PLS SEE ALL MY TASKS AT: https://github.com/AkshayAnand2002/CLBLSQL

SQL LEVEL-B TASK

Stored Procedures

1)

```
select* from Production.Product;
select * from Production.ProductInventory;
SELECT * from Sales.SalesOrderDetail;
SELECT * FROM Sales.SpecialOfferProduct;
SELECT * from Sales.SalesOrderHeader;


EXEC InsertOrderDetails
    @OrderID = 43659,
    @ProductID = 767,
    @UnitPrice = NULL,
    @Quantity = 10,
    @Discount = 0.05;


DROP PROCEDURE InsertOrderDetails;

CREATE PROCEDURE InsertOrderDetails
    @OrderID INT,
    @ProductID INT,
    @UnitPrice DECIMAL(18, 2) = NULL,
    @Quantity INT,
    @Discount DECIMAL(5, 2) = 0
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ProductUnitPrice DECIMAL(18, 2);
    DECLARE @CurrentStock INT;
    DECLARE @ReorderLevel INT;
    DECLARE @SpecialOfferID INT;

    IF @UnitPrice IS NULL
    BEGIN
        SELECT @ProductUnitPrice = ListPrice
        FROM Production.Product
        WHERE ProductID = @ProductID;

        IF @ProductUnitPrice IS NULL
        BEGIN
            PRINT 'Invalid ProductID. Please try again.';
            RETURN;
        END
    END
    ELSE
    BEGIN
        SET @ProductUnitPrice = @UnitPrice;
    END

    SELECT @CurrentStock = Quantity
    FROM Production.ProductInventory
    WHERE ProductID = @ProductID;

    SELECT @ReorderLevel = SafetyStockLevel
    FROM Production.Product
```

```
    WHERE ProductID = @ProductID;

    IF @CurrentStock IS NULL
    BEGIN
        PRINT 'Invalid ProductID. Please try again.';
        RETURN;
    END

    IF @CurrentStock < @Quantity
    BEGIN
        PRINT 'Not enough stock available. Please try again.';
        RETURN;
    END

    SELECT @SpecialOfferID = SpecialOfferID
    FROM Sales.SpecialOfferProduct
    WHERE ProductID = @ProductID;

    INSERT INTO Sales.SalesOrderDetail (SalesOrderID, ProductID, UnitPrice, OrderQty,
UnitPriceDiscount, SpecialOfferID)
    VALUES (@OrderID, @ProductID, @ProductUnitPrice, @Quantity, @Discount,
@SpecialOfferID);

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Failed to place the order. Please try again.';
        RETURN;
    END

    UPDATE Production.ProductInventory
    SET Quantity = Quantity - @Quantity
    WHERE ProductID = @ProductID;

    IF (SELECT Quantity FROM Production.ProductInventory WHERE ProductID = @ProductID)
< @ReorderLevel
    BEGIN
        PRINT 'Warning: The quantity in stock of the product has dropped below its
reorder level.';
    END

    PRINT 'Order placed successfully.';
END;

2)

SELECT * from Production.Product ;
Select * from Person.Address;
Select * from Person.Person;
Select * from Sales.Customer where PersonID IS NOT NULL;
Select * from Person.BusinessEntityAddress;
SELECT * FROM Person.StateProvince;
SELECT * FROM Person.CountryRegion;
SELECT * FROM Sales.Customer;
SELECT * from HumanResources.EmployeePayHistory;
SELECT * from Sales.SalesOrderDetail;
select * from Person.StateProvince;
SELECT * FROM Person.Address;
SELECT * from Sales.SalesOrderHeader;
SELECT * FROM Person.BusinessEntityContact;
SELECT * FROM HumanResources.Employee;
SELECT * FROM Person.Person;
select* from production.Product;
select* from Production.Product;
```

```sql
select * from Production.ProductInventory;
SELECT * from Sales.SalesOrderDetail;
SELECT * FROM Sales.SpecialOfferProduct;
SELECT * from Sales.SalesOrderHeader;


EXEC UpdateOrderDetails
    @OrderID = 43659,
    @ProductID = 767,
    @Quantity = 15;

CREATE PROCEDURE UpdateOrderDetails
    @OrderID INT,
    @ProductID INT,
    @UnitPrice DECIMAL(18, 2) = NULL,
    @Quantity INT = NULL,
    @Discount DECIMAL(5, 2) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CurrentUnitPrice DECIMAL(18, 2);
    DECLARE @CurrentQuantity INT;
    DECLARE @CurrentDiscount DECIMAL(5, 2);
    DECLARE @OldQuantity INT;
    DECLARE @NewQuantity INT;

    SELECT @CurrentUnitPrice = UnitPrice,
           @CurrentQuantity = OrderQty,
           @CurrentDiscount = UnitPriceDiscount
    FROM Sales.SalesOrderDetail
    WHERE SalesOrderID = @OrderID AND ProductID = @ProductID;

    IF @CurrentQuantity IS NULL
    BEGIN
        PRINT 'Invalid OrderID or ProductID. Please try again.';
        RETURN;
    END

    SET @OldQuantity = @CurrentQuantity;
    SET @NewQuantity = ISNULL(@Quantity, @CurrentQuantity);

    UPDATE Sales.SalesOrderDetail
    SET UnitPrice = ISNULL(@UnitPrice, @CurrentUnitPrice),
        OrderQty = @NewQuantity,
        UnitPriceDiscount = ISNULL(@Discount, @CurrentDiscount)
    WHERE SalesOrderID = @OrderID AND ProductID = @ProductID;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Failed to update the order. Please try again.';
        RETURN;
    END

    UPDATE Production.ProductInventory
    SET Quantity = Quantity + @OldQuantity - @NewQuantity
    WHERE ProductID = @ProductID;

    PRINT 'Order updated successfully.';
END;
```

