the 'a'='a' component certifies that it is legitimate. The attacker will be able to bypass the login process without really knowing a strong username/secret word combination thanks to this!

What are some ways to spot and prevent a SQL injection attack?

Successful SQL injection attacks have the ability to expose confidential information and impair client trust, both of which could result in severe harm. It is crucial to identify this type of attack as soon as you can because of this.

The most widely-used defense against SQL injection threats is web application firewalls (WAF). WAFs may be set up to recognize dangerous SQL queries in online apps because they are built on a library of recent attack signatures.

Businesses should abide by the following guidelines to stop SQL injection attacks:

Provide personnel with preventative training.

IT personnel must undergo the essential security training to understand how SQLi attacks occur and how they can be avoided in online applications, especially DevOps experts, system administrators, and software development teams.

1. Do not believe user input.

The probability of a successful SQL injection grows with each user input included in a SQL query. The easiest method to reduce this kind of danger is to surround user input with security protections.

2. Rather than using a blocklist, use an allow list.

It is advisable to check and filter user input via an accept list as opposed to a blocklist because hackers may regularly get around a blocklist.

3. Update your route and make use of the most recent applications.

One of the most common reasons for SQL injection vulnerabilities is outdated software. SQLi security is not likely to be integrated into older technology, and unpatched software is typically easier to manipulate. The same is true for programming languages. Languages and syntax from earlier times are more vulnerable. Use PDO instead of older MySQL, for example.

4. Employ effective preventive measures.

Written from scratch query strings provide insufficient defense against a SQLi attack. Input validation, prepared statements, and parameterized queries are the best ways to secure web applications.

5. Conduct routine security scans

Regular web application scanning will find and fix such vulnerabilities before they do major harm.

7. Verifying the accuracy of input Simple input

Checks largely stop attacks. Check the type, size, length, format, and range before using user input. Only expected values should be permitted for string variables. Reject submissions containing data that is binary, escape, or not ASCII. This avoids buffer overruns and script insertion. Compare the data in an XML document with the schema.

Denial of Service Attack

What is DOS Attack?

Denial-of-service attacks can affect database management systems as well. Denial of Service (DoS) assaults are referred to as network-based attacks. Using a DOS attack, one can stop authorized users from using a resource or make it less responsive. 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 in an effort to tax the system's constrained resources.

Dos assaults attempt to harm a company's reputation, hurt clients, and result in financial losses; in the worst cases, they may even end in fatalities. Denial of service attacks are now the cybercrime that costs victims' organizations the most money.

Types of DOS Attacks and How they Work

1. **Bandwidth Attacks**

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

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 as a result of these assaults due to a particular feature or implementation defect of a protocol installed there.

3. **ICMP Flood Attacks**

   When an attacker repeatedly sends ICMP echo requests to a victim's computer, it is known as a "ping flood" (DoS). The attacker expects the victim to transmit an ICMP "echo reply" packet in response to each ICMP request, using up both incoming and outgoing bandwidth. When the attacker has more bandwidth than the target, it is most effective. User performance may suffer from a slow target computer.

4. **UDP Flood**

A UDP flood is a type of denial-of-service attack that aims to overwhelm a targeted server with User Datagram Protocol (UDP) packets. Additionally, if UDP flooding overwhelms the firewall defending the targeted server, a denial-of-service attack may take place.

5. **TCP SYN Flood Attacks**

An attacker can transmit connection requests from unreachable source addresses by taking advantage of TCP's three-way handshaking. The sender is unable to receive the last handshake message due to TCP-SYN flooding. The server must then allot RAM and wait for a pending connection as a result. Other systems are unable to communicate with the target system due to its clogged buffer.

Impact of DOS Attack

- Performance issues with the database/server
- Database or server unavailability
- Inability to access any database/server causes a rise in spam emails.
- A loss of wireless or wired internet service.

Countermeasures and Mitigation

All Database DoS attacks cannot be thwarted by a single strategy. Every feature has a potential for abuse; therefore no one fix can defend against every assault. DoS defense requires resources and time. Make sure your databases are configured and patched. Vulnerability is hard to understand. Removing network software

Preventive and investigative processes are required for high-profile databases.

By making the right decisions, we can reduce our vulnerability to a DOS assault.

- **Build redundancy into infrastructure**

If you want to make it harder for an attacker to carry out a DDoS attack against your servers, split your servers across different data centers and employ a load balancing system to disperse traffic among them. If possible, data centers should be situated in different countries or regions within the same nation.

- **Practice Basic Network Security**

Strict security controls can be implemented to protect networks against intrusion. Safe practices include things like anti-phishing tools and safe firewalls that only let in a certain amount of outside traffic.

- **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.

- **Check for security patches and keep updated.**