



Dublin Business School

excellence through learning

Advanced Databases Assignment

Easy Travels Database

Module Title: Advanced Databases (B9IS100)

Module Leader: Dr. Shazia A Afzal

Group Members: Manik Mahashabde (10518579)

Saurabh Devade (10531140)

Sabitha Maram (10533048)

TABLE OF CONTENTS

TABLE OF CONTENTS.....	1
TABLE OF FIGURES	3
1. BUSINESS CASE.....	4
2. SCOPE OF THE PROJECT	4
2.1 BUSINESS REQUIREMENTS.....	4
3. BUSINESS RULES.....	5
4. RELATIONAL SCHEMA	8
4.1 XML SCHEMA	10
5. Implementation in SQL SERVER	12
5.1 Tables with Data Diagram	12
5.2 Referential Integrity	12
5.3 STORED PROCEDURES.....	15
5.3.1. Employees should be able to login to the system, and their session should be maintained.	15
5.3.2. Employees should be able to create holiday/tour packages.....	16
5.3.3. Employees should be able to update package details.....	18
5.3.4. There should be functionality to search through the packages itinerary.	19
5.3.5 On Deletion of Customer entry, all the enquiries associated with that customer will be deleted.	20
5.3.6. Employees should be able to retrieve bookings created by customers with package details.....	22
5.3.7. Employees should be able to see the number of package based on package type..	22
5.4 TRIGGERS	23
5.5 VIEWS.....	27
8. Conclusion	29
9. Innovation	29
10. BIBLIOGRAPHY	31
11.APPENDIX A.....	32
CREATE TABLE QUERIES	32
12. APPENDIX B	34
13. INDIVIDUAL CONTRIBUTION.....	34

TABLE OF FIGURES

Figure 1 - ON DELETE CASCADE	5
Figure 2 - Package Id foreign key	6
Figure 3 - Employee role mapping	6
Figure 4 - Role mapping table	7
Figure 5 - Employee right function	7
Figure 6 - Schema diagram	8
Figure 7 - Schema.xsd file	9
Figure 8 - XML Schema.....	10
Figure 9 - Data Diagram	12
Figure 10 - package table	13
Figure 11 - Enquiry Table	13
Figure 12 - Booking Table	14
Figure 13 - Employee Role Mapping.....	14
Figure 14 - generating a unique session token	15
Figure 15 - Login Employee.....	15
Figure 16 - SP1 Output.....	16
Figure 17 - Create Package	17
Figure 18 - Check Employee Session Function.....	17
Figure 19 - Check employee has right function	18
Figure 20 - SP2 Output.....	18
Figure 21 - Update package Sp	19
Figure 22 - Update package execution	19
Figure 23 - SP3 Output.....	19
Figure 24 - Search xml sp	20
Figure 25 - search xml output	20
Figure 26 – Enquiry Table	20
Figure 27 - Delete customer sp.....	20
Figure 28 - Customer booking retrieve sp	22
Figure 29 - Customer booking retrieve output	22
Figure 30 - Get Customer Count	23
Figure 31 - Get Customer Count Execution	23
Figure 32 - Customer audit table	24
Figure 33 - package view.....	25
Figure 34 - Package Caption.....	26
Figure 35 - package trigger	26
Figure 36 - get_customer_enquiry_for_employee.....	27
Figure 37 - Get Customer PackageBooking CreatedBy Employee.....	28
Figure 38 - Azure Clod Database.....	29
Figure 39 - Connection from management studio	30

1. BUSINESS CASE

Easy Travel is a system developed for a Tour packages management company to handle their travel/tour packages. It also provides features for managing and engaging their customers. The system is designed in such a way that every employee of the company will be able to use it and will perform operation they are allowed to. Every employee has a specific role which constraint them to performs specific task based on their roles. System is designed in such a way that employees can have many roles and there can be many employees of one role. The system can also be used for creating a customer's booking and also for entering the enquiries generated by the customer. For any enquiries customer contacts, the employee of the company based on their role and then enquiry gets entered into the system by employee. Customer can make booking on their own from a website of company based on available packages. Further reports can be generated using the system based on multiple factors like customers booking, enquiries, types of packages available etc.

2. SCOPE OF THE PROJECT

This project involves developing a system for tour management. In this report, the database aspect of this system is covered. Database management system used is MS SQL SERVER 2017, because fo its robust architecture and mainly for its feature which supports the XML datatypes as well.

According to the requirement, we have developed a database which includes several tables connected to each other.

We have entered a 5 records in every table for demo purpose including XML data. According to the requirements, every user will have different and multiple roles. But In this project, we have assigned every user a single role as of now but there is a functionality implemented for assigning them a multiple role.

For packages, we have inserted five records including 3 INTERNATIONAL and 2 DOMESTIC packages. Every package has a minimum of 3 days of itinerary. Image URL and date of every itinerary is kept null.

Also, Just two employee roles are being used now which are OPERATION & SALES, and other roles are created just to show the feature.

Other than the requirements and rules mentioned in this project, there are a lot more requirements and rules which is not discussed here.

2.1 BUSINESS REQUIREMENTS

1. Employees should be able to login to the system, and their session should be maintained.
2. Employees should be able to create holiday/tour packages.

3. Employees should be able to update package details
4. There should be functionality to search through the packages itinerary.
5. Employees should be able to retrieve bookings created by customers with package details.
6. Employees should be able to see the number of package based on package type
7. On Deletion of Customer entry, all the enquiries associated with that customer will be deleted.

3. BUSINESS RULES

1. On deletion of a customer, all the enquiries associated with that customer must be deleted.

It is required that if we delete the customer from the customer table all the enquires associated to that customer should be deleted from the enquiry table.

For this purpose, we have assigned the DELETE CASCADE RULE in enquiries table on customer_id which is a foreign key in enquiry table and primary key in package table.

```
CREATE TABLE easy_travels.dbo.enquiry (
    enquiry_id VARCHAR(40) NOT NULL
    ,employee_id VARCHAR(40) NOT NULL
    ,customer_id VARCHAR(40) NOT NULL
    ,enquiry_detail TEXT NOT NULL
    ,enquiry_type VARCHAR(20) NOT NULL
    ,created_on DATETIME NOT NULL DEFAULT (GETDATE())
    ,required_days INT NULL
    ,required_nights INT NULL
    ,required_country VARCHAR(50) NULL
    ,CONSTRAINT enquiry_id_pk PRIMARY KEY CLUSTERED (enquiry_id)
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE easy_travels.dbo.enquiry
ADD CONSTRAINT customer_id_fk FOREIGN KEY (customer_id) REFERENCES dbo.customer (customer_id) ON DELETE CASCADE
GO

ALTER TABLE easy_travels.dbo.enquiry
ADD CONSTRAINT employee_id_fk1 FOREIGN KEY (employee_id) REFERENCES dbo.employee (employee_id)
GO
```

Figure 1 - ON DELETE CASCADE

2. One booking must have only one valid package from the package table

It is required that bookings done by customer must have a package, else booking cannot be done.

We have achieved this by setting not null in booking table on package_id field and assigning foreign key for package_id in booking table which is primary key in packages table

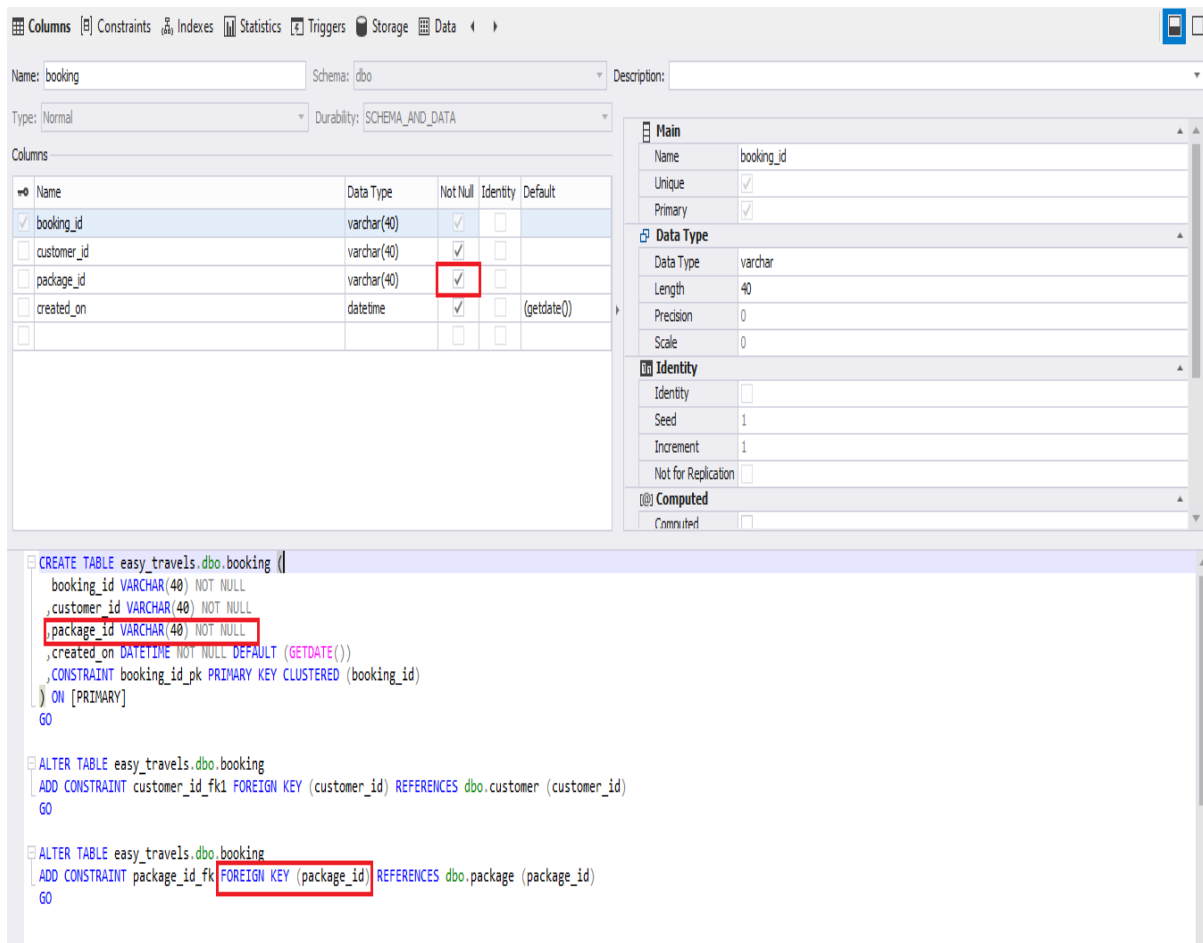


Figure 2 - Package Id foreign key

3. Every employee should have at least one or many roles.

It is required that every employee can have one or more than one roles. Ideally, every employee should have one role but, for a company which we are building a system is small scale company so one employee of that company can handle the many aspects of the organization.