3)

```
SELECT * from Production.Product ;
Select * from Person.Address;
Select * from Person.Person;
Select * from Sales.Customer where PersonID IS NOT NULL;
Select * from Person.BusinessEntityAddress;
SELECT * FROM Person.StateProvince;
SELECT * FROM Person.CountryRegion;
SELECT * FROM Sales.Customer;
SELECT * from HumanResources.EmployeePayHistory;
SELECT * from Sales.SalesOrderDetail;
select * from Person.StateProvince;
SELECT * FROM Person.Address;
SELECT * from Sales.SalesOrderHeader;
SELECT * FROM Person.BusinessEntityContact;
SELECT * FROM HumanResources.Employee;
SELECT * FROM Person.Person;
select* from production.Product;
select* from Production.Product;
select * from Production.ProductInventory;
SELECT * from Sales.SalesOrderDetail;
SELECT * FROM Sales.SpecialOfferProduct;
SELECT * from Sales.SalesOrderHeader;

DROP Procedure GetOrderDetails;

EXEC GetOrderDetails @OrderID = 43659;

CREATE PROCEDURE GetOrderDetails
    @OrderID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT SalesOrderID FROM Sales.SalesOrderDetail WHERE SalesOrderID
= @OrderID)
    BEGIN
        PRINT 'The OrderID ' + CAST(@OrderID AS NVARCHAR(20)) + ' does not exist';
        RETURN 1;
    END

    SELECT *
    FROM Sales.SalesOrderDetail
    WHERE SalesOrderID = @OrderID;
END;


4)

SELECT * from Production.Product ;
Select * from Person.Address;
Select * from Person.Person;
Select * from Sales.Customer where PersonID IS NOT NULL;
Select * from Person.BusinessEntityAddress;
SELECT * FROM Person.StateProvince;
SELECT * FROM Person.CountryRegion;
SELECT * FROM Sales.Customer;
SELECT * from HumanResources.EmployeePayHistory;
SELECT * from Sales.SalesOrderDetail;
select * from Person.StateProvince;
SELECT * FROM Person.Address;
SELECT * from Sales.SalesOrderHeader;
SELECT * FROM Person.BusinessEntityContact;
```

```sql
SELECT * FROM HumanResources.Employee;
SELECT * FROM Person.Person;
select* from production.Product;
select* from Production.Product;
select * from Production.ProductInventory;
SELECT * from Sales.SalesOrderDetail;
SELECT * FROM Sales.SpecialOfferProduct;
SELECT * from Sales.SalesOrderHeader;

EXEC DeleteOrderDetails @OrderID = 42619, @ProductID = 767;


CREATE PROCEDURE DeleteOrderDetails
    @OrderID INT,
    @ProductID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT SalesOrderID FROM Sales.SalesOrderDetail WHERE SalesOrderID
= @OrderID)
    BEGIN
        PRINT 'Invalid OrderID. The OrderID ' + CAST(@OrderID AS NVARCHAR(20)) + '
does not exist.';
        RETURN -1;
    END

    IF NOT EXISTS (SELECT 1 FROM Sales.SalesOrderDetail WHERE SalesOrderID = @OrderID
AND ProductID = @ProductID)
    BEGIN
        PRINT 'Invalid ProductID. The ProductID ' + CAST(@ProductID AS NVARCHAR(20)) +
' does not exist for the OrderID ' + CAST(@OrderID AS NVARCHAR(20)) + '.';
        RETURN -1;
    END

    DELETE FROM Sales.SalesOrderDetail
    WHERE SalesOrderID = @OrderID AND ProductID = @ProductID;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Failed to delete the order detail. Please try again.';
        RETURN -1;
    END

    PRINT 'Order detail deleted successfully.';
END;



FUNCTIONS

1)

CREATE FUNCTION dbo.FormatDateToMMDDYYYY (@InputDate DATETIME)
RETURNS VARCHAR(10)
AS
BEGIN
    DECLARE @FormattedDate VARCHAR(10);

    SET @FormattedDate = RIGHT('0' + CAST(MONTH(@InputDate) AS VARCHAR(2)), 2) + '/' +
                         RIGHT('0' + CAST(DAY(@InputDate) AS VARCHAR(2)), 2) + '/' +
                         CAST(YEAR(@InputDate) AS VARCHAR(4));
```

```
    RETURN @FormattedDate;
END;

SELECT dbo.FormatDateToMMDDYYYY('2006-11-21 23:34:05.920') AS Date;
```

