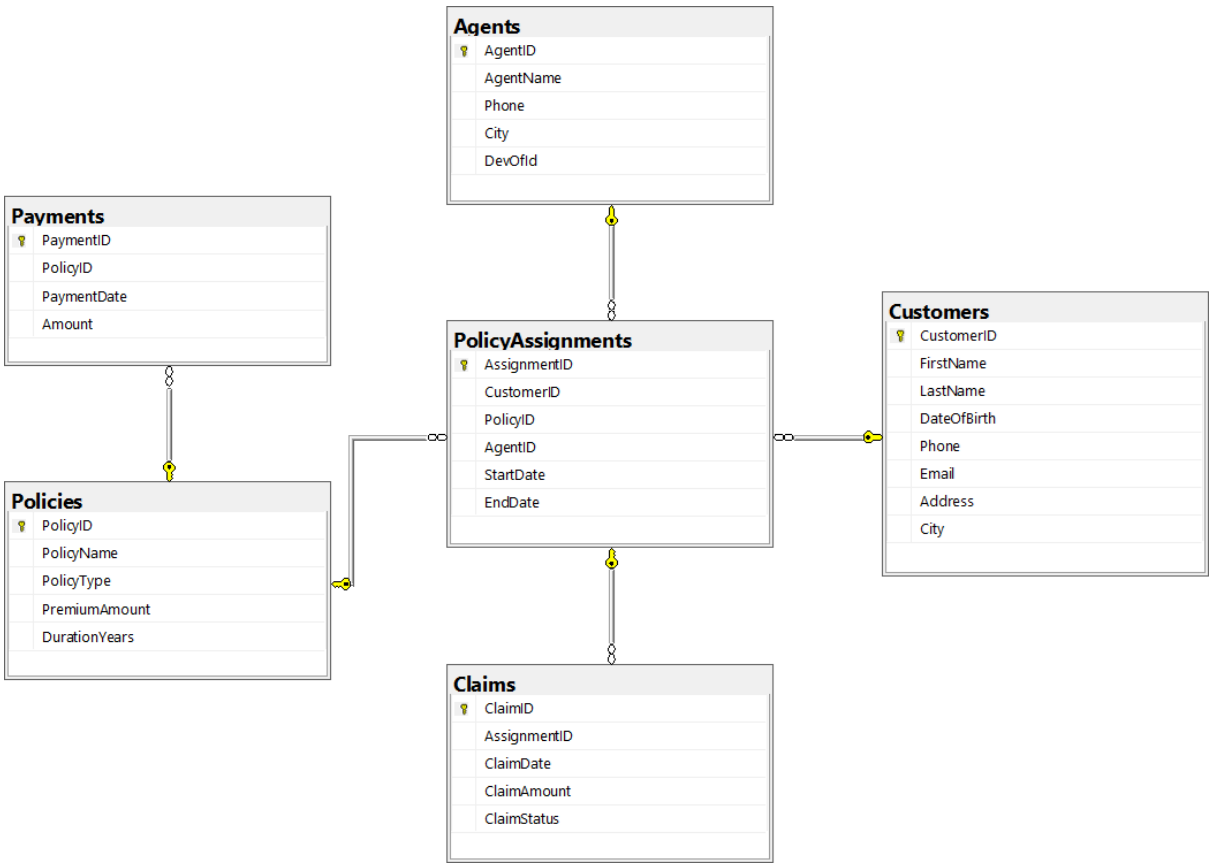
Module 4.4 Practical Project Assignment

1)Create insurance Database:

```
create  database insurancedb;

use insurancedb;
```

2) Schema diagram:



3)Create Table Commands:

Customers Table:

```
CREATE TABLE Customers (
CustomerID INT PRIMARY KEY IDENTITY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
DateOfBirth DATE,
Phone VARCHAR(50),
```

```
Email VARCHAR(50));
```

Policies Table:

```
CREATE TABLE Policies (
PolicyID INT PRIMARY KEY IDENTITY,
PolicyName VARCHAR(50),
PolicyType VARCHAR(50),
PremiumAmount INT,
DurationYears INT);
```

Agents Table:

```
CREATE TABLE Agents (
AgentID INT PRIMARY KEY IDENTITY,
AgentName VARCHAR(50),
Phone VARCHAR(10),
City VARCHAR(30));
```

