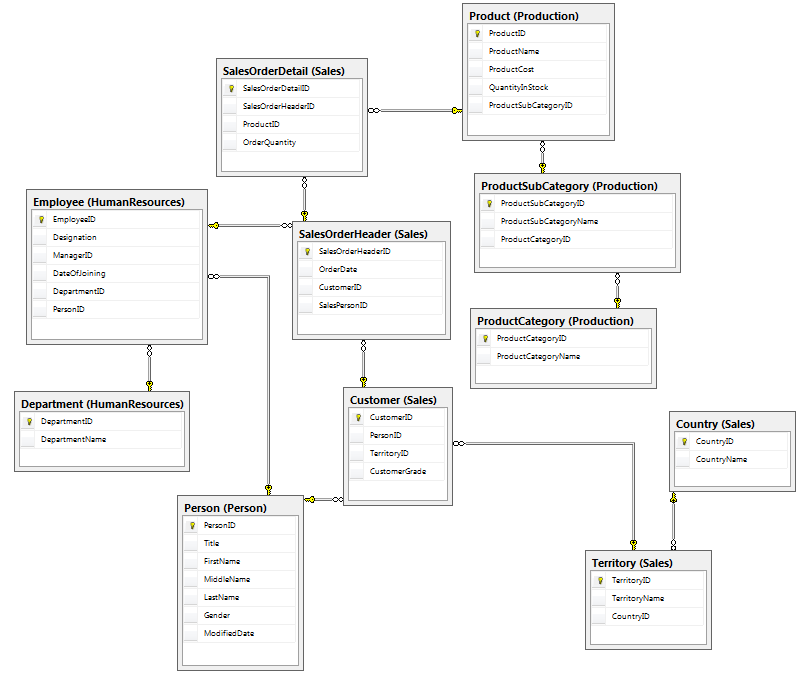
MS SQL SERVER – LABS

1. Create Physical Data Model as per the diagram below. Identify and apply correct datatypes and constraints for the columns of each table

  
Solution:

CREATE SCHEMA production;

CREATE SCHEMA sales;

CREATE SCHEMA HumanResources;

CREATE TABLE production.product(

productId INT PRIMARY KEY IDENTITY(1,1),

productName VARCHAR(20) NOT NULL,

productCost MONEY NOT NULL,

quantityInStock INT NOT NULL,

productSubCategoryId INT REFERENCES

production.productSubCategory(productSubCategoryId));

CREATE TABLE production.productSubCategory(

productSubCategoryId INT PRIMARY KEY,

productSubCategoryName VARCHAR(20) NOT NULL,

productCategoryId INT REFERENCES production.productCategory(productCategoryId));

CREATE TABLE production.productCategory(

productCategoryId INT PRIMARY KEY,

productCategoryName VARCHAR(20));

CREATE TABLE sales.SalesOrderDetail(

SalesOrderDetailID INT PRIMARY KEY,

SalesOrderHeaderID INT REFERENCES sales.SalesOrderHeader(SalesOrderHeaderID),

ProductID INT REFERENCES production.product(productId),

OrderQuantity INT NOT NULL);

CREATE TABLE sales.SalesOrderHeader(

SalesOrderHeaderID INT PRIMARY KEY,

OrderDate INT NOT NULL,

CustomerID INT REFERENCES sales.Customer(CustomerID),

SalesPersonID INT NOT NULL);

CREATE TABLE sales.Customer(

CustomerID INT PRIMARY KEY,

PersonID INT REFERENCES HumanResources.Person(PersonID),

TerritoryID INT sales.Territory(TerritoryID),

CustomerGrade INT);

CREATE TABLE sales.Country(

CountryID INT PRIMARY KEY,

CountryName VARCHAR(20) NOT NULL);

CREATE TABLE sales.Territory(

TerritoryID INT PRIMARY KEY,

TerritoryName VARCHAR(20) NOT NULL,

CountryID INT REFERENCES sales.Country(CountryID));

CREATE TABLE HumanResources.Employee(

EmployeeID INT PRIMARY KEY,

Designation VARCHAR(20) NOT NULL,

ManagerID INT NOT NULL,

DateOfJoining DATE,

DepartmentID INT REFERENCES HumanResources.Department(DepartmentID),

PersonID INT REFERENCES HumanResources.Person(PersonID));

CREATE TABLE HumanResources.Department(

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(20) NOT NULL);