2)

```
CREATE FUNCTION dbo.FormatDateToYYYYMMDD (@InputDate DATETIME)
RETURNS VARCHAR(8)
AS
BEGIN
    DECLARE @FormattedDate VARCHAR(8);

    SET @FormattedDate = CAST(YEAR(@InputDate) AS VARCHAR(4)) +
                         RIGHT('0' + CAST(MONTH(@InputDate) AS VARCHAR(2)), 2) +
                         RIGHT('0' + CAST(DAY(@InputDate) AS VARCHAR(2)), 2);

    RETURN @FormattedDate;
END;

SELECT dbo.FormatDateToYYYYMMDD('2006-11-21 23:34:05.920') AS Date;
```

VIEWS

1)

```
SELECT * from Production.Product ;
Select * from Person.Address;
Select * from Person.Person;
Select * from Sales.Customer where PersonID IS NOT NULL;
Select * from Person.BusinessEntityAddress;
SELECT * FROM Person.StateProvince;
SELECT * FROM Person.CountryRegion;
SELECT * FROM Sales.Customer;
SELECT * from HumanResources.EmployeePayHistory;
SELECT * from Sales.SalesOrderDetail;
select * from Person.StateProvince;
SELECT * FROM Person.Address;
SELECT * from Sales.SalesOrderHeader;
SELECT * FROM Person.BusinessEntityContact;
SELECT * FROM HumanResources.Employee;
SELECT * FROM Person.Person;
select* from production.Product;
select* from Production.Product;
select * from Production.ProductInventory;
SELECT * from Sales.SalesOrderDetail;
SELECT * FROM Sales.SpecialOfferProduct;
SELECT * from Sales.SalesOrderHeader;
SELECT * from Sales.Customer;
SELECT * from Sales.Store;

SELECT * FROM vwCustomerOrders;

CREATE VIEW vwCustomerOrders AS
SELECT
    s.Name AS CompanyName,
    soh.SalesOrderID AS OrderID,
    soh.OrderDate,
    sod.ProductID,
    pr.Name AS ProductName,
    sod.OrderQty AS Quantity,
```

```
    sod.UnitPrice,
    sod.OrderQty * sod.UnitPrice AS TotalPrice
FROM
    Sales.Customer c
    INNER JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID
    INNER JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
    INNER JOIN Production.Product pr ON sod.ProductID = pr.ProductID
    LEFT JOIN Sales.Store s ON c.StoreID = s.BusinessEntityID;


2)

SELECT * FROM vwCustomerOrdersYesterday;

-- sp_helptext vwCustomerOrdersYesterday;

CREATE VIEW vwCustomerOrdersYesterday AS
SELECT
    s.Name AS CompanyName,
    soh.SalesOrderID AS OrderID,
    soh.OrderDate,
    sod.ProductID,
    pr.Name AS ProductName,
    sod.OrderQty AS Quantity,
    sod.UnitPrice,
    sod.OrderQty * sod.UnitPrice AS TotalPrice
FROM
    Sales.Customer c
    INNER JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID
    INNER JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID
    INNER JOIN Production.Product pr ON sod.ProductID = pr.ProductID
    LEFT JOIN Sales.Store s ON c.StoreID = s.BusinessEntityID
WHERE
    soh.OrderDate = CONVERT(date, GETDATE() - 1);