Policy Assignment Table:

```
CREATE TABLE PolicyAssignments (
AssignmentID INT PRIMARY KEY IDENTITY,
CustomerID INT,
PolicyID INT,
AgentID INT,
StartDate DATE,
EndDate DATE,
CONSTRAINT customer_fk FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID),
CONSTRAINT policy_fk FOREIGN KEY (PolicyID) REFERENCES Policies(PolicyID),
CONSTRAINT agent_fk FOREIGN KEY (AgentID) REFERENCES Agents(AgentID));
```

Claims Table:

```
CREATE TABLE Claims (
ClaimID INT PRIMARY KEY,
AssignmentID INT,
ClaimDate DATE,
ClaimAmount DECIMAL(10,2),
ClaimStatus VARCHAR(10),
CONSTRAINT fk_pa_Claims
FOREIGN KEY (AssignmentID) REFERENCES PolicyAssignments(AssignmentID));
```

Payments table:

```
create table Payments(
PaymentID int primary key identity,
PolicyID int,
PaymentDate date,
Amount decimal
constraint Payments_policies foreign key(PolicyID) references
Policies(PolicyID));
```

4)Insert Commands:

```
INSERT INTO Customers (FirstName, LastName, DateOfBirth, Phone, Email)
VALUES
('Karthik', 'Mehra', '1994-08-14', '9812345601', 'karthik.mehra@gmail.com'),
('Pooja', 'Nair', '1998-02-21', '9823456702', 'pooja.nair@gmail.com'),
('Rohan', 'Malik', '1992-06-11', '9834567803', 'rohan.malik@gmail.com'),
('Ananya', 'Iyer', '1996-09-30', '9845678904', 'ananya.iyer@gmail.com'),
('Siddharth', 'Jain', '1995-01-19', '9856789015', 'siddharth.jain@gmail.com'),
('Meera', 'Kulkarni', '2001-05-07', '9867890126', 'meera.k@gmail.com');

INSERT INTO Policies (PolicyName, PolicyType, PremiumAmount, DurationYears)
VALUES
('Secure Life Plus', 'Life Insurance', 18000, 25),
('MediCare Gold', 'Health Insurance', 20000, 10),
('Auto Protect', 'Car Insurance', 9000, 7),
('Marine Shield', 'Boat Insurance', 14000, 12),
('Bike Guard', 'Vehicle Insurance', 4500, 3),
('Home Care', 'Property Insurance', 12000, 20),
('Home Elite', 'Property Insurance', 25000, 30),
('Travel Guard', 'Travel Insurance', 3500, 1),
('Child Secure', 'Life Insurance', 24000, 20),
('Pension Plan', 'Life Insurance', 27000, 35),
('Disability Cover', 'Disability Insurance', 15000, 2);

INSERT INTO Agents (AgentName, Phone, City)
VALUES
('Ramesh Verma', '9871112233', 'Mumbai'),
('Kavya Rao', '9882223344', 'Hyderabad'),
('Aakash Singh', '9893334455', 'Pune'),
('Nitin Chawla', '9904445566', 'Noida'),
('Divya Shah', '9915556677', 'Ahmedabad');

INSERT INTO PolicyAssignments (CustomerID, PolicyID, AgentID, StartDate, EndDate)
VALUES
(1, 1, 1, '2021-04-01', '2046-04-01'),
(2, 2, 2, '2023-01-10', '2033-01-10'),
(3, 4, 4, '2024-03-15', '2036-03-15'),
(4, 5, 3, '2022-11-20', '2025-11-20'),
(5, 7, 5, '2023-07-01', '2053-07-01'),
(1, 3, 2, '2024-02-05', '2031-02-05'),
(6, 9, 1, '2023-09-18', '2043-09-18');

INSERT INTO Claims (ClaimID, AssignmentID, ClaimDate, ClaimAmount, ClaimStatus)
VALUES
(1, 1, '2022-08-12', 60000, 'Approved'),
(2, 2, '2024-05-09', 35000, 'Pending'),
(3, 3, '2025-01-20', 80000, 'Approved'),
(4, 4, '2023-12-01', 25000, 'Rejected'),
(5, 1, '2024-10-14', 30000, 'Approved'),
(6, 6, '2025-06-22', 18000, 'Pending');

INSERT INTO Payments (PolicyID, PaymentDate, Amount)
VALUES
(1, '2022-04-05', 18000),
(1, '2023-04-05', 18000),
(2, '2023-01-10', 20000),
(2, '2024-01-10', 20000),
(3, '2024-02-05', 9000),
(4, '2024-03-15', 14000),
(5, '2023-11-20', 4500),
(6, '2023-07-01', 12000),
```

```
(7, '2023-07-01', 25000),
(8, '2023-12-20', 3500),
(9, '2023-09-18', 24000),
(10, '2023-05-30', 27000),
(11, '2023-08-08', 15000);
```