Since it is many to many mapping, we have created a table named employee_mapping which contains the mapping of employee and roles. role is a table which contains all the static data which is mapped against the employees in mapping table with employee_id and role id as a foreign key constraints.

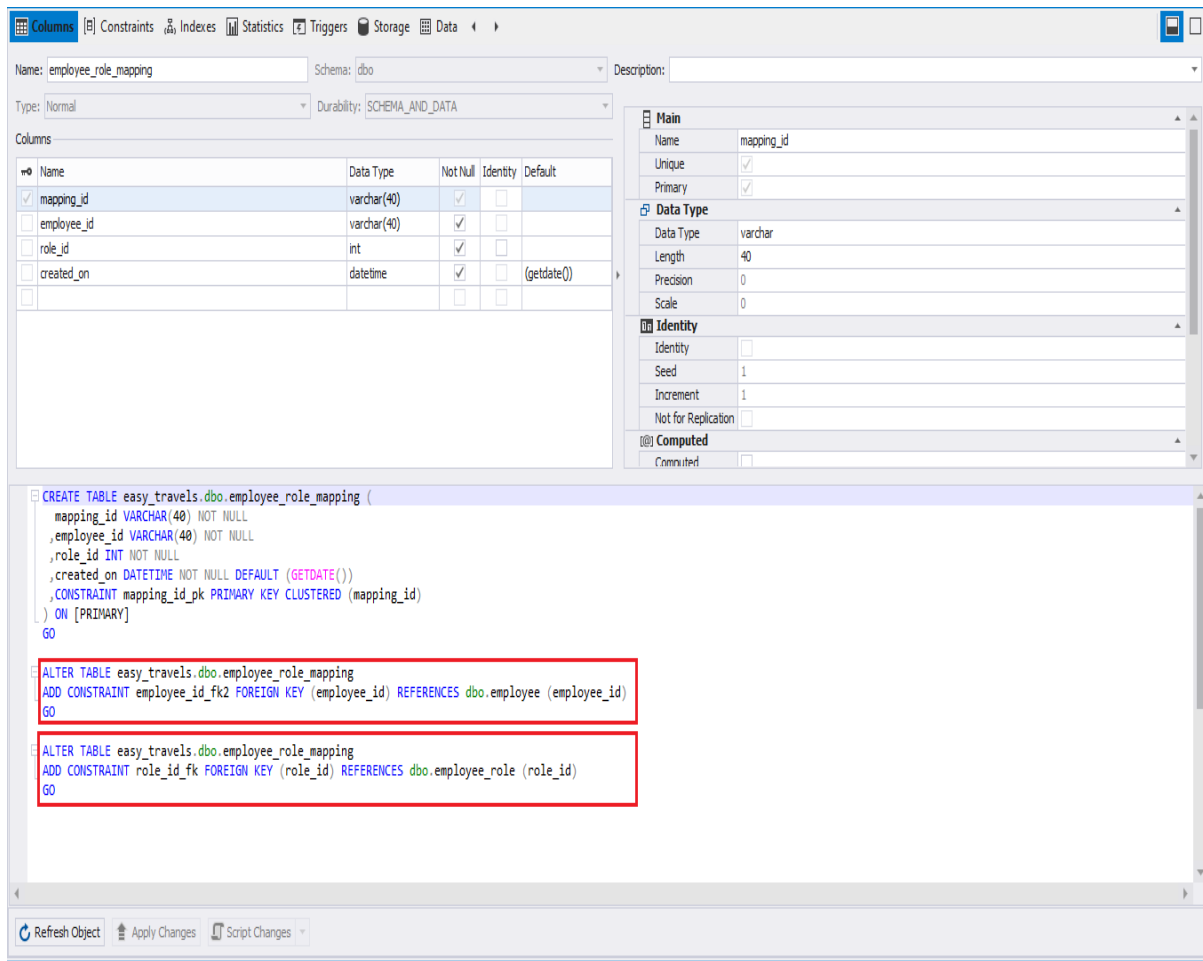


Figure 3 - Employee role mapping

mapping_id varchar(40)	employee_id varchar(40)	role_id int	created_on datetime
4af729be-7089-43f6-bc19-9fca6d3726ee	6848be80-5c58-4d4f-b11f-d1e76f6c92b5	4	13-12-2019 11:17:47.983 PM
ff3a7226-49a3-42ff-a842-1a7be63fc1b0	f8bc75b0-7b89-4a56-942e-139426b77483	2	09-12-2019 08:56:24.517 PM

Figure 4 - Role mapping table

4. For the creation of packages, only employees with operation role can create packages.

One of the rules is that every package should be created by the employee who has an OPERATION role i.e 4. Employees with other roles cannot create or update package. And employee with operation role cannot perform any other operations.

To achieve this functionality we have mapped a role to every employee. And while creating a package we check whether the employee creating a package has the authority to create the package. We have written a function for this.

mapping_id varchar(40)	employee_id varchar(40)	role_id int	created_on datetime
4af729be-7089-43f6-bc19-9fca6d3726ee	6848be80-5c58-4d4f-b11f-d1e76fc92b5	4	13-12-2019 11:17:47.983 PM
ff3a7226-49a3-42ff-a842-1a7be63fc1b0	f8bc75b0-7b89-4a56-942e-139426b77483	2	09-12-2019 08:56:24.517 PM

Below function will take two params @e_id i.e employee_id and @e_id. It will return **bit 1** if the employee has a specified role and will return **bit 0** if he doesn't have a right to perform this operation.

```

USE easy_travels
GO

ALTER FUNCTION dbo.check_employee_has_right (@e_id AS VARCHAR(40), @e_role_id AS INT)
RETURNS BIT
AS
BEGIN
    DECLARE @result BIT = 0
    IF (LEN(@e_id) != 0
        AND EXISTS (SELECT
            *
            FROM employee_role_mapping erm
            WHERE erm.employee_id = @e_id
            AND erm.role_id = @e_role_id)
    )
    BEGIN
        SET @result = 1
    END
    RETURN @result;
END

GO

```

Figure 5 - Employee right function

4. RELATIONAL SCHEMA

Normalization is a process of designing a database in such a manner that data redundancy and data dependency is reduced. It is a process of dividing a larger table into smaller sub tables and then linking the tables via relationship.(Kumar and Azad, 2017)

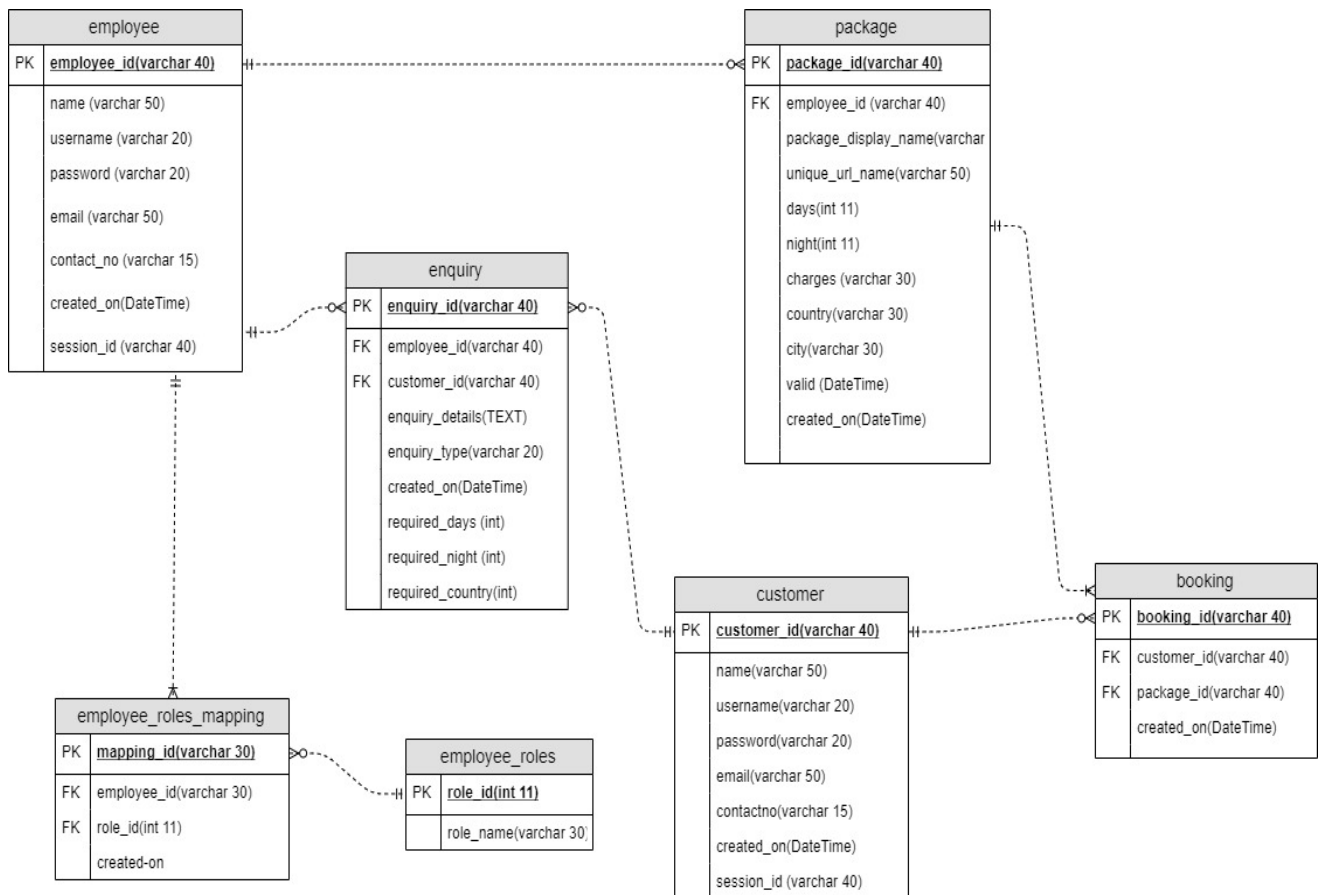


Figure 6 - Schema diagram

First Normal form: In the first normal form all the records should contain a single value and each record or row must be unique. Our schema is already in first normal form according to the data we have selected. (Ramez Elmasri and Navathe)

Second Normal form: In the second normal form the schema should be in first normal form and there should be no partial dependency. In our schema, all the values are already in 1NF and each table is having a unique primary key. There is no composite key, hence the schema is in 2NF. (Ramez Elmasri and Navathe)

Third Normal form: In the third normal form the schema should be in second normal form and there should be no transitive dependency. Transitive dependency is when a non-prime attribute depends upon another non-prime attribute in the table. In our schema, the data we have taken is in such a manner that all the non-prime attribute depends upon the primary key (Ramez Elmasri and Navathe)

of the table only and not on other non-prime attribute of the same table. We can say that our schema is in 3NF.