3)

SELECT * FROM MyProducts;
CREATE VIEW MyProducts AS
SELECT
    p.ProductID,
    p.Name AS ProductName,
    p.Size AS QuantityPerUnit,
    p.ListPrice AS UnitPrice,
    v.Name AS CompanyName,
    pc.Name AS CategoryName
FROM
    Production.Product p
    INNER JOIN Purchasing.ProductVendor pv ON p.ProductID = pv.ProductID
    INNER JOIN Purchasing.Vendor v ON pv.BusinessEntityID = v.BusinessEntityID
    INNER JOIN Production.ProductSubcategory ps ON p.ProductSubcategoryID =
ps.ProductSubcategoryID
    INNER JOIN Production.ProductCategory pc ON ps.ProductCategoryID =
pc.ProductCategoryID
WHERE
    p.DiscontinuedDate IS NULL;

TRIGGERS

DROP TABLE Orders;
DROP TABLE Products;
DROP TABLE OrderDetails;
```

```sql
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID NVARCHAR(10),
    OrderDate DATE
);

CREATE TABLE Products (
    ProductID INT PRIMARY KEY,
    ProductName NVARCHAR(50),
    UnitsInStock INT
);

CREATE TABLE OrderDetails1 (
    OrderID INT,
    ProductID INT,
    Quantity INT,
);

INSERT INTO Orders (OrderID, CustomerID, OrderDate)
VALUES
    (1, 'ALFKI', '2024-05-30'),
    (2, 'BERGS', '2024-05-31');


INSERT INTO OrderDetails1 (OrderID, ProductID, Quantity)
VALUES
    (1, 1, 5),
    (1, 2, 3),
    (2, 3, 2);

INSERT INTO Products (ProductID, ProductName, UnitsInStock)
VALUES
    (1, 'Chai', 10),
    (2, 'Chang', 8),
    (3, 'Aniseed Syrup', 5);



CREATE TRIGGER InsteadOfDeleteOrders
ON Orders
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM OrderDetails1
    WHERE OrderID IN (SELECT OrderID FROM deleted);

    DELETE FROM Orders
    WHERE OrderID IN (SELECT OrderID FROM deleted);

       SELECT * FROM deleted;
END;

DELETE FROM Orders WHERE OrderID = 1;
select * from OrderDetails1;
select * from Orders;

CREATE TRIGGER CheckStockAvailability
ON OrderDetails1
AFTER INSERT
```

```
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ProductID INT;
    DECLARE @Quantity INT;

    SELECT @ProductID = ProductID, @Quantity = Quantity
    FROM inserted;

    DECLARE @StockAvailable INT;
    SELECT @StockAvailable = UnitsInStock
    FROM Products
    WHERE ProductID = @ProductID;

    IF @StockAvailable >= @Quantity
    BEGIN

        UPDATE Products
        SET UnitsInStock = UnitsInStock - @Quantity
        WHERE ProductID = @ProductID;
    END
    ELSE
    BEGIN
        RAISERROR ('Insufficient stock to fill the order. Please try again.', 16, 1);
        ROLLBACK TRANSACTION; -- Rollback the transaction to prevent insertion
    END;
END;
INSERT INTO OrderDetails1 (OrderID, ProductID, Quantity) VALUES (1, 1, 5);
INSERT INTO OrderDetails1 (OrderID, ProductID, Quantity) VALUES (3, 3, 5);
```