5)Queries:

A) Comparison Operators (=, <>, >, =, <=):

```
-- find all agents who live in delhi
select * from agents where city = 'Delhi';

-- display policies with premiumamount greater than 25000
select policyname from policies where premiumamount > 25000;

-- list claims where claimamount is less than 50000
select * from claims where claimamount < 50000;

-- find customers whose state is not maharashtra
select * from customers where state <> 'Maharashtra';

-- display policies that expire on or before 2025-12-31
select p.policyname from policies p join policyassignments a on p.policyid =
a.policyid where a.enddate <= '2025-12-31';

-- retrieve agents who joined after 1st jan 2020 (assuming joindate column
exists)
select * from agents where joindate > '2020-01-01';
```

B) Logical Operators (AND, OR, NOT):

```
-- find policies where policytype is health and premiumamount greater than 20000
select * from policies where policytype = 'Health' and premiumamount > 20000;

-- list customers who live in mumbai or pune
select * from customers where city in ('Mumbai', 'Pune');

-- display claims where claimstatus is not approved
select * from claims where claimstatus <> 'Approved';

-- find active policies with premiumamount less than 15000 (assuming active means
enddate >= current date)
select * from policies where premiumamount < 15000 and enddate >= getdate();

-- retrieve customers who are male and live in bangalore (assuming gender column
exists)
select * from customers where gender = 'Male' and city = 'Bangalore';

-- show agents who joined before 2019 or live in chennai (assuming joindate
column exists)
select * from agents where joindate < '2019-01-01' or city = 'Chennai';
```

C) IN Operator:

```sql
-- display customers whose city is in delhi, mumbai, chennai
select * from customers where city in ('Delhi', 'Mumbai', 'Chennai');

-- find policies with policytype in life or health
select policyname from policies where policytype in ('Life', 'Health');

-- retrieve claims where claimstatus is rejected or pending
select * from claims where claimstatus in ('Rejected', 'Pending');

-- list payments made using upi or credit card (assuming paymentmode column
exists)
select * from payments where paymentmode in ('UPI', 'Credit Card');
```

D) BETWEEN Operator:

```sql
-- find policies with premiumamount between 10000 and 30000
select * from policies where premiumamount between 10000 and 30000;

-- retrieve claims made between 2024-01-01 and 2024-12-31
select * from claims where claimdate between '2024-01-01' and '2024-12-31';

-- list customers born between 1985-01-01 and 1995-12-31
select * from customers where dateofbirth between '1985-01-01' and '1995-12-31';

-- display payments where amount is between 5000 and 20000
select * from payments where amount between 5000 and 20000;
```

E) LIKE Operator (Pattern Matching):

```sql
-- find customers whose firstname starts with 'a'
select * from customers where firstname like 'A%';

-- display agents whose agentname ends with 'sharma'
select * from agents where agentname like '%Sharma';

-- retrieve customers whose lastname contains 'singh'
select * from customers where lastname like '%Singh%';

-- list cities from customers where city starts with 'b'
select city from customers where city like 'B%';

-- find policies where policytype contains the word 'vehicle'
select * from policies where policytype like '%Vehicle%';
```

F) IS NULL / IS NOT NULL:

```sql
-- display policies where enddate is null
select p.policyname from policies p join policyassignments a on p.policyid =
a.policyid where a.enddate is null;

-- find claims where claimstatus is not null
```

```sql
select * from claims where claimstatus is not null;


-- retrieve customers where state is null
select * from customers where state is null;
```