4.1 XML SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="1" maxOccurs="unbounded" ref="dayWiseIteneraries"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="dayWiseIteneraries">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="dayNumber"/>
        <xs:element ref="id"/>
        <xs:element ref="dayDate"/>
        <xs:element ref="packageName"/>
        <xs:element ref="dayImage"/>
        <xs:element ref="dayDetails"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="dayNumber" type="xs:integer"/>
  <xs:element name="id" type="xs:string"/>
  <xs:element name="dayDate" type="xs:date">
    <xs:complexType/>
  </xs:element>
  <xs:element name="packageName" type="xs:string"/>
  <xs:element name="dayImage" type="xs:string"/>
  <xs:complexType/>
</xs:element>
<xs:element name="dayDetails" type="xs:string"/>
</xs:schema>
```

Figure 7 - Schema.xsd file

Above diagram is created for defining our XML. XML schema is nothing but how and in what sequence your data will appear in XML format. It also shows what data type is used by each element. How many number of child elements it will have. It basically a well-defined structure of your XML. We can also validate our XML data with this schema. ("W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures", (Checiu and Ionescu, 2010))

4.2 Use of XML

As the use of Internet is growing day by day, data is also generated in proportion to that. Relational databases have certain limitation on various factors like scalability, maintenance and performance. So to store this vast amount of data, use of just relational databases is inappropriate, to overcome that now hybrid databases are being used. In hybrid databases, one can use XML data type to store huge amount of data. Using XML datatype multiple rows and columns can be converted to a single cell. Also, data can be stored in a well-structured form using datatypes as well as their properties in a sequential manner.

We can perform all the CRUD operations in XML data also like we use in relational databases.

Overall XML is used for storing the well defined, well-structured data which is relation with other data.

In our system, we have used XML for storing our day-wise itinerary. In this day-wise itinerary, we have to store the records of the number of days which contains many different types of data. And one package contains many such days, so if we were not using XML for this field, we had to use many rows for this data. Hence using this data we can combine all the values in single filed hence performance is also increased.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <dayWiseIteneraries>
    <dayNumber>1</dayNumber>
    <id>8180f742-7ba4-44b0-911d-c69342119246</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails>Bali: Arrive at the airport. Transfer to your hotel and check-in. Rest of the day is free at leisure.</dayDetails>
  </dayWiseIteneraries>
  <dayWiseIteneraries>
    <dayNumber>2</dayNumber>
    <id>c14db002-7aeb-4afe-950e-408d185d7106</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails> Bali: Breakfast at the hotel. Proceed for Full day Barong Kintamani tour. The highlight is the magnificent view of Mount and Lake Batur with the smoky Agung Volcano in
  </dayWiseIteneraries>
  <dayWiseIteneraries>
    <dayNumber>3</dayNumber>
    <id>dc93f809-fb84-48ee-a9e8-3215db6ec81c</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails>Bali: Breakfast at the hotel. Proceed to Safari Marine Park- Gianyar (approx. 1.5-2 hrs). The Safari Marine Park activities include Fresh water aquarium, Animal show, E
  </dayWiseIteneraries>
  <dayWiseIteneraries>
    <dayNumber>4</dayNumber>
    <id>0104b7fe-19f1-430f-ac10-21caa6fea7a</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails> Bali: Breakfast at the hotel. Proceed to Benoa Beach to take Glass Bottom Boat Turtle Island. On the way you can see many fishes and Coral reef. Arrive at the island wh
  </dayWiseIteneraries>
  <dayWiseIteneraries>
    <dayNumber>5</dayNumber>
    <id>1a951463-d3df-4aa1-a6ca-d3b7c392f85a</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails>Bali: Breakfast at the hotel. Free time during the morning to experience facilities at the hotel or go for a spa massage. Afternoon proceed to Tanah Lot. Tanah Lot Temp
  </dayWiseIteneraries>
```

Figure 8 - XML Schema

This is the sample of our XML data.

5. Implementation in SQL SERVER

5.1 Tables with Data Diagram

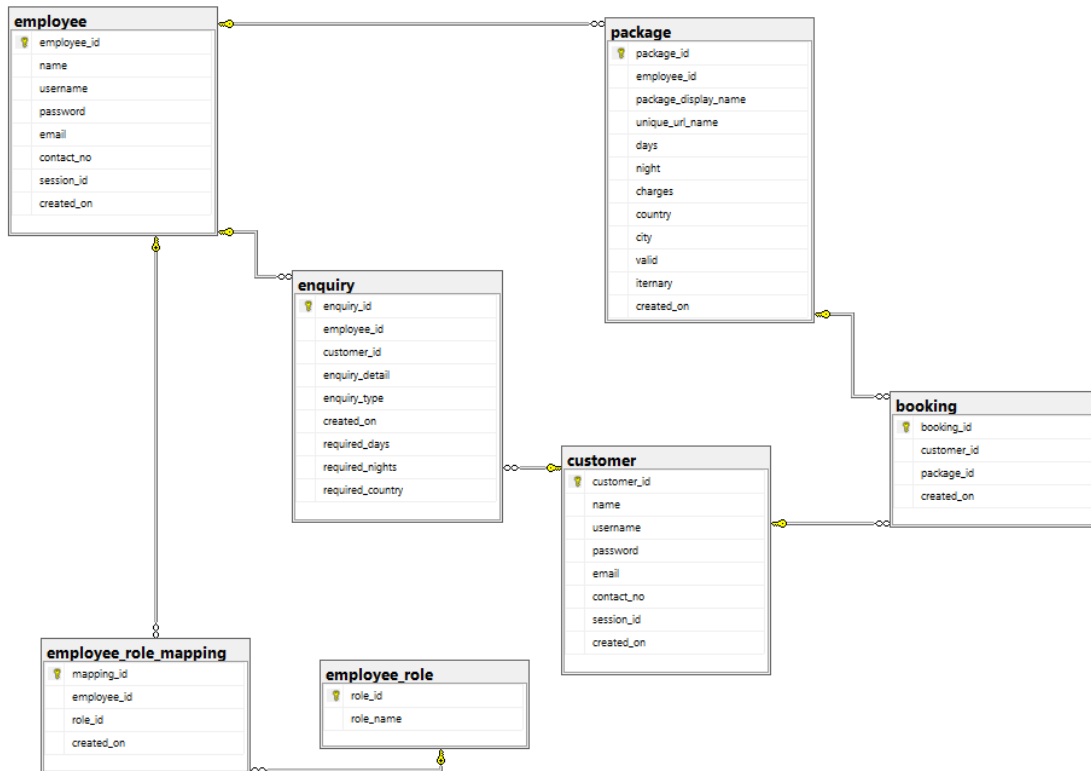


Figure 9 - Data Diagram

Above diagram shows the data diagram for our system.

Create a table and insert queries are placed in Appendix Section

5.2 Referential Integrity

Referential integrity is nothing but when we use the primary key of one table as the foreign key in another table, then that primary key should exist and should be valid. Referential integrity makes sure that all the data is integrated and accurate. Primary key cannot be modified or deleted if that is used as a foreign key in some other child table unless we set some specific rules. And using referential integrity data duplication is also avoided.

In this system, many referential integrities are being used discussed below.

1.Package Table

```

CREATE TABLE easy_travels.dbo.package (
    package_id VARCHAR(40) NOT NULL
    ,employee_id VARCHAR(40) NOT NULL
    ,package_display_name VARCHAR(100) NOT NULL
    ,unique_url_name VARCHAR(50) NOT NULL
    ,days INT NOT NULL
    ,night INT NOT NULL
    ,charges VARCHAR(30) NOT NULL
    ,country VARCHAR(30) NOT NULL
    ,city VARCHAR(30) NOT NULL
    ,valid DATETIME NOT NULL
    ,itinerary XML NULL
    ,created_on DATETIME NOT NULL DEFAULT (GETDATE())
    ,CONSTRAINT package_id_pk PRIMARY KEY CLUSTERED (package_id)
    ,UNIQUE (unique_url_name)
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE easy_travels.dbo.package
ADD CONSTRAINT employee_id_fk FOREIGN KEY (employee_id) REFERENCES dbo.employee (employee_id)
GO

```

Figure 10 - package table

As mentioned earlier employees can create packages. So we have used **employee_id_fk** as a foreign key constraint in the package table. employee_id is a primary key in the employee table.

If we try to delete the employee who has his entry in the package table, it will throw an error because we have used that employee's primary key as a foreign key package table. On the other hand package's entry cannot be created without employee id.

2.Enquiry Table

```

CREATE TABLE easy_travels.dbo.enquiry (
    enquiry_id VARCHAR(40) NOT NULL
    ,employee_id VARCHAR(40) NOT NULL
    ,customer_id VARCHAR(40) NOT NULL
    ,enquiry_detail TEXT NOT NULL
    ,enquiry_type VARCHAR(20) NOT NULL
    ,created_on DATETIME NOT NULL DEFAULT (GETDATE())
    ,required_days INT NULL
    ,required_nights INT NULL
    ,required_country VARCHAR(50) NULL
    ,CONSTRAINT enquiry_id_pk PRIMARY KEY CLUSTERED (enquiry_id)
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE easy_travels.dbo.enquiry
ADD CONSTRAINT customer_id_fk FOREIGN KEY (customer_id) REFERENCES dbo.customer (customer_id) ON DELETE CASCADE
GO

ALTER TABLE easy_travels.dbo.enquiry
ADD CONSTRAINT employee_id_fk1 FOREIGN KEY (employee_id) REFERENCES dbo.employee (employee_id)
GO

```

Figure 11 - Enquiry Table

In this table, two foreign keys are used **customer_id_fk** and **employee_id_fk1** because enquiries are created for customer so customer's id is used as foreign key and enquiries are created by employee so his id is also used as a foreign key.