CREATE TABLE HumanResources.Person(

PersonId INT PRIMARY KEY,

Title VARCHAR(20) NOT NULL,

FirstName VARCHAR(20) NOT NULL,

MiddleName VARCHAR(20) NOT NULL,

LastName VARCHAR(20) NOT NULL,

ModifiedDate DATE NOT NULL);

1. Restore AdventureWorks Database

Step 1: Download: <https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms> – Download AdventureWorks2019.bak

Step 2: Restore the database from backup file - <https://www.youtube.com/watch?v=mRI8vpn-tyk>

Solution: Done

1. Solve the below (Any 25)
2. Write a query that displays all the rows from the Person.Person table where the rows were modified after December 29, 2000. Display the business entity ID number, the name columns, and the modified date.

Solution:

SELECT BusinessEntityID, NameStyle ,FirstName, MiddleName, LastName, ModifiedDate

FROM Person.Person WHERE ModifiedDate >'29-Dec-2000';

1. Rewrite the query from question 1 so that it displays the rows that were not modified during December 2000.

Solution:

SELECT BusinessEntityID, FirstName, MiddleName, LastName, ModifiedDate

FROM Person.Person

WHERE ModifiedDate NOT BETWEEN '2000-12-01' AND '2000-12-31';

1. Write a query that displays the product ID and name for each product from the Production.Product table with the name starting with Chain.

Solution:

SELECT ProductID, Name FROM Production.Product

WHERE Name LIKE 'Chain%';

1. Write a query that displays the business entity ID number, first name, middle name, and last name from the Person.Person table for only those rows that have E or B stored in the middle name column.

Solution:

SELECT BusinessEntityID, FirstName, MiddleName, LastName, ModifiedDate FROM

Person.Person WHERE MiddleName IN ('E', 'B');

1. Write a query displaying the order ID, order date, and total due from the Sales.SalesOrderHeader table. Retrieve only those rows where the order was placed during the month of September 2001 and the total due exceeded $1,000.

Solution:

SELECT SalesOrderID, OrderDate, TotalDue FROM Sales.SalesOrderHeader WHERE TotalDue >

1000 AND OrderDate >= '01-Sep-2012' AND OrderDate <= '30-Sep-2012';

1. Write a query displaying the sales orders where the total due exceeds $1,000. Retrieve only those rows where the salesperson ID is 279 or the territory ID is 6.

Solution:

   SELECT \* FROM Sales.SalesOrderHeader

WHERE TotalDue > 1000 AND (SalesPersonID = 279 OR TerritoryID = 6);

1. Write a query displaying the ProductID, Name, and Color columns from rows in the Production.Product table. Display only those rows in which the color is not blue.

Solution:

SELECT ProductID, Name, Color FROM Production.Product

WHERE Color <> 'blue';

1. Write a query that returns the business entity ID and name columns from the Person.Person table. Sort the results by LastName, FirstName, and MiddleName.

Solution:

SELECT BusinessEntityID, FirstName, MiddleName, LastName, ModifiedDate

FROM Person.Person ORDER BY LastName, FirstName, MiddleName;

1. Write a query that displays in the “AddressLine1 (City PostalCode)” format from the Person.Address table.

Solution:

SELECT AddressLine1 + ' (' + City + ' ' + PostalCode + ')' FROM Person.Address;

1. Write a query using the Production.Product table displaying the product ID, color, and name columns. If the color column contains a NULL value, replace the color with ‘No Color’.

Solution:

SELECT ProductID, ISNULL(Color, 'No Color') AS [Color], Name FROM Production.Product;

1. Modify the query written in question 2 so that the description of the product is displayed in the “Name: Color” format. Make sure that all rows display a value even if the Color value is missing.

Solution:

SELECT ProductID, Name + ISNULL(': ' + Color,'') AS ProductNameWithColor FROM

Production.Product;

1. Write a query using the Sales.SpecialOffer table. Display the difference between the MinQty and MaxQty columns along with the SpecialOfferID and Description columns.

Solution:

SELECT SpecialOfferID, Description, MaxQty - MinQty AS [Difference]

FROM Sales.SpecialOffer;

1. Write a query using the Sales.SpecialOffer table that multiplies the MaxQty column by the DiscountPCT column. If the MaxQty value is null, replace it with the value 10. Include the SpecialOfferID and Description columns in the results.