G)Mixed / Tricky Scenarios:

```sql
/* Find policies where PolicyType is Health
and PremiumAmount is between 15,000 and 40,000 */
SELECT *
FROM Policies
WHERE PolicyType = 'Health Insurance'
AND PremiumAmount BETWEEN 15000 AND 40000;

/* Display claims where ClaimAmount is greater than 1,00,000
and ClaimStatus is not Rejected */
SELECT *
FROM Claims
WHERE ClaimAmount > 100000
AND ClaimStatus <> 'Rejected';
```

FUNCTION QUERIES:

A)STRING FUNCTION QUESTIONS:

```sql
/* 1. Display customer full name */
SELECT FirstName + ' ' + LastName AS FullName
FROM Customers;

/* 2. FirstName starts with 'A' */
SELECT *
FROM Customers
WHERE FirstName LIKE 'A%';

/* 3. LastName in uppercase */
SELECT UPPER(LastName) AS LastName_Upper
FROM Customers;

/* 4. LastName length more than 6 */
SELECT *
FROM Customers
WHERE LEN(LastName) > 6;

/* 5. First 3 characters of FirstName */
SELECT LEFT(FirstName, 3) AS First3Chars
FROM Customers;

/* 6. Replace 'Life' with 'Term Life' in PolicyType */
SELECT REPLACE(PolicyType, 'Life', 'Term Life') AS UpdatedPolicyType
FROM Policies;

/* 7. City contains 'del' (case-insensitive) */
SELECT *
FROM Customers
```

```sql
WHERE LOWER(City) LIKE '%del%';


/* 8. Remove leading and trailing spaces from FirstName */
SELECT LTRIM(RTRIM(FirstName)) AS CleanFirstName
FROM Customers;

/* 9. PolicyType and its character length */
SELECT PolicyType, LEN(PolicyType) AS Length
FROM Policies;

/* 10. Last 4 characters from PolicyType */
SELECT RIGHT(PolicyType, 4) AS Last4Chars
FROM Policies;
```

B) NUMERIC FUNCTION QUESTIONS:

```sql
-- display premiumamount rounded to nearest integer
select round(premiumamount, 0) as rounded_premium from policies;

-- find policies where premiumamount is greater than 5000 after rounding
select * from policies where round(premiumamount, 0) > 5000;

-- display absolute value of claimamount
select abs(claimamount) as absolute_claimamount from claims;

-- calculate 10 percent tax on premiumamount
select premiumamount, premiumamount * 0.10 as tax_amount from policies;

-- display premiumamount rounded to 2 decimal places
select round(premiumamount, 2) as rounded_premium from policies;

-- find the square root of premiumamount
select sqrt(premiumamount) as sqrt_premium from policies;

-- display the ceiling value of claimamount
select ceiling(claimamount) as ceiling_claimamount from claims;

-- display the floor value of premiumamount
select floor(premiumamount) as floor_premium from policies;

-- calculate total premium including 18 percent gst
select premiumamount, premiumamount * 1.18 as total_with_gst from policies;

-- find claims where claimamount modulo 2 equals 0
select * from claims where claimamount % 2 = 0;
```

C) DATE FUNCTION QUESTIONS:

```sql
-- display current system date
select getdate() as current_date;

-- find policies that started in the year 2023
select * from policies where year(startdate) = 2023;
```

```sql
-- calculate policy duration in days
select policyid, datediff(day, startdate, enddate) as duration_days from
policies;

-- find customers whose birthday is in the current month
select * from customers where month(dob) = month(getdate());

-- display claimdate in dd-mm-yyyy format
select format(claimdate, 'dd-MM-yyyy') as formatted_claimdate from claims;

-- find policies that expire within the next 30 days
select * from policies where enddate between getdate() and dateadd(day, 30,
getdate());

-- calculate customer age from dob
select customerid, datediff(year, dob, getdate()) as age from customers;

-- find claims filed in the last 6 months
select * from claims where claimdate >= dateadd(month, -6, getdate());

-- display month name from policy startdate
select policyid, datename(month, startdate) as start_month from policies;

-- find policies where startdate is a monday
select * from policies where datename(weekday, startdate) = 'Monday';
```

