Database Foundations

**PRACTICE SESSION 3**

Please use the following logical data model to write SQL statements for the questions that follow. This relational schema was created for an extended version of the Pine Valley Furniture Company (PVFC) ER diagram. Before you start with the questions, please make sure that you understand the structure of the database. Please use the Oracle SQL syntax.

**A) Relations**

CUSTOMER (CustomerID, CustomerName, CustomerAddress, CustomerCity,   
 CustomerState, CustomerPostalCode)

DOESBUSINESS (CustomerID, TerritoryID)

ORDER (OrderID, OrderDate, CustomerID, FulfillmentDate, SalespersonID)

PAYMENT (PaymentID, OrderID, PaymentDate, PaymentAmount)

ORDERLINE (OrderID, ProductID, OrderedQuantity)

PRODUCT (ProductID, ProductDescription, ProductFinish, ProductStandardPrice,   
 ProductOnHand, ProductLineID)

PRODUCTLINE (ProductLineID, ProductLineName)

USES (ProductID, MaterialID, QuantityRequired)

RAWMATERIAL (MaterialID, MaterialName, Thickness, Width, Size,   
 MaterialStandardPrice, UnitofMeausure, MaterialType)

WORKCENTER (WorkCenterID, WorkCenterLocation)

PRODUCEDIN (ProductID, WorkCenterID)

EMPLOYEE (EmployeeID, EmployeeName, EmployeeAddress, EmployeeCity,   
 EmployeeState, EmployeeZip, EmployeeDateHired, EmployeeBirthdate,   
 EmployeeSupervisor)

WORKSIN (EmployeeID, WorkCenterID)

EMPLOYEESKILLS (EmployeeID, SkillID, QualifyDate)

SKILL (SkillID, SkillDescription)

SALESPERSON (SalespersonID, SalespersonName, SalespersonAdress,   
 SalespersonCity, SalespersonState, SalespersonZip, TerritoryID)

TERRITORY (TerritoryID, TerritoryName)

VENDOR (VendorID, VendorName, VendorAddress, VendorCity, VendorState,   
 VendorZip, VendorPhone, VendorContact)

SUPPLIES (MaterialID, VendorID)

**B) Referential Integrity Constraints**

CustomerID in DOESBUSINESS references CustomerID in CUSTOMER

CustomerID in ORDER references CustomerID in CUSTOMER

SalespersonID in ORDER references SalespersonID in SALESPERSON

OrderID in PAYMENT references OrderID in ORDER

OrderID in ORDERLINE references OrderID in ORDER

ProductID in ORDERLINE references ProductID in PRODUCT

ProductID in USES references ProductID in PRODUCT

MaterialID in USES references MaterialID in RAWMATERIAL

ProductID in PRODUCEDIN references ProductID in PRODUCT

WorkCenterID in PRODUCEDIN references WorkCenterID in WORKCENTER

EmployeeSupervisor in EMPLOYEE references EmployeeID in EMPLOYEE

EmployeeID in EMPLOYEESKILLS references EmployeeID in EMPLOYEE

SkillID in EMPLOYEESKILLS references SkillID in SKILL

TerritoryID in SALESPERSON references TerritoryID in TERRITORY

ProductLineID in PRODUCT references ProductLineID in PRODUCTLINE

MaterialID in SUPPLIES references MaterialID in RAWMATERIAL

VendorID in SUPPLIES references VendorID in VENDOR

EmployeeID in WORKSIN references EmployeeID in EMPLOYEE

WorkCenterID in WORKSIN references WorkCenterID in WORKCENTER

**C) Data Types**

|  |  |  |
| --- | --- | --- |
| (i) Number | (ii) Text | (iii) Date |
| CustomerID  TerritoryID  OrderID  PaymentID  PaymentAmount  ProductID  SalespersonID  OrderedQuantity  ProductStandardPrice  ProductOnHand  ProductLineID  MaterialID  QuantityRequired  MaterialStandardPrice  WorkCenterID  EmployeeID  VendorID  EmployeeSupervisor | CustomerName  CustomerAddress  CustomerCity  CustomerState  CustomerPostalCode  ProductDescription  ProductFinish  MaterialName  Thickness  Width  Size  UnitofMeausure  MaterialType  WorkCenterLocation  EmployeeName  EmployeeAddress  EmployeeCity  EmployeeState  EmployeeZip  SkillID  SkillDescription  SalespersonName  SalespersonAdress  SalespersonCity  SalespersonState  SalespersonZip  VendorName  VendorAddress  VendorCity  VendorState  VendorZip  VendorPhone  VendorContact  TerritoryName | OrderDate  FulfillmentDate  PaymentDate  EmployeeDateHired  EmployeeBirthdate  QualifyDate |

**Note:** The solutions, which include SQL statements, are not intended as the definitive answer to the questions, but as possible solutions. You can approach the problems using different SQL capabilities, achieving results that are also correct. I illustrate the SQL statements with capitalized SQL Reserve Words, and Upper/Lower case usage for data object names, to be consistent with the naming conventions in lectures. Oracle results will display table and column names with all upper case letters.