Solution:

SELECT SpecialOfferID, Description, ISNULL(MaxQty,10) \* DiscountPct AS Discount FROM

Sales.SpecialOffer;

1. Write a query that displays the first 10 characters of the AddressLine1 column in the Person.Address table.

Solution:

SELECT SUBSTRING(AddressLine1,1,10) AS AddressCharFirst10

FROM Person.Address;

1. Write a query that calculates the number of days between the date an order was placed and the date that it was shipped using the Sales.SalesOrderHeader table. Include the SalesOrderID, OrderDate, and ShipDate columns.

Solution:

SELECT SalesOrderID, OrderDate, ShipDate, DATEDIFF(d,OrderDate,ShipDate) AS NumberOfDays FROM Sales.SalesOrderHeader;

1. Write a query that displays only the date, not the time, for the order date and ship date in the Sales.SalesOrderHeader table. (Use any of the styles that return only date)

Solution:

SELECT CONVERT(VARCHAR,OrderDate,1) AS OrderDate, CONVERT(VARCHAR, ShipDate,1) AS ShipDate FROM Sales.SalesOrderHeader;

1. Write a query that adds six months to each order date in the Sales.SalesOrderHeader table. Include the SalesOrderID and OrderDate columns.

Solution:

SELECT SalesOrderID, OrderDate, DATEADD(m,6,OrderDate) Plus6Months FROM Sales.SalesOrderHeader;

1. Write a query that displays the year of each order date and the numeric month of each order date in separate columns in the results. Include the SalesOrderID and OrderDate columns.

Solution:

SELECT SalesOrderID, OrderDate, YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth FROM Sales.SalesOrderHeader;

1. Write a statement that generates a random number between 1 and 10 each time it is run.

Solution:

SELECT CAST(RAND() \* 10 AS INT) + 1;

1. Write a query using the Sales.SalesOrderHeader table to display the orders placed during 2001 by using a function. Include the SalesOrderID and OrderDate columns in the results.

Solution:

SELECT SalesOrderID, OrderDate FROM Sales.SalesOrderHeader ORDER BY MONTH(OrderDate), YEAR(OrderDate);

1. Write a query using the Sales.SalesOrderHeader table listing the sales in order of the month the order was placed and then the year the order was placed. Include the SalesOrderID and OrderDate columns in the results.

Solution:

1. The HumanResources.Employee table does not contain the employee names. Join that table to the Person.Person table on the BusinessEntityID column. Display the job title, birth date, first name, and last name.

Solution:

SELECT JobTitle, BirthDate, FirstName, LastName

FROM HumanResources.Employee AS E

INNER JOIN Person.Person AS P ON E.BusinessEntityID = P.BusinessEntityID;

1. The customer names also appear in the Person.Person table. Join the Sales.Customer table to the Person.Person table. The BusinessEntityID column in the Person.Person table matches the PersonID column in the Sales.Customer table. Display the CustomerID, StoreID, and TerritoryID columns along with the name columns.

Solution:

SELECT CustomerID, StoreID, TerritoryID, FirstName, MiddleName, LastName

FROM Sales.Customer AS C

INNER JOIN Person.Person AS P ON C.PersonID = P.BusinessEntityID;

1. Write a query that joins the Sales.SalesOrderHeader table to the Sales. SalesPerson table. Join the BusinessEntityID column from the Sales.SalesPerson table to the SalesPersonID column in the Sales.SalesOrderHeader table. Display the SalesOrderID along with the SalesQuota and Bonus.

Solution:

SELECT SalesOrderID, SalesQuota, Bonus FROM Sales.SalesOrderHeader AS S

INNER JOIN Sales.SalesPerson AS SP ON S.SalesPersonID = SP.BusinessEntityID;

1. The catalog description for each product is stored in the Production.ProductModel table. Display the columns that describe the product from the Production.Product table, such as the color and size along with the catalog description for each product.

Solution:

SELECT PM.CatalogDescription, Color, Size

FROM Production.Product AS P

INNER JOIN Production.ProductModel AS PM ON P.ProductModelID = PM.ProductModelID;

1. Write a query that displays the names of the customers along with the product names that they have purchased. Hint: Five tables will be required to write this query!