D) AGGREGATE FUNCTION QUESTION:

```sql
-- find the total number of customers
select count(*) as total_customers from customers;

-- calculate total premium collected
select sum(premiumamount) as total_premium_collected from policies;

-- find average premium amount
select avg(premiumamount) as average_premium from policies;

-- find the maximum claim amount
select max(claimamount) as max_claim_amount from claims;

-- find the minimum premium amount
select min(premiumamount) as min_premium from policies;

-- count number of policies per policytype
select policytype, count(*) as policy_count from policies group by policytype;

-- find total claimamount per claimstatus
select claimstatus, sum(claimamount) as total_claimamount from claims group by
claimstatus;

-- find policytypes where average premium is greater than 6000
select policytype from policies group by policytype having avg(premiumamount) >
6000;

-- count number of claims per year
select year(claimdate) as claim_year, count(*) as claim_count from claims group
by year(claimdate);

-- find customers having more than 2 policies
```

```sql
select customerid, count(*) as policy_count from policies group by customerid
having count(*) > 2;
```

Queries On Joins:

```sql
-- list all policies for customerid 5
select p.policyname from policies p join policyassignments pa on p.policyid =
pa.policyid where pa.customerid = 5;

-- view all customers with their policies
select c.customerid, c.firstname + ' ' + c.lastname as customername, p.policyname
from customers c join policyassignments pa on c.customerid = pa.customerid join
policies p on p.policyid = pa.policyid;

-- view claims with customer name
select cl.claimid, cu.firstname + ' ' + cu.lastname as customername from
customers cu join policyassignments pa on cu.customerid = pa.customerid join
claims cl on cl.assignmentid = pa.assignmentid;

-- display firstname, policyname, agentname, startdate and enddate
select cu.firstname, p.policyname, a.agentname, pa.startdate, pa.enddate from
customers cu join policyassignments pa on cu.customerid = pa.customerid join
policies p on p.policyid = pa.policyid join agents a on a.agentid = pa.agentid;

-- display claims report with firstname, policyname, claimamount, claimstatus and
claimdate
select c.firstname, p.policyname, cl.claimamount, cl.claimstatus, cl.claimdate
from customers c join policyassignments pa on c.customerid = pa.customerid join
policies p on p.policyid = pa.policyid join claims cl on cl.assignmentid =
pa.assignmentid;

-- display records of customers with or without policies
select c.firstname, p.policyname from customers c left join policyassignments pa
on c.customerid = pa.customerid left join policies p on p.policyid = pa.policyid;

-- display all customers with no claims
select distinct c.firstname from customers c join policyassignments pa on
c.customerid = pa.customerid left join claims cl on cl.assignmentid =
pa.assignmentid where cl.claimid is null;

-- show customer name with total claim amount per customer
select c.firstname, sum(cl.claimamount) as totalclaim from customers c join
policyassignments pa on c.customerid = pa.customerid join claims cl on
cl.assignmentid = pa.assignmentid group by c.firstname;

-- show customers with total claim amount greater than 50000 using having
select c.firstname, sum(cl.claimamount) as totalclaim from customers c join
policyassignments pa on c.customerid = pa.customerid join claims cl on
cl.assignmentid = pa.assignmentid group by c.firstname having sum(cl.claimamount)
> 50000;

-- display agent wise policy count
select a.agentname, count(pa.policyid) as policycount from agents a left join
policyassignments pa on a.agentid = pa.agentid group by a.agentname;
```

SubQueries:

A) Subqueries using EXISTS :

```sql
-- find customers who have at least one policy
select * from customer c where exists (select 1 from policy p where p.customerid
= c.customerid);

-- list customers who have made at least one claim
select * from customer c where exists (select 1 from policy p join claim cl on
p.policyid = cl.policyid where p.customerid = c.customerid);

-- find policies for which at least one claim exists
select * from policy p where exists (select 1 from claim c where c.policyid =
p.policyid);

-- display agents who handle at least one policy
select * from agent a where exists (select 1 from policyagent pa where pa.agentid
= a.agentid);

-- display customers who do not have any claims
select * from customer c where not exists (select 1 from policy p join claim cl
on p.policyid = cl.policyid where p.customerid = c.customerid);
```