1. Write an SQL command that will find any customers who have not placed orders.

SELECT CustomerID, CustomerName

FROM Customer\_T

WHERE CustomerID

NOT IN (SELECT CustomerID FROM Order\_T);

ALTERNATIVELY

SELECT Customer\_T.CustomerID, CustomerName

FROM Customer\_T LEFT OUTER JOIN Order\_T

ON Customer\_T.CustomerID= Order\_T.CustomerID

WHERE Order\_T.CustomerID IS NULL;

ALTERNATIVELY

SELECT CustomerID

FROM Customer\_T

MINUS

SELECT CustomerID

FROM Order\_T;

**Note:** The last solution can only display CustomerIDs.

1. List the number of employees supervised (label this value *HeadCount*) for each supervisor who supervises more than two employees.

SELECT S.EmployeeID, S.EmployeeName, COUNT(E.EmployeeID) AS HeadCount

FROM Employee\_T E, Employee\_T S

WHERE E.EmployeeSupervisor = S.EmployeeID

GROUP BY S.EmployeeID, S.EmployeeName

HAVING COUNT(E.EmployeeID)>2;

**Note:** If you just want the employee numbers of these supervisors, you do not need to do a self-join.

SELECT EmployeeSupervisor, COUNT(EmployeeID) AS HeadCount

FROM Employee\_T

GROUP BY EmployeeSupervisor

HAVING COUNT(EmployeeID)>2;

1. List the name of each employee, his or her birthdate, the name of his or her manager, and the manager’s birthdate for those employees who were born before their manager was born; label manager’s data *Manager* and *ManagerBirth*.

SELECT E1.EmployeeName, E1.EmployeeBirthdate,

E2.EmployeeName AS Manager,

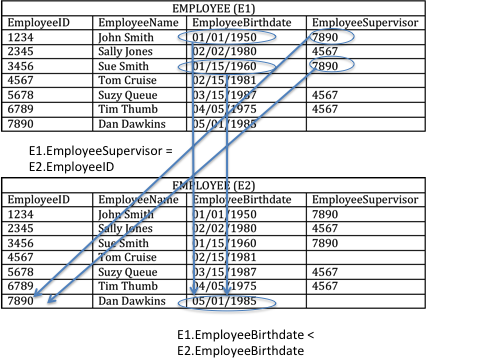
E2.EmployeeBirthdate AS ManagerBirth

FROM Employee\_T E1, Employee\_T E2

WHERE E1.EmployeeSupervisor = E2.EmployeeID

AND E1.EmployeeBirthdate < E2.EmployeeBirthdate;

An instance diagram (with sample data for illustration purposes only)



1. Write an SQL command to display the order number(s), customer number, order date(s), and product description(s) of the items ordered by customer number 4.

SELECT Order\_T.OrderID, CustomerID, OrderDate,

ProductDescription

FROM OrderLine\_T, Order\_T, Product\_T

WHERE Order\_T.OrderID=OrderLine\_T.OrderID AND

OrderLine\_T.ProductID=Product\_T.ProductID AND

CustomerID = 4;

1. Write an SQL command to display each item ordered in order number 1, its standard price, and the total price for each item ordered.

SELECT OrderLine\_T.ProductID, ProductDescription,

ProductStandardPrice,

(OrderedQuantity\*ProductStandardPrice) AS TotalPrice

FROM Product\_T,OrderLine\_T

WHERE Product\_T.ProductID = OrderLine\_T.ProductID AND

OrderID=1;

1. For each product, calculate the total raw material cost (label this column TotCost) and compare this cost with the standard price of the product. Display product ID, product description, standard price, and total raw material cost in the result table.

SELECT P.ProductID, ProductDescription,

ProductStandardPrice,

SUM(QuantityRequired\*MaterialStandardPrice) AS TotCost

FROM Product\_T P, Uses\_T U, RawMaterial\_T R

WHERE P.ProductID = U.ProductID

AND U.MaterialID = R.MaterialID

GROUP BY P.ProductID, ProductDescription,

ProductStandardPrice;

Or here is another interesting approach using a derived table, which is named Cost\_T:

SELECT Product\_T.ProductID, ProductDescription,

ProductStandardPrice, TotCost

FROM Product\_T, (SELECT ProductID,

SUM(MaterialStandardPrice\*QuantityRequired) AS TotCost

FROM Uses\_T, RawMaterial\_T

WHERE Uses\_T.MaterialID = RawMaterial\_T.MaterialID

GROUP BY ProductID) Cost\_T

WHERE Product\_T.ProductID = Cost\_T.ProductID;

1. For every order that has been received, display the order ID, the total dollar value of the order (label this column *Total Due*), and the amount received in payments on that order (note that there can be multiple payments made on each order). To make this query a little simpler, you don’t have to include those orders for which no payment has yet been received. List the results in decreasing order of the remaining balance (i.e., the difference between total due and amount paid.)