Solution:

FROM Sales.Customer AS C INNER JOIN Person.Person AS P ON C.PersonID =

P.BusinessEntityID

INNER JOIN Sales.SalesOrderHeader AS SOH ON C.CustomerID = SOH.CustomerID

INNER JOIN Sales.SalesOrderDetail AS SOD ON SOH.SalesOrderID = SOD.SalesOrderID

INNER JOIN Production.Product AS Prod ON SOD.ProductID = Prod.ProductID;

1. Write a query that displays all the products along with the SalesOrderID even if an order has never been placed for that product. Join to the Sales.SalesOrderDetail table using the ProductID column.

Solution:

SELECT SalesOrderID, P.ProductID, P.Name

FROM Production.Product AS P

LEFT OUTER JOIN Sales.SalesOrderDetail AS SOD ON P.ProductID = SOD.ProductID;

1. The Sales.SalesOrderHeader table contains foreign keys to the Sales.CurrencyRate and Purchasing.ShipMethod tables. Write a query joining all three tables, making sure it contains all rows from Sales.SalesOrderHeader. Include the CurrencyRateID, AverageRate, SalesOrderID, and ShipBase columns.

Solution:

SELECT CR.CurrencyRateID, CR.AverageRate, SM.ShipBase, SalesOrderID

FROM Sales.SalesOrderHeader AS SOH

LEFT OUTER JOIN Sales.CurrencyRate AS CR ON SOH.CurrencyRateID = CR.CurrencyRateID

LEFT OUTER JOIN Purchasing.ShipMethod AS SM ON SOH.ShipMethodID = SM.ShipMethodID;

1. Write a query to determine the number of customers in the Sales.Customer table.

Solution:

SELECT COUNT(\*) AS CountOfCustomers FROM Sales.Customer;

1. Write a query using the Production.Product table that displays the minimum, maximum, and average ListPrice.

Solution:

SELECT MIN(ListPrice) AS Minimum, MAX(ListPrice) AS Maximum, AVG(ListPrice) AS Average

FROM Production.Product;

1. Write a query that shows the total number of items ordered for each product. Use the Sales.SalesOrderDetail table to write the query.

Solution:

SELECT SUM(OrderQty) AS TotalOrdered, ProductID

FROM Sales.SalesOrderDetail GROUP BY ProductID;

1. Write a query using the Sales.SalesOrderDetail table that displays a count of the detail lines for each SalesOrderID.

Solution:

SELECT COUNT(\*) AS CountOfOrders, SalesOrderID

FROM Sales.SalesOrderDetail GROUP BY SalesOrderID;

1. Write a query using the Production.Product table that lists a count of the products in each product line.

Solution:

SELECT COUNT(\*) AS CountOfProducts, ProductLine

FROM Production.Product GROUP BY ProductLine;

1. Write a query that displays the count of orders placed by year for each customer using the Sales.SalesOrderHeader table.

Solution:

SELECT CustomerID, COUNT(\*) AS CountOfSales, YEAR(OrderDate) AS OrderYear

FROM Sales.SalesOrderHeader GROUP BY CustomerID, YEAR(OrderDate);

1. Write a query that creates a sum of the LineTotal in the Sales.SalesOrderDetail table grouped by the SalesOrderID. Include only those rows where the sum exceeds 1,000.

Solution:

SELECT SUM(LineTotal) AS SumOfLineTotal, SalesOrderID

FROM Sales.SalesOrderDetail GROUP BY SalesOrderID HAVING SUM(LineTotal) > 1000;

1. Write a query that groups the products by ProductModelID along with a count. Display the rows that have a count that equals 1.

Solution:

SELECT ProductModelID, COUNT(\*) AS CountOfProducts

FROM Production.Product GROUP BY ProductModelID HAVING COUNT(\*) = 1;

1. Write a query using the Sales.SalesOrderHeader, Sales.SalesOrderDetail, and Production.Product tables to display the total sum of products by ProductID and OrderDate.