3.Booking table

```
CREATE TABLE easy_travels.dbo.booking (
    booking_id VARCHAR(40) NOT NULL
    ,customer_id VARCHAR(40) NOT NULL
    ,package_id VARCHAR(40) NOT NULL
    ,created_on DATETIME NOT NULL DEFAULT (GETDATE())
    ,CONSTRAINT booking_id_pk PRIMARY KEY CLUSTERED (booking_id)
) ON [PRIMARY]
GO

ALTER TABLE easy_travels.dbo.booking
ADD CONSTRAINT customer_id_fk1 FOREIGN KEY (customer_id) REFERENCES dbo.customer (customer_id)
GO

ALTER TABLE easy_travels.dbo.booking
ADD CONSTRAINT package_id_fk FOREIGN KEY (package_id) REFERENCES dbo.package (package_id)
GO
```

Figure 12 - Booking Table

In this table, two foreign keys are used one is **customer_id_fk1** because booking is done by the customer and one is **package_id_fk** because booking is done against the package.

4.Employee Role Mapping

The screenshot displays the SQL Server Enterprise Manager interface for the 'employee_role_mapping' table in the 'dbo' schema. The 'Columns' tab is active, showing a table with five columns: 'mapping_id' (varchar(40), primary key, not null), 'employee_id' (varchar(40), not null), 'role_id' (int, not null), 'created_on' (datetime, default (getdate()), not null), and an empty column. The 'Data Type' tab is also visible, showing the properties for 'mapping_id'. Below the table design, the SQL script for creating and altering the table is shown. The script includes the CREATE TABLE statement with a primary key on 'mapping_id', and two ALTER TABLE statements adding foreign key constraints: 'employee_id_fk2' referencing 'employee' and 'role_id_fk' referencing 'employee_role'.

```
CREATE TABLE easy_travels.dbo.employee_role_mapping (
    mapping_id VARCHAR(40) NOT NULL
    ,employee_id VARCHAR(40) NOT NULL
    ,role_id INT NOT NULL
    ,created_on DATETIME NOT NULL DEFAULT (GETDATE())
    ,CONSTRAINT mapping_id_pk PRIMARY KEY CLUSTERED (mapping_id)
) ON [PRIMARY]
GO

ALTER TABLE easy_travels.dbo.employee_role_mapping
ADD CONSTRAINT employee_id_fk2 FOREIGN KEY (employee_id) REFERENCES dbo.employee (employee_id)
GO

ALTER TABLE easy_travels.dbo.employee_role_mapping
ADD CONSTRAINT role_id_fk FOREIGN KEY (role_id) REFERENCES dbo.employee_role (role_id)
GO
```

Figure 13 - Employee Role Mapping

As per the business requirement, every employee should have some role and every employee can have multiple roles, and a single role can be shared by multiple employees.

So, one foreign key used for **role_id_fk** which is primary key of roles table and other is **employee_id_fk2** because roles are mapped to employees.

5.3 STORED PROCEDURES

5.3.1. Employees should be able to login to the system, and their session should be maintained.

Features Used – Custom functions, Inbuilt functions

It is one of the most important requirement that while performing any operation, employee should be logged in and has a valid session token.

We have achieved this using a stored procedure and function as shown below.

```
CREATE FUNCTION dbo.generate_session_token(@e_id as varchar(40))
returns varchar(80)
AS
BEGIN
    DECLARE @session_token as varchar(80)
    DECLARE @timestamp as varchar(40)
    SET @timestamp= CONVERT(varchar(40),(DATEDIFF_BIG(MILLISECOND,'1970-01-01 00:00:00.000', SYSUTCDATETIME())));
    SET @session_token=CONCAT(@e_id,@timestamp)

    return @session_token
END
GO
```

Figure 14 - generating a unique session token

The above-shown image is a function written for generating a unique session token. It accepts an employee id as a parameter. Then using DATEDIFF_BIGG function it finds a current timestamp in milliseconds and in the next step it concatenates the time stamp with the employee id provided as a parameter and returns the same.

For eg – if we pass **'5d2f-f25g4f5g-2s4dsd-sd41s5d4-ds'** as emp id then function will return **'5d2f-f25g4f5g-2s4dsd-sd41s5d4-ds11515151115'** string as a output

```
1 ALTER PROCEDURE dbo.login_employee(@e_username varchar(20),@e_password varchar(20))
2 AS
3 BEGIN
4     DECLARE @e_count int
5
6     SELECT @e_count = COUNT (e.employee_id) FROM employee e where e.username=@e_username AND e.password=@e_password
7     IF (@e_count=1)
8     BEGIN
9         DECLARE @sessionToken varchar(80)
10        DECLARE @e_empId varchar(40)
11
12        SELECT @e_empId = e.employee_id FROM employee e where e.username=@e_username AND e.password=@e_password;
13        set @sessionToken = dbo.generate_session_token(@e_empId);
14        update employee set session_id=@sessionToken where employee_id=@e_empId
15    END
16    else
17    BEGIN
18        print('WRONG CREDENTIALS')
19    END
20 END
21 GO
22
23 EXEC dbo.login_employee 'saurabh_123' , 'pass@123'
```

100 %
Messages
(1 row affected)
Completion time: 2019-12-16T18:14:20.0018330+00:00

Figure 15 - Login Employee

Above stored procedure is created for login functionality of an employee.

It takes @e_username and @e_password as an input parameter first it checks whether employee records exist with this username and password if it exists it stores 1 in a count variable. If count is 1 then it proceeds ahead or it will print '**WRONG CREDENTIALS**' as an error message. After that, it fetches employee id from the employee table with that credentials And passes it to the function **generate_session_token(@e_empId)** for generating session token. Once the session token is generated, then the employee table is updated with the new session token. And further operations are performed based on this session token.

OUTPUT –

employee_id varchar(40)	name varchar(50)	username varchar(20)	password varchar(20)	email varchar(50)	contact_no varchar(15)	session_id varchar(80)	created_on datetime
02b10ba6-4853-4442-b5a3-5697367fd122	Pratik Yadav	psy_1234	ydava@pratik	ypiyadav@gmail.com	9898956556	(null)	17-12-2019 02:10:53.440 PM
6848be80-5c58-4d4f-b11f-d1e76f6c92b5	Manik Mahashabde	manik_123	man@1991	manik@mydbs.ie	867866545	(null)	09-12-2019 08:51:27.437 PM
9eb496f7-449a-4db4-8cf4-1f9c02ddec3f	Sagar Patil	patil_sagar	sg@1995	sagar.patil@gmail.com	5645345454	(null)	17-12-2019 02:09:04.393 PM
a344f02-3272-44bb-8507-4f96d65738f	Sabitha Maram	sabitha@1997	sab_maram	sabitha@gmail.com	9565968595	(null)	09-12-2019 08:52:27.907 PM
f8bc75b0-7b89-4a56-942e-139426b77483	Saurabh Devade	saurabh_123	pass@123	saurabh.devade21@gmail.com	897646767	f8bc75b0-7b89-4a56-942e-139426b774831576433659993	09-12-2019 08:44:02.313 PM

Figure 16 - SP1 Output

Session token is generated for the user whose credentials are passed.

5.3.2. Employees should be able to create holiday/tour packages.

Features Used – XML insert, Custom functions, Inbuilt functions

One of the requirement is that the employee should be able to create holiday packages for a specific tour. But as mentioned in the Business case it is required that employees with OPERATION role can create packages and other employees cannot do it.

Separate table is mentioned for maintaining employees and roles mapping.

mapping_id varchar(40)	employee_id varchar(40)	role_id int	created_on datetime
4af729be-7089-43f6-bc19-9fca6d3726ee	6848be80-5c58-4d4f-b11f-d1e76f6c92b5	4	13-12-2019 11:17:47.983 PM
ff3a7226-49a3-42ff-a842-1a7be63fc1b0	f8bc75b0-7b89-4a56-942e-139426b77483	2	09-12-2019 08:56:24.517 PM

And static data of employees are kept in another table.


```

CREATE PROCEDURE dbo.sp_create_package(
    @package_id varchar(40),
    @employee_id varchar(40),
    @package_display_name varchar(100),
    @unique_url_name varchar(50),
    @night int,
    @days int,
    @charges varchar(30),
    @country varchar(30),
    @city varchar(30),
    @valid datetime,
    @itinerary xml,
    @session_token varchar(80))
AS
BEGIN
    if(dbo.check_employee_session(@employee_id,@session_token)!=0 AND
       dbo.check_employee_has_right(@employee_id,2)!=0
    )
    BEGIN
        INSERT INTO package (charges,city,country,days,employee_id,night,package_display_name,package_id,unique_url_name,valid,itinerary)
        VALUES (@charges,@city,@country,@days,@employee_id,@night,@package_display_name,@package_id,@unique_url_name,@valid,@itinerary)
    END
    ELSE
    BEGIN
        print('FALSE')
    END
END
GO

```

Figure 17 - Create Package

This stored procedure is created for creating packages.

It accepts all the parameter which is required for creating a package and an extra **@session_token** parameter to check employees authority.

```

USE easy_travels
GO

ALTER FUNCTION dbo.check_employee_session (@e_id AS VARCHAR(40), @session_token VARCHAR(80))
RETURNS BIT
AS
BEGIN
    DECLARE @result BIT = 0

    IF (LEN(@session_token) != 0)
    BEGIN
        DECLARE @e_count INT
        SELECT
            @e_count = COUNT(*)
        FROM employee e
        WHERE e.employee_id = @e_id
        AND e.session_id = @session_token;

        IF (@e_count != 0)
        BEGIN
            SET @result = 1;
        END
    END

    RETURN @result;
END
GO

```

Figure 18 - Check Employee Session Function

Above function is created to check whether the employee is logged in and has a valid session. It accepts two parameters **@e_id** and **@session_token** and will return BIT as an output.

It will return 1 if the record is found with provided session_token and employee id that means employee is logged in and has a valid session token and will return a 0 if no such record is found that means the employee is not logged in.