B) subqueries using any:

```sql
-- find policies with premium greater than any premium of policies held by
customerid 101
select * from policy where premiumamount > any (select premiumamount from policy
where customerid = 101);

-- list customers whose policy premium is greater than any premium in health
policy type
select distinct c.* from customer c join policy p on c.customerid = p.customerid
where p.premiumamount > any (select premiumamount from policy where policytype =
'Health');

-- find claims where claim amount is greater than any claim made in 2024
select * from claim where claimamount > any (select claimamount from claim where
year(claimdate) = 2024);

-- display policies whose premium is less than any premium of policies issued
after 2023
select * from policy where premiumamount < any (select premiumamount from policy
where startdate > '2023-12-31');

-- list agents who handle policies whose premium is greater than any premium of
life policies
select distinct a.* from agent a join policyagent pa on a.agentid = pa.agentid
join policy p on p.policyid = pa.policyid where p.premiumamount > any (select
premiumamount from policy where policytype = 'Life');
```

C) subqueries using all:

```sql
-- find policies with premium greater than all premiums of motor policies
```

```sql
select * from policy where premiumamount > all (select premiumamount from policy
where policytype = 'Motor');


-- list customers whose policy premium is less than all premiums of life policies
select distinct c.* from customer c join policy p on c.customerid = p.customerid
where p.premiumamount < all (select premiumamount from policy where policytype =
'Life');

-- find claims where claim amount is greater than all claims made in 2023
select * from claim where claimamount > all (select claimamount from claim where
year(claimdate) = 2023);

-- display policies whose premium is less than all premiums of policies having
claims
select * from policy where premiumamount < all (select p2.premiumamount from
policy p2 join claim c on p2.policyid = c.policyid);

-- find customers whose policy premium is greater than all premiums of customers
having claims
select distinct c.* from customer c join policy p on c.customerid = p.customerid
where p.premiumamount > all (select p2.premiumamount from policy p2 join claim cl
on p2.policyid = cl.policyid);
```

Set Operations:

```sql
-- customers who bought life or health policies
select customerid from policies where policytype = 'Life' union select customerid
from policies where policytype = 'Health';

-- customers who bought both life and health policies
select customerid from policies where policytype = 'Life' intersect select
customerid from policies where policytype = 'Health';

-- customers who bought life but not health policies
select customerid from policies where policytype = 'Life' except select
customerid from policies where policytype = 'Health';

-- all customers with policies allowing duplicates
select customerid from policies union all select customerid from policies;

-- policies having claims or assignments
select policyid from policyassignments union select policyid from claims;
```

CASE…ELSE in SQL Server:

```sql
-- classify policies based on premium amount
select policyname,
case
  when premiumamount < 5000 then 'low premium'
  when premiumamount between 5000 and 15000 then 'medium premium'
  else 'high premium'
end as premium_category
from policies;
```

Merge in SQL Server:

```
-- create staging table for agents
create table agents_staging (
    agentid int,
    agentname varchar(50),
    phone varchar(10),
    city varchar(30)
);

-- insert data into staging table
insert into agents_staging (agentid, agentname, phone, city) values
(1, 'ravi kumar', '9998887776', 'hyderabad'),
(2, 'neha singh', '8887776665', 'bangalore'),
(6, 'anil sharma', '7776665554', 'delhi');


-- merge staging data into main agents table
merge agents as target
using agents_staging as source
on target.agentid = source.agentid
when matched then
    update set
        target.agentname = source.agentname,
        target.phone = source.phone,
        target.city = source.city
when not matched then
    insert (agentname, phone, city)
    values (source.agentname, source.phone, source.city);
```

ROLLUP in SQL Server:

```
-- rollup with two columns showing subtotals and grand total
select policytype, durationyears, sum(premiumamount) as total_premium
from policies
group by rollup (policytype, durationyears);
```

Cube in SQL Server:

```
-- cube with two columns showing all possible subtotals
select policytype, durationyears, sum(premiumamount) as total_premium
from policies
group by cube (policytype, durationyears);
```

Grouping Sets in SQL Server:

```sql
-- grouping sets to show selected subtotals explicitly
select policytype, durationyears, sum(premiumamount) as total_premium
from policies
group by grouping sets (
  (policytype, durationyears),
  (policytype),
  (durationyears),
  ()
);
```