Solution:

SELECT SUM(OrderQty) SumOfOrderQty, P.ProductID, SOH.OrderDate

FROM Sales.SalesOrderHeader AS SOH

INNER JOIN Sales.SalesOrderDetail AS SOD ON SOH.SalesOrderID = SOD.SalesOrderDetailID

INNER JOIN Production.Product AS P ON SOD.ProductID = P.ProductID GROUP BY

P.ProductID, SOH.OrderDate;

1. Based on Stored Procedures
   1. An input string representing passenger data comes in a below format to a procedure.  
       Extract the data from the string and store in a temporary table. String format: “[P9001,John Roy,Male,12-Jan-2009]”
   2. Modify the Day 4 - Q1 to validate the below  
          A.    No duplicate entry for a passenger must be attempted to insert in table  
          B.    The Age of the passenger must be between 6 to 90

Solution:

CREATE TABLE #Passengers

(

Passenger INT PRIMARY KEY IDENTITY (1,1),

PassengerId varchar(20),

PassengerName VARCHAR(20),

PassengerGender varchar(20),

Birthdate Date

)

Create Procedure sp\_passengerdata

(

@pPassengerId varchar(20),

@pPassengerName varchar(20),

@pPassengerGender varchar(20),

@pBirthdate Date

)

AS

BEGIN

IF (DATEDIFF (YYYY,@pBirthdate,GetDate()) BETWEEN 6 AND 90) and @pPassengerGender

in ('Male','Female','other') and @pPassengerName is not null and @pBirthdate is

not null

BEGIN

INSERT INTO #Passengers (PassengerId,PassengerName,PassengerGender,Birthdate)

VALUES (@pPassengerId,@pPassengerName,@pPassengerGender,@pBirthdate)

END

Else

BEGIN

Print 'Incorrect Data!'

END

END

execute sp\_passengerdata 'P9003','Renna ','Female','21-Jan-2010'

execute sp\_passengerdata 'P9002','sam john','Male','12-Dec-2021'

Select \* from #Passengers;

Drop table #Passengers

drop procedure sp\_passengerdata

1. USER Defined Functions
   1. Create a Function to check and print all the prime numbers between a range defined by user.

Solution:

DECLARE

LOW number(2);

HIGH number(2);

n number(2);

m number(2);

c number(20);

BEGIN

dbms\_output.put\_line('Enter the lower and higher limit:');

LOW:=&LOW;

HIGH:=&HIGH;

FOR n IN LOW.. HIGH

LOOP

c:=0;

FOR m IN 1.. n

LOOP

IF MOD(n, m)=0 THEN

c:=c+1;

END IF;

END LOOP;

IF c<=2 THEN

dbms\_output.put\_line(n||'\n');

END IF;

END LOOP;

END;

* 1. Create a Function that takes CategoryName as a parameter and gets the products associated with that category.

Solution:

ALTER FUNCTION Production.get\_ProductDetails(@CategoryID INT)

RETURNS @table TABLE(ProductID INT,Name VARCHAR(50),ProductNumber NVARCHAR(25),

ProductModelID INY,Size NVARCHAR(5),StandardCost money,style NCHAR(2),

Color NVARCHAR(15),Weight DECIMAL(8,2), ProductCategoryID INT)

AS

BEGIN

INSERT INTO @table SELECT

prd.ProductID,prd.Name,prd.ProductNumber,prd.ProductModelID,

prd.Size, prd.StandardCost, prd.Style, prd.Color, prd.Weight,

cat.ProductCategoryID

FROM Production.Product AS prd

LEFT JOIN Production.ProductSubcategory AS sub

ON prd.ProductSubcategoryID= sub.ProductSubcategoryID

LEFT JOIN Production.ProductCategory AS cat

ON sub.ProductCategoryID=cat.ProductCategoryID WHERE

cat.ProductCategoryID=@categoryID

RETURN;

END

SELECT \* FROM Production.get\_ProductDetails(4)

1. Trigger
   1. Create a trigger to implement banking scenario. Whenever the bank balance is updated for any bank account a transaction is captured in a  
      Bank Transaction table. The Bank Transaction table acts as a source to generate bank statements.  
      **Hint: Need to create two tables (BankAccounts with some rows and BankTransactions with no records) and a trigger for Update on Balance column of BankAccounts table.**  
      Table: BankAccounts  
      Columns:  
      •    AccountID (Numeric- Autogenerated)  
      •    CustomerName  
      •    AccountType (Current / Saving)  
      •    Balance (>0)  
      •    Modified Date  
      Table: BankTransactions  
      Columns  
      •    TransactionID (Numeric- Autogenerated)  
      •    AccountID (FK)  
      •    TransactionDate  
      •    TransactionType (Debit / Credit)  
      •    TransactionAmount

Solution:

CREATE TABLE BankAccounts.BankAccounts(

AccountID INT PRIMARY KEY IDENTITY(1,1),

CustomerName VARCHAR(100) NOT NULL,

AccountType VARCHAR(10) NOT NULL CHECK (AccountType IN('Current ','Saving')),

BALANCE MONEY CHECK (BALANCE>0),

LastModified DATE DEFAULT GETDATE()

);

INSERT INTO BankAccounts.BankAccounts (CustomerName,AccountType,BALANCE) VALUES

('Tanmay', 'Saving', 6000),

('Vishal', 'Saving', 12780),

('Manish', 'Saving', 4300),

('Tejas', 'Saving', 9870),

('Pranav', 'Current', 140990),

('Kalpesh', 'Current', 160005),

('Anicket', 'Current', 1400000),

('Navoday', 'Current', 350000);

CREATE TABLE BankAccounts.BankTransactions(

TransactionID INT PRIMARY KEY IDENTITY(1,1),

AccountID INT REFERENCES BankAccounts.BankAccounts(AccountID) NOT NULL,

TransactionDate DATE CHECK (TransactionDate<=GETDATE()),

TransactionType VARCHAR(10) NOT NULL CHECK (TransactionType IN('Debit ','Credit')),

TransactionAmount MONEY CHECK (TransactionAmount>0)

)

CREATE TRIGGER update\_bankAccount

ON BankAccounts.BankTransactions

AFTER INSERT

AS

BEGIN

IF(SELECT TransactionType FROM inserted)='Credit'

UPDATE BankAccounts.BankAccounts SET BALANCE=BALANCE+(SELECT TransactionAmount FROM inserted) WHERE AccountID=(SELECT AccountID FROM inserted)

ELSE

UPDATE BankAccounts.BankAccounts SET BALANCE=BALANCE-(SELECT TransactionAmount FROM inserted) WHERE AccountID=(SELECT AccountID FROM inserted)

Print 'Balance updated in account table for the customer!'

END

INSERT INTO BankAccounts.BankTransactions (AccountID,TransactionDate,TransactionType,TransactionAmount)

VALUES (1,'02-APR-2022','Debit',1000);

INSERT INTO BankAccounts.BankTransactions (AccountID,TransactionDate,TransactionType,TransactionAmount)

VALUES (2,'02-APR-2022','Credit',1000);

SELECT \* FROM BankAccounts.BankAccounts;

SELECT \* FROM BankAccounts.BankTransactions;

1. Exception Handling
   1. Create a procedure that takes a string parameter. The input string may be a string or a numeric or NULL value.  
      Convert the string to Integer. If it cannot be converted write an exception handling section to handle the appropriate error.  
      If the string is converted to integer print Hello the input integer number of times

Solution:

CREATE PROCEDURE sp\_strException @INPUT VARCHAR(15)

AS

DECLARE @num int;

DECLARE @k int;

SET @k=0;

BEGIN TRY

SET @num = Cast(@input AS BIGINT)

WHILE(@k<@num)

BEGIN

SET @k=@k+1;

PRINT 'Hello';

END

END TRY

BEGIN CATCH

PRINT 'Cannot convert '''+@input+''' into number';

END CATCH

* 1. Create a temp table to represent employees. Design a user defined exception to handle the salary input less than 10000.

Solution:

CREATE TABLE #employee(

empID INT PRIMARY KEY IDENTITY(1,1),

name VARCHAR(30),

salary INT CHECK(salary>=10000)

);

DECLARE @name VARCHAR(10);

DECLARE @salary INT;

SET @name='Ketan';

SET @salary=50000;

BEGIN TRY

INSERT INTO #employee (name,salary)VALURES(@name,@salary)

END TRY

BEGIN CATCH

PRINT 'Salary should be greater than 10000'

END CATCH

1. Transactions and Deadlock
   1. Document your understanding and possible solutions to Deadlock concept [Hint: You may explore online]

Solution:

a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks. Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt.

Possible solutions to Deadlock:   
When a database is stuck in a deadlock, it is always better to avoid the deadlock rather than restarting or aborting the database. The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.   
One method of avoiding deadlock is using application-consistent logic, Transactions should always access the tables in some order. In this way, Transaction T1 simply waits for transaction T2 to release the lock before it begins. When transaction T2 releases the lock, Transaction T1 can proceed freely.   
Another method for avoiding deadlock is to apply both row-level locking mechanism and READ COMMITTED isolation level. However, it does not guarantee to remove deadlocks completely.