Then to check whether the employee has a right to perform this operation we have another function named as **check_employee_has_right**.

```
USE easy_travels
GO

ALTER FUNCTION dbo.check_employee_has_right (@e_id AS VARCHAR(40), @e_role_id AS INT)
RETURNS BIT
AS
BEGIN
    DECLARE @result BIT = 0
    IF (LEN(@e_id) != 0
        AND EXISTS (SELECT
            *
            FROM employee_role_mapping erm
            WHERE erm.employee_id = @e_id
            AND erm.role_id = @e_role_id)
    )
    BEGIN
        SET @result = 1
    END
    RETURN @result;
END

GO
```

Figure 19 - Check employee has right function

While calling this function we have to pass two arguments i.e **@e_id** and **@e_role_id**. Function will check whether a record exists for an employee for the provided role and will return BIT accordingly that. And if both the conditions satisfy, it will insert data into the package table having itinerary as an XML datatype.

OUTPUT-

Once the query is executed successfully data gets inserted into the table.

package_id varchar(40)	employee_id varchar(40)	package_display_name varchar(100)	unique_url_name varchar(30)	days int	night int	charges varchar(30)	country varchar(30)	city varchar(30)	valid datetime	itinerary xml	created_on datetime
10ddc398-4405-48ad-ab5f-95322451eff3	f8bc75b0-7b89-4a56-942e-139426b77483	Bali Escape	bali-escape	6	5	50,000	Indonesia	Baturiti	01-01-2020 12:00:00.000 AM	<root>...	13-12-2019 09:
50fe200a-70ec-40e9-90ca-8a30760b001e	f8bc75b0-7b89-4a56-942e-139426b77483	Spectacular Thailand	Spectacular Thailand	6	5	1525296	Thailand	Phuket	27-12-2019 02:49:30.853 PM	<root>...	17-12-2019 02:
a13ad548-639c-4462-83d0-d858b23b341c	f8bc75b0-7b89-4a56-942e-139426b77483	Singapore Sentosa Getaway	singapore-sentosa-getaway	4	3	100000	Singapore	Sentosa	19-12-2019 02:28:42.430 PM	<root>...	17-12-2019 02:
b7c12385-856d-46b4-bd05-954a531c7444	f8bc75b0-7b89-4a56-942e-139426b77483	Australia's Northern Territory	australia	8	7	1022201	Australia	Sydney	19-12-2019 03:07:33.273 PM	<root>...	17-12-2019 03:
bc66376f-21f6-4bac-84b3-fa016321b89b	f8bc75b0-7b89-4a56-942e-139426b77483	Glimpses of Muscat	glimpses-of-muscat	4	3	120000	Oman	Muscat	12-12-2019 02:58:07.063 PM	<root>...	17-12-2019 02:

Figure 20 - SP2 Output

5.3.3. Employees should be able to update package details

Features Used – XML modify, Custom functions, Inbuilt functions

Employees should be able to update XML data available in package

```

1 ALTER PROCEDURE dbo.sp_update_package_itinerary_day_details(
2     @package_id varchar(40),
3     @employee_id varchar(40),
4     @session_token varchar(80),
5     @day_number as varchar(10),
6     @day_data as varchar(max))
7 AS
8 BEGIN
9
10
11     if(dbo.check_employee_session(@employee_id,@session_token)!=0 AND
12        dbo.check_employee_has_right(@employee_id,2)!=0
13        )
14     BEGIN
15         UPDATE package
16         set itinerary.modify('replace value of (/root/dayWiseItineraries/dayDetails/text())[1] with sql:variable("@day_data")' ) where package_id=@package_id
17     END
18 ELSE
19     BEGIN
20         print('FALSE')
21     END
22 END
23
24 END
25 GO

```

Figure 21 - Update package Sp

```

EXEC sp_update_package_itinerary_day_details '10ddc398-4405-48ad-ab5f-95322451eff3',
'f8bc75b0-7b89-4a56-942e-139426b77483',
'f8bc75b0-7b89-4a56-942e-139426b774831576433659993','1','Updated DATA'

```

Figure 22 - Update package execution

When we execute the stored procedure and pass data to be updated. Its update the XML data of package which is passed as an input parameter.

OUTPUT –

```

<?xml version="1.0"?>
<root>
  <dayWiseItineraries>
    <dayNumber>1</dayNumber>
    <id>8180f742-7ba4-44b0-911d-c69342119246</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails>Updated DATA</dayDetails>
  </dayWiseItineraries>
  <dayWiseItineraries>
    <dayNumber>2</dayNumber>
    <id>c14db002-7aeb-4afe-950e-408d185d7106</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails> Bali: Breakfast at the hotel. Proceed for Full day Barong Kintamani tour. The highlight is the magnificent view of Mount and Lake Batur with the smoky Agung Vo.
  </dayWiseItineraries>
  <dayWiseItineraries>
    <dayNumber>3</dayNumber>
    <id>dc93f809-fb84-48ee-a9e8-3215db6ec81c</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails>Bali: Breakfast at the hotel. Proceed to Safari Marine Park - Gianyar (approx. 1.5-2 hrs). The Safari Marine Park activities include Fresh water aquarium, Anima.
  </dayWiseItineraries>
  <dayWiseItineraries>
    <dayNumber>4</dayNumber>
    <id>0104b7fe-19f1-430f-ac10-21caac6fea7a</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails> Bali: Breakfast at the hotel. Proceed to Benoa Beach to take Glass Bottom Boat Turtle Island. On the way you can see many fishes and Coral reef. Arrive at the :
  </dayWiseItineraries>
  <dayWiseItineraries>
    <dayNumber>5</dayNumber>
    <id>1a951463-d3df-4aa1-a6ca-d3b7c392f85a</id>
    <dayDate />
    <packageName>bali-escape</packageName>
    <dayImage />
    <dayDetails>Bali: Breakfast at the hotel. Free time during the morning to experience facilities at the hotel or go for a spa massage. Afternoon proceed to Tanah Lot. Tanah
  </dayWiseItineraries>
</root>

```

Figure 23 - SP3 Output

5.3.4. There should be functionality to search through the packages itinerary.

Features Used – XML search, LIKE statement, Custom functions, Inbuilt functions

If employee or customer wants to search for a package with specific information there should be some feature for that. Every package has separate Day-wise itinerary in XML format, and it is possible that employee or customer want to search through that XML data as well.

```
USE easy_travels
GO

ALTER PROCEDURE dbo.search_iternaries (@searchtext AS VARCHAR(10))
AS
BEGIN
    SELECT
        *,
        iternary.value('(/root)[1]', 'varchar(max)')
    FROM package p
    WHERE iternary.value('(/root)[1]', 'varchar(max)') LIKE '%' + @searchtext + '%'
END
GO
```

Figure 24 - Search xml sp

Above stored procedure search_iternaries accepts one input parameter which is nothing but a search field. We have to search from an entire package itinerary that's why we are searching from a root level using the **LIKE** keyword. '%' sign at both ends indicates that it will find all matched records which contain the passed input.

Output:

EXEC dbo.search_iternaries 'Bali'

package_id	employee_id	package_display_name	unique_url_name	days	night	charges	country	city	valid	iternary	created_on
104dc398-4405-48ad-ab5f-95322451ef3	f8bc79b0-7689-4a56-942e-139426b77483	Bali Escape	bali-escape	6	5	50,000	Indonesia	Batu...	2020-01-01 00:00:00.000	root->day/WeekItinerary->dayNumber-1->dayNumber	2019-12-13 21:06:16.087

Figure 25 - search xml output

From the screenshot above it can be seen that when we execute the Stored procedure passing Bali as value it returns us all the packages which have Bali as a text in it.

5.3.5 On Deletion of Customer entry, all the enquiries associated with that customer will be deleted.

Features Used - DELETE CASCADE statement, Inbuilt functions

When an employee wants to delete a customer's record. All the enquiries associated with that customer should automatically get deleted. It is needed to maintain the database very neatly and reduce the employee's works to remove the enquiries manually.

For that, we have mapped a customer_id as a foreign key in enquiries table, and have set ON DELETE CASCADE rule which will automatically remove the customer's enquiry from the enquiries table.

```

CREATE TABLE easy_travels.dbo.enquiry (
    enquiry_id VARCHAR(40) NOT NULL
    ,employee_id VARCHAR(40) NOT NULL
    ,customer_id VARCHAR(40) NOT NULL
    ,enquiry_detail TEXT NOT NULL
    ,enquiry_type VARCHAR(20) NOT NULL
    ,created_on DATETIME NOT NULL DEFAULT (GETDATE())
    ,required_days INT NULL
    ,required_nights INT NULL
    ,required_country VARCHAR(50) NULL
    ,CONSTRAINT enquiry_id_pk PRIMARY KEY CLUSTERED (enquiry_id)
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE easy_travels.dbo.enquiry
ADD CONSTRAINT customer_id_fk FOREIGN KEY (customer_id) REFERENCES dbo.customer (customer_id) ON DELETE CASCADE
GO

ALTER TABLE easy_travels.dbo.enquiry
ADD CONSTRAINT employee_id_fk1 FOREIGN KEY (employee_id) REFERENCES dbo.employee (employee_id)
GO

```

Figure 26 – Enquiry Table

Above image shows the foreign key **customer_id_fk** mapping and the **ON DELETE CASCADE**.

```

CREATE PROCEDURE dbo.sp_delete_customer(
    @customer_id varchar(40),
    @employee_id varchar(40),
    @session_token varchar(80))
AS
BEGIN
    if(dbo.check_employee_session(@employee_id,@session_token)!=0 AND
    dbo.check_employee_has_right(@employee_id,4)!=0
    )
    BEGIN
        --Delete customer where customer_id = @customer_id
        print(@customer_id)

        END
    ELSE
    BEGIN
        print('FALSE')
        END
    END
GO

```

Figure 27 - Delete customer sp

As mentioned earlier every employee can perform specific operations only based on their role. And it is also discussed that employee should have valid session token to perform operations. This two check is implemented in separate functionalities and called multiple time as shown in the above figure.

Once an employee is authenticated and authorized, the delete query is executed and as soon as this operation is performed customers enquiries associated with it is also deleted automatically.

5.3.6. Employees should be able to retrieve bookings created by customers with package details.

Features Used – Multiple Sql Joins, Custom functions, Inbuilt functions

Once the booking of an employee is done against the package id, its records will be saved in a booking table. To avoid the data duplication we are not storing all the package data into the booking table. We are just mapping the package id against the booking id and customer id.