#HACKERRANK ID: https://www.hackerrank.com/profile/akanand20072002

1)Revisiting the select query 1

```
select * FROM CITY WHERE POPULATION > 100000 AND COUNTRYCODE = 'USA'
```

2)Revising Aggregations-sum

```
select sum(population) from CITY where District='California';
```

3)Revising Aggregation-count

```
select count(population) from CITY where Population>100000;
```

4)Averages

```
select AVG(Population) FROM CITY where district = 'California';
```

5)Population Density Difference

```
Select MAX(Population)-MIN(Population) FROM City;
```

6)Placements

```
SELECT Name FROM

(SELECT S1.ID, F.Friend_ID, S1.Name, S2.Name AS Friend, P1.Salary, P2.Salary AS
Friend_salary
```

```
 FROM
 Students S1 INNER JOIN Friends F
 ON
 S1.ID=F.ID
INNER JOIN Students S2
 ON
 F.Friend_ID=S2.ID
INNER JOIN Packages P1
 ON
 S1.ID=P1.ID
INNER JOIN Packages P2
 ON
 S2.ID=P2.ID
ORDER BY S1.ID) AS A
WHERE Salary<Friend_salary
ORDER BY Friend_salary;
```

7)Average Population Of Each Continent

```
SELECT COUNTRY.CONTINENT, FLOOR(AVG(CITY.POPULATION)) FROM CITY JOIN COUNTRY ON
CITY.CountryCode = COUNTRY.Code GROUP BY COUNTRY.CONTINENT
```

8)Symmetric Pairs

```
SELECT X1, Y1 FROM
(SELECT F1.X AS X1, F1.Y AS Y1, F2.X AS X2, F2.Y AS Y2 FROM Functions F1
INNER JOIN Functions F2 ON F1.X=F2.Y AND F1.Y=F2.X
ORDER BY F1.X) AS A
GROUP BY X1, Y1
HAVING COUNT(X1)>1 OR X1<Y1
ORDER BY X1;
```

9)Population Census

```
SELECT SUM(City.Population) FROM City INNER JOIN Country ON
City.CountryCode=Country.Code
WHERE Continent='Asia';
```

10)The Report

```
SELECT CASE WHEN Grade>=8 THEN Name ELSE NULL END AS Name, Grade, Marks FROM Students
JOIN Grades
ON Students.Marks<=Max_Mark AND Min_Mark<=Students.Marks
ORDER BY Grade DESC,Name,Marks;
```