SELECT OrderLine\_T.OrderID,

SUM(OrderedQuantity\*ProductStandardPrice) AS

TotalDue, AmountPaid

FROM OrderLine\_T, Product\_T, (SELECT OrderID,   
 SUM(PaymentAmount) AS AmountPaid FROM Payment\_T   
 GROUP BY OrderID) TotalPayment\_T

WHERE OrderLine\_T.ProductID = Product\_T.ProductID

AND OrderLine\_T.OrderID = TotalPayment\_T.OrderID

GROUP BY OrderLine\_T.OrderID, AmountPaid

ORDER BY TotalDue - AmountPaid DESC;

1. Write an SQL query to list each customer who has bought computer desks and the number of units sold to each customer.

SELECT Customer\_T.CustomerID, CustomerName,

SUM(OrderedQuantity) AS UnitsBought

FROM OrderLine\_T, Order\_T, Product\_T, Customer\_T

WHERE ProductDescription LIKE '%Computer Desk%'

AND Order\_T.OrderID = OrderLine\_T.OrderID

AND Product\_T.ProductID = OrderLine\_T.ProductID

AND Customer\_T.CustomerID = Order\_T.CustomerID

GROUP BY Customer\_T.CustomerID, CustomerName;

1. List, in alphabetical order, the names of all managers who are managing an employee with skill ID BS12. List each such manager’s name only once, even if that manager manages several people with this skill.

SELECT DISTINCT M.EmployeeName

FROM Employee\_T E, Employee\_T M,

EmployeeSkills\_T ES

WHERE SkillID = 'BS12'

AND ES.EmployeeID = E.EmployeeID

AND E.EmployeeSupervisor = M.EmployeeID

ORDER BY 1;

1. Display the salesperson name, product finish, and total quantity sold (label as *TotSales*) for each finish by each salesperson.

SELECT SalespersonName, ProductFinish, SUM(OrderedQuantity) AS TotSales

FROM Salesperson\_T, OrderLine\_T, Product\_T, Order\_T

WHERE Salesperson\_T.SalespersonID = Order\_T.SalespersonID

AND Order\_T.OrderID = OrderLine\_T.OrderID

AND OrderLine\_T.ProductID = Product\_T.ProductID

GROUP BY SalespersonName, ProductFinish;

1. Write a query to list the number of products produced in each work center (label as TotalProducts). If a work center does not produce any products, display the result with a total of 0.

SELECT WorkCenter\_T.WorkCenterID, COUNT(ProductID) AS TotalProducts

FROM WorkCenter\_T LEFT OUTER JOIN ProducedIn\_T

ON WorkCenter\_T.WorkCenterID = ProducedIn\_T.WorkCenterID

GROUP BY WorkCenter\_T.WorkCenterID;

|  |  |
| --- | --- |
| **WORKCENTERID** | **TOTALPRODUCTS** |
| WR1 | 0 |
| Tampa1 | 14 |
| SM1 | 0 |

**Note:** If you use COUNT(\*) as below, this will count the rows even if there is no ProductID associated with a group defined by WorkCenterID. This query will run but it won’t give the correct answer.

SELECT WorkCenter\_T.WorkCenterID, COUNT(\*) AS TotalProducts

FROM WorkCenter\_T LEFT OUTER JOIN ProducedIn\_T

ON WorkCenter\_T.WorkCenterID = ProducedIn\_T.WorkCenterID

GROUP BY WorkCenter\_T.WorkCenterID;

|  |  |
| --- | --- |
| **WORKCENTERID** | **TOTALPRODUCTS** |
| WR1 | 1 |
| Tampa1 | 14 |
| SM1 | 1 |

1. The production manager at PVFC is concerned about support for parts in products owned by customers. He wants to do a simple analysis to determine for each customer how many vendors are in the same state as that customer. Develop a list of all the PVFC customers (by ID and name) with the number of vendors in the same state as that customer. (Label this computed result *NumVendors*)

SELECT CustomerID, CustomerName, COUNT(VendorID) AS NumVendors

FROM Customer\_T LEFT OUTER JOIN Vendor\_T

ON CustomerState = VendorState

GROUP BY CustomerID, CustomerName;

1. Display the order IDs for customers who have not made any payment, yet, on that order. Use the set command UNION, INTERSECT, or MINUS in your query.

SELECT OrderID

FROM Order\_T

MINUS

SELECT OrderID

FROM Payment\_T;

1. Display the names of the states in which customers reside but for which there is no salesperson residing in that state.

*First, a solution using the MINUS operator:*

SELECT CustomerState

FROM Customer\_T

MINUS

SELECT SalespersonState

FROM Salesperson\_T;

**Note:** Table B MINUS Table A returns the distinct set of rows in Table