After that, when customer or employee wants to see all the booking details which is nothing but a package and customers details we will have to join three tables.

```
CREATE PROCEDURE dbo.get_customer_all_booking_details(@customer_id varchar(40))
AS
BEGIN
    DECLARE @e_count int

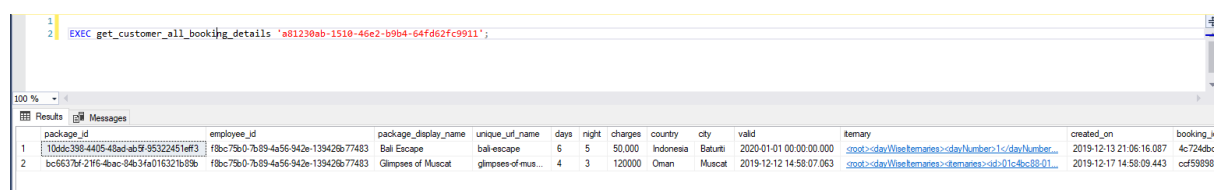
    SELECT * FROM package p LEFT JOIN booking b ON p.package_id=b.package_id
    LEFT JOIN customer c ON c.customer_id=b.customer_id where b.customer_id=@customer_id
END
GO
```

Figure 28 - Customer booking retrieve sp

We can see two joins here and customer_id is passed as an argument. Joins are used for joining two tables and accessing their data.

Here we want to retrieve all the package details from package_id which is stored in bookings table so we have used a join on **package_id** of package table and **package_id** from booking table. Same is done for customer details also we have joined customer table and booking table over the customer_id for retrieving customers details as well.

OUTPUT –



The screenshot shows a SQL Server query window with the following command executed:

```
EXEC get_customer_all_booking_details 'a81230ab-1510-46e2-b9b4-64fd62fc9911';
```

The output is displayed in a table with the following columns: package_id, employee_id, package_display_name, unique_url_name, days, night, charges, country, city, valid, itemary, created_on, and booking_id. The table contains two rows of data.

package_id	employee_id	package_display_name	unique_url_name	days	night	charges	country	city	valid	itemary	created_on	booking_id
10d6c398-4405-48ad-ab5f-95322451ef93	f8bc79b0-7b89-4a56-942e-139429b77483	Bali Escape	bali-escape	6	5	50,000	Indonesia	Batuti	2020-01-01 00:00:00.000	groot->dayWiseItemaries->dayNumber>1<dayNumber...	2019-12-13 21:06:16.087	4c724dbc
bc6637af-21f5-4bac-94b3-fa016321b89b	f8bc79b0-7b89-4a56-942e-139429b77483	Glimpses of Muscat	glimpses-of-muscat	4	3	120000	Oman	Muscat	2019-12-12 14:58:07.063	groot->dayWiseItemaries->itemaries>id>01<c4bc8801...	2019-12-17 14:58:09.443	cdf59898

Figure 29 - Customer booking retrieve output

5.3.7. Employees should be able to see the number of package based on package type

Features Used – HAVING, GROUP BY, COUNT, Custom functions, Inbuilt functions

There are two types of enquiries basically. INTERNATIONAL and DOMESTIC. Once all the enquiries are generated if someone wants to see the number of enquires generated based on type there is featured implemented for this.

```

CREATE PROCEDURE dbo.get_customer_count_by_enquiry_type(@count int)
AS
BEGIN
SELECT COUNT(customer_id) as number_of_customer, enquiry_type
from enquiry group by enquiry_type having COUNT(customer_id) > @count
END
GO

```

Figure 30 - Get Customer Count

Fro the code shown in the above screenshot we have used an inbuilt **COUNT()** function for counting a number of records. But instead of showing all the records we are grouping them bye a **group by** statement. We are grouping them using type column so it will group all the row which has the same type. And at last, we are passing a count as an argument so it will return the type of enquires which are above this passed count.

1	
2	EXEC get_customer_count_by_enquiry_type 1

100 %		
Results Messages		
	number_of_customer	enquiry_type
1	2	DOMESTIC
2	3	INTERNATION...

Figure 31 - Get Customer Count Execution

5.4 TRIGGERS

Triggers are the special stored procedure which is automatically executed whenever any DML, DDL action takes place. (Yoon *et al.*, 2001; Ding *et al.*, 2013)

We have created two triggers for our database.

1. AFTER Trigger for customer record delete and insert from customer table.

In this case, we have created a table named Customer.audit which will consist of inserted or deleted records from the customer table.

It is shown below:

```
CREATE TABLE [dbo].[customer.audit](
    [change_id] INT IDENTITY PRIMARY KEY,
    [customer_id] [varchar](40) NOT NULL,
    [name] [varchar](50) NOT NULL,
    [username] [varchar](20) NOT NULL,
    [password] [varchar](20) NOT NULL,
    [email] [varchar](50) NOT NULL,
    [contact_no] [varchar](15) NOT NULL,
    [session_id] [varchar](40) NULL,
    [created_on] [datetime] NOT NULL DEFAULT (getdate()),
    updated_at DATETIME NOT NULL,
    operation CHAR(3) NOT NULL,
    CHECK(operation = 'INS' or operation='DEL')
);
```

100 %

Messages

Command(s) completed successfully.

Figure 32 - Customer audit table

Then we have an AFTER trigger delete-customer-trigger which will be called after action is taken on the customer table. It will insert the delete/insert records in the customer-audit table.

```
CREATE TRIGGER [dbo].[delete_customer_trigger]
ON [dbo].[customer]
AFTER INSERT, DELETE
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO [dbo].[customer.audit](
        customer_id,
        name,
        username,
        password,
        email,
        contact_no,
        session_id,
        created_on,
        updated_at,
        operation
    )
    SELECT
        i.customer_id,
        name,
        username,
        password,
        email,
        contact_no,
        session_id,
        created_on,
        updated_at,
        operation
    FROM [dbo].[customer] i
```

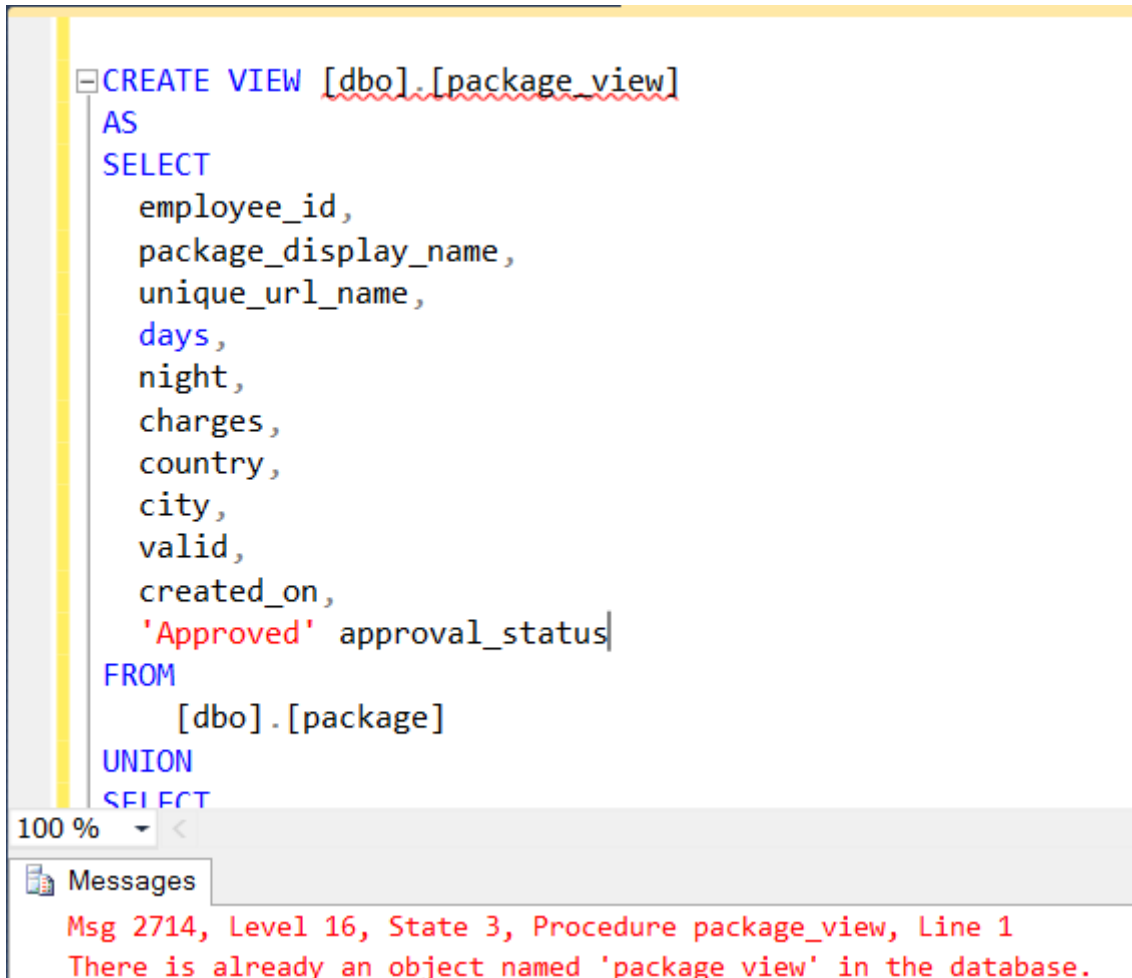
91 %

Messages

Command(s) completed successfully.

2. INSTEAD OF trigger used for authentication before placing record in parent table.

To perform this we created a view package_view which consist of records of package table along with the column approved or not approved.



```
CREATE VIEW [dbo].[package_view]
AS
SELECT
    employee_id,
    package_display_name,
    unique_url_name,
    days,
    night,
    charges,
    country,
    city,
    valid,
    created_on,
    'Approved' approval_status
FROM
    [dbo].[package]
UNION
SELECT
```

100 %

Messages

Msg 2714, Level 16, State 3, Procedure package_view, Line 1
There is already an object named 'package_view' in the database.

Figure 33 - package view

There is a secondary table package_validity in which package records are inserted if the package name doesn't exist in the package table. An INSTEAD OF trigger named package_trigger is performing this job. As soon as records are inserted in the view trigger is fired which places the record in package_validity table if it is not found in package table. The purpose of this trigger for authentication. If in case a new record is inserted in package table. It needs to be authenticated and validated. If everything is fine then it is inserted into the main table.

```
package_validity_t...anik\Manik325 (58)) x SQLQuery23.sql - M...nik\Manik325 (57))
CREATE TABLE [dbo].[package_validity] (
    package_id INT IDENTITY PRIMARY KEY,
    [employee_id] [varchar](40) NOT NULL,
    [package_display_name] [varchar](100) NOT NULL,
    [unique_url_name] [varchar](50) NOT NULL,
    [days] [int] NOT NULL,
    [night] [int] NOT NULL,
    [charges] [varchar](30) NOT NULL,
    [country] [varchar](30) NOT NULL,
    [city] [varchar](30) NOT NULL,
    [valid] [datetime] NOT NULL,
    [itinerary] [xml] NULL,
    [created_on] [datetime] NOT NULL DEFAULT (getdate()),
    UNIQUE ([unique_url_name])
)
100 %
Messages
Msg 2714, Level 16, State 6, Line 1
There is already an object named 'package_validity' in the database.
```

Figure 34 - Package Caption

INSTEAD OF trigger is shown below:

```
GO
ALTER TRIGGER [dbo].[package_trigger]
ON [dbo].[package_view]
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO [dbo].[package_validity] (
        employee_id,
        package_display_name,
        unique_url_name,
        days,
        night,
        charges,
        country,
        city,
        valid,
        created_on
    )
100 %
Messages
Command(s) completed successfully.
```

Figure 35 - package trigger

5.5 VIEWS

Views are used to create a virtual table. If we want to hide the data implementation from outside world views are generally used. If there is a requirement to join multiple tables and gives access to the user for specific fields from the table. Then as a general practice, a view is created and access of view is given to the customer so they would know about internal tables.

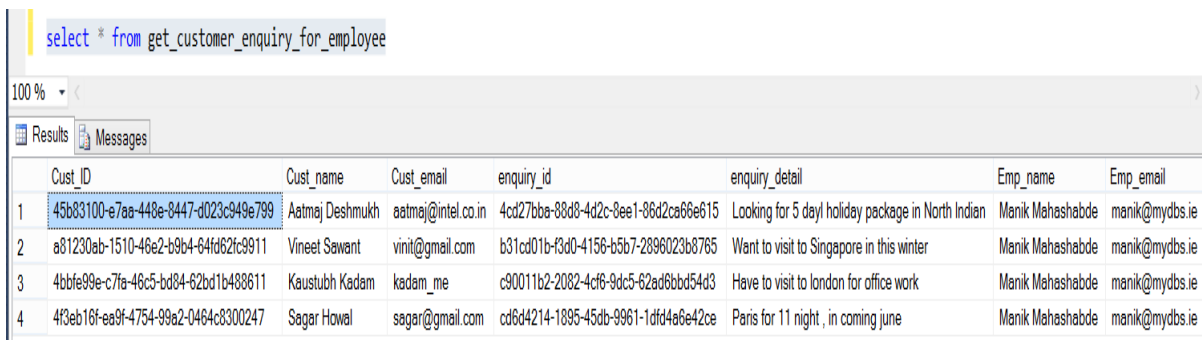
We have created two views for our database. (stevestein,2017)

1. In the first view `get_customer_enquiry_for_employee`, we are joining three tables customer, enquiry and employee.

```
ALTER VIEW [dbo].[get_customer_enquiry_for_employee]
AS
SELECT
    c.customer_id AS Cust_ID,
    c.name AS Cust_name,
    c.email AS Cust_email,
    e.enquiry_id,
    e.enquiry_detail,
    ee.name AS Emp_name,
    ee.email AS Emp_email
FROM
    [dbo].[customer] AS c |
INNER JOIN [dbo].[enquiry] AS e
    ON c.customer_id=e.customer_id
INNER JOIN [dbo].[employee] AS ee
    ON e.employee_id=ee.employee_id;
GO
```

Figure 36 - `get_customer_enquiry_for_employee`

If we select from the view result can be seen below.



```
select * from get_customer_enquiry_for_employee
```

	Cust_ID	Cust_name	Cust_email	enquiry_id	enquiry_detail	Emp_name	Emp_email
1	45b83100-e7aa-448e-8447-d023c949e799	Aatmaj Deshmukh	aatmaj@intel.co.in	4cd27bba-88d8-4d2c-8ee1-86d2ca66e615	Looking for 5 dayl holiday package in North Indian	Manik Mahashabde	manik@mydbs.ie
2	a81230ab-1510-46e2-b9b4-64fd62fc9911	Vineet Sawant	vinit@gmail.com	b31cd01b-f3d0-4156-b5b7-2896023b8765	Want to visit to Singapore in this winter	Manik Mahashabde	manik@mydbs.ie
3	4bbfe99e-c7fa-46c5-bd84-62bd1b488611	Kaustubh Kadam	kadam_me	c90011b2-2082-4cf6-9dc5-62ad6bbd54d3	Have to visit to london for office work	Manik Mahashabde	manik@mydbs.ie
4	4f3eb16f-ea9f-4754-99a2-0464c8300247	Sagar Howal	sagar@gmail.com	cd6d4214-1895-45db-9961-1dfd4a6e42ce	Paris for 11 night , in coming june	Manik Mahashabde	manik@mydbs.ie

Purpose of this view is to give access to the customer about the enquiry raised by them and which employee is handling that enquiry request.

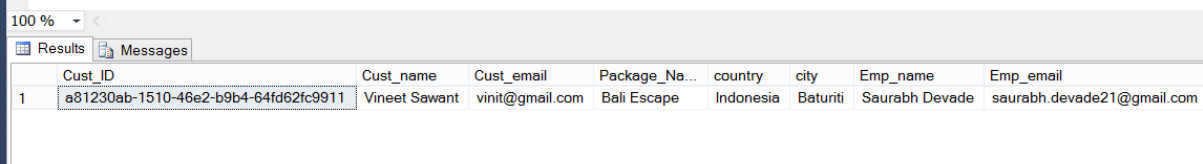
2. In this view [get_customer_packageBooking_CreatedBy_employee] we are creating a view by joining customer package, booking, employee as seen below.

```
ALTER VIEW [dbo].[get_customer_packageBooking_createdBy_employee]
AS
SELECT
    c.customer_id AS Cust_ID,
    c.name AS Cust_name,
    c.email AS Cust_email,
    p.package_display_name AS Package_Name,
    p.country,
    p.city,
    ee.name AS Emp_name,
    ee.email AS Emp_email
FROM
    [dbo].[customer] AS c
INNER JOIN [dbo].[booking] AS b
    ON c.customer_id=b.customer_id
INNER JOIN [dbo].[package] AS p
    ON b.package_id=p.package_id
INNER JOIN [dbo].[employee] AS ee
    ON p.employee_id=ee.employee_id;
GO
```

Figure 37 - Get Customer PackageBooking CreatedBy Employee

This purpose of this view to give customers the access so that they can view their booking and package details assigned to an employee of the organization as shown below.

```
select * from [get_customer_packageBooking_createdBy_employee]
```



	Cust_ID	Cust_name	Cust_email	Package_Na...	country	city	Emp_name	Emp_email
1	a81230ab-1510-46e2-b9b4-64fd62fc9911	Vineet Sawant	vinit@gmail.com	Bali Escape	Indonesia	Baturiti	Saurabh Devade	saurabh.devade21@gmail.com

8. Conclusion

Finally, we conclude that the Use of SQL SERVER is preferable for so many reasons. One of the reason is, it gives support to the XML datatype which helps to keep data in a well-structured manner and also reduces the number of rows in a table. Also, the view feature provide by SQL server helps you in abstraction of data. i.e you can show only the required data to the users who shouldn't be seeing any high sensitive data, you show them just virtual created tables.

Triggers help you to detect some events which allows you to generate audit reports. Using stored procedure is also essential because it enhances the performance of your application and also you can perform complex operation rather than performing just simple queries.

Even though making a schema diagram takes time, it is essential for your team to have one so they will easily get an idea of how your system communicates with each other. So overall Using of SQL SERVER with proper project planning helps you in creating a well-managed Database system.

9. Innovation

For innovation, we have deployed our easy_travels database in management studio into the AWS cloud server. Then on our SQL Server Management Studio, we are creating a server named **easytraveldbcloud**. Service used for this is Azure SQL server. ("SQL Database – Cloud Database as a Service | Microsoft Azure")

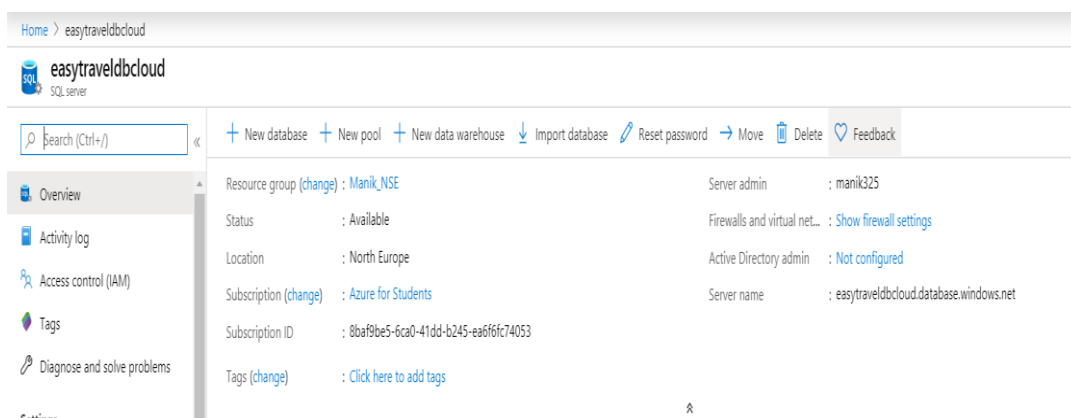
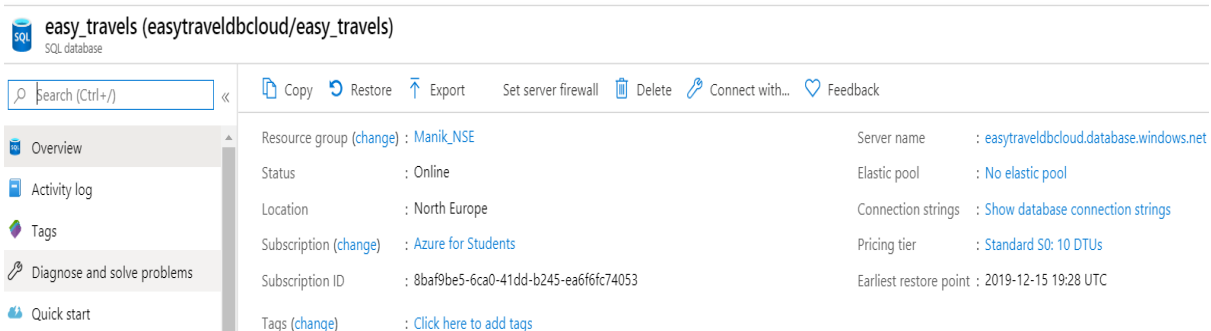


Figure 38 - Azure Clod Database

After creating and selecting hardware configuration of our server. Next step was to deploy the easy_travels into the cloud server and adding the local system IP address into firewall settings of the Azure server.



Then we can access the azure server from our management studio using the below details:

Server Name: **easytraveldbcloud.database.windows.net**

Username: **manik325**

Password: **June-2019**

After this, it can be seen that we can connect to our azure cloud server instead of a regular one which is installed on the local machine.

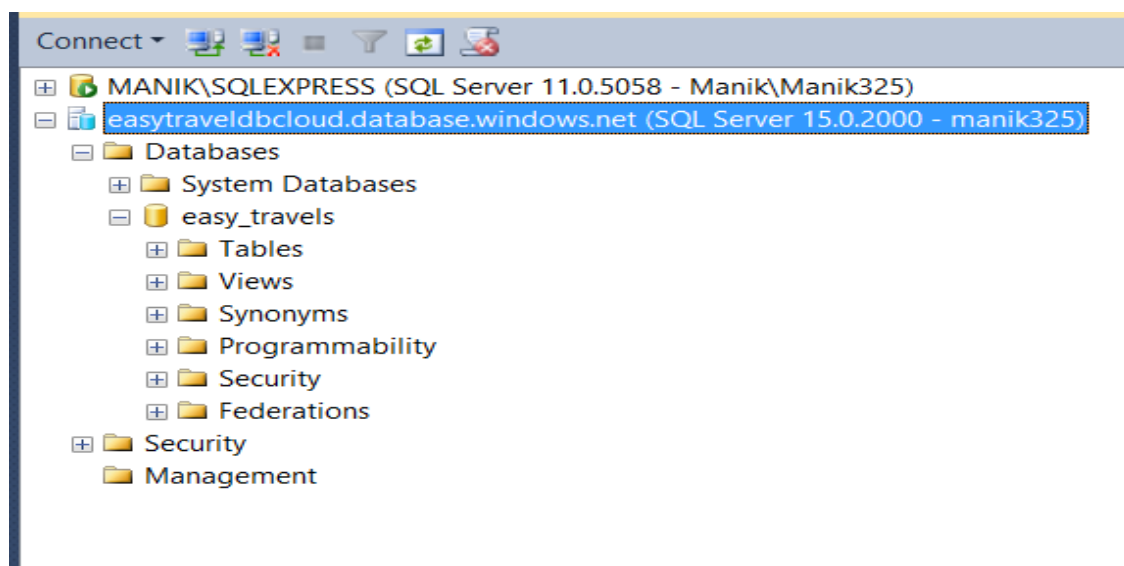


Figure 39 - Connection from management studio

10. BIBLIOGRAPHY

Checiu, L. and Ionescu, D. (2010) 'A new algorithm for mapping XML schema to XML schema', ICC-CONTI 2010 - IEEE International Joint Conferences on Computational Cybernetics and Technical Informatics, Proceedings. IEEE, pp. 625–630. doi: 10.1109/ICCCYB.2010.5491337.

Ding, L. et al. (2013) 'Research on SQL Server trigger to implement referential integrity', Proceedings of 2013 6th International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII 2013. IEEE, 3, pp. 286–288. doi: 10.1109/ICIII.2013.6703572.

Kumar, K. and Azad, S. K. (2017) 'Database normalization design pattern', 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics, UPCON 2017, 2018-Janua, pp. 318–322. doi: 10.1109/UPCON.2017.8251067.

Ramez Elmasri, and Shamkant B Navathe. Fundamentals of Database Systems. Boston, Pearson Education, 2007.

"SQL Database – Cloud Database as a Service | Microsoft Azure." Microsoft.Com, Microsoft Azure, 2019, azure.microsoft.com/en-in/services/sql-database/.

stevestein. "Views - SQL Server." Microsoft.Com, 14 Mar. 2017, docs.microsoft.com/en-us/sql/relational-databases/views/views?view=sql-server-ver15. Accessed 17 Dec. 2019.

"W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures." W3.Org, 2012, www.w3.org/TR/xmlschema11-1/. Accessed 17 Dec. 2019.

Yoon, H. S. et al. (2001) 'Developing a triggering system for real-time databases in distributed environment', Proceedings - 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2001, pp. 366–371. doi: 10.1109/ISORC.2001.922861.

11.APPENDIX A

CREATE TABLE QUERIES

1. Create Package Query

```
GO
CREATE TABLE package
(
    package_id varchar(40) not null,
    employee_id varchar(40) not null,
    package_display_name varchar(100) not null,
    unique_url_name varchar (50) not null UNIQUE,
    days int not null,
    night int not null,
    charges varchar (30) not null,
    country varchar (30) not null,
    city varchar (30) not null,
    valid DateTime not null,
    iternary xml not null,
    created_on datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT package_id_pk PRIMARY KEY(package_id),
    CONSTRAINT employee_id_fk FOREIGN KEY(employee_id)
    REFERENCES employee(employee_id)
)
```

2. Create Employee Table

```
GO
CREATE TABLE employee
(
    employee_id varchar(40) not null,
    name varchar(50) not null,
    username varchar(20) not null,
    password varchar(20) not null,
    email varchar(50) not null,
    contact_no varchar(15) not null,
    session_id varchar(40) null,
    created_on datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT employee_id_pk PRIMARY KEY(employee_id)
)
```

3. Create Customer Table

```
CREATE TABLE customer
(
    customer_id varchar(40) not null,
    name varchar(50) not null,
    username varchar(20) not null,
    password varchar(20) not null,
    email varchar(50) not null,
    contact_no varchar(15) not null,
    session_id varchar(40) null,
    created_on datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,

    CONSTRAINT customer_id_pk PRIMARY KEY(customer_id)
```


)

4. Create Enquiry Table

GO

```
CREATE TABLE enquiry(  
    enquiry_id varchar(40) NOT NULL,  
    employee_id varchar (40) NOT NULL,  
    customer_id varchar (40) NOT NULL,  
    enquiry_detail text NOT NULL,  
    enquiry_type varchar (20) NOT NULL,  
    required_days int NULL,  
    required_nights int NULL,  
    required_country varchar(50) NULL,  
    created_on datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  
    CONSTRAINT enquiry_id_pk PRIMARY KEY (enquiry_id),  
    CONSTRAINT employee_id_fk1 FOREIGN KEY(employee_id)  
    REFERENCES employee(employee_id),  
    CONSTRAINT customer_id_fk FOREIGN KEY(customer_id)  
    REFERENCES customer(customer_id)
```

)

5. Create Booking Table

GO

```
CREATE TABLE booking(  
    booking_id varchar(40) NOT NULL,  
    customer_id varchar (40) NOT NULL,  
    package_id varchar (40) NOT NULL,  
    created_on datetime NULL,  
  
    CONSTRAINT booking_id_pk PRIMARY KEY (booking_id),  
    CONSTRAINT customer_id_fk1 FOREIGN KEY(customer_id)  
    REFERENCES customer(customer_id),  
    CONSTRAINT package_id_fk FOREIGN KEY(package_id)  
    REFERENCES package(package_id)
```

)

6. Create Employee Role Table

GO

```
CREATE TABLE employee_role(  
    role_id int NOT NULL,  
    role_name varchar (30) NOT NULL,  
  
    CONSTRAINT role_id_pk PRIMARY KEY (role_id)
```

)

7. Create Employee role mapping table

GO

```
CREATE TABLE employee_role_mapping(  
    mapping_id varchar(40) NOT NULL,
```

```

employee_id varchar (40) NOT NULL,
role_id int NOT NULL,
created_on datetime NULL,
CONSTRAINT mapping_id_pk PRIMARY KEY (mapping_id),
CONSTRAINT employee_id_fk2 FOREIGN KEY(employee_id)
REFERENCES employee(employee_id),
CONSTRAINT role_id_fk FOREIGN KEY(role_id)
REFERENCES employee_role(role_id),

)

```

12. APPENDIX B

All insert queries are attached as separate sql files.

13. Individual Contribution

My contribution in the project was to collaborate with Saurabh on selecting a database model for the project. We discussed various databases to work on for exam:- departmental store, hospital, travelling. Then finally both of us came up with easy-travels database. Then we came up with what kind of business requirements should be there. I specifically came up with the idea about 3 tables employee, enquiry and customer and learned about table creation in SQL management studio. Deciding on the type of attributes in each table should be there and what kind of relation should be there between them. I also learned about the relation between 3 tables. Then Saurabh and I read through CA and decided what kind of business requirements should be there and later came with 3 stored procedures for their implementation. I worked on **employee login stored procedure** and worked on **creating session token function, get all customer booking stored procedure** and **search for specific event stored procedure**. I worked on searching about xml field and suggested about kind of xml field we can keep in our project which was later continued by Saurabh. Then later I worked on creating **customer enquiry for employees view** and learned about joining tables and data abstraction for clients. After this I worked on trigger can came up with **INSTEAD OF trigger** where I created a view for storing package table data and creating a secondary table for package validity. I learned and implemented that whenever a new package is inserted in the view so before moving it to main package table I should be inserted in package_validity table by the trigger for authentication purpose. Then me and Sourabh worked on schema diagram design where we studied about normalization is done to 3NF and various integrity constraints for our database. After that we designed the schema on draw.io and identified about weak and strong entity in the relation. Also identified about various types of relationship in the table. Once everything was completed and we had a running database in the management studio and all the functionalities which me and Sourabh added were working correctly. The final step was to introduce innovation in the project. We thought of

introducing cloud computing in our project where we created a SQL server in AZURE and later on deployed our local database in the cloud from management studio. The reason for adding cloud is that anyone can access the database server from their local system. We first need to add the client IP into our azure firewall then using the credentials given by us in the report anyone can connect to the server and view our database. Overall, I learned about various functionalities of a database system as taught in the class and later on researched more about them on the internet and also by relating to my previous work experience and how I can efficiently use them for the given CA. I am very sure that these new skills which I learned are going to help me to become a good backend developer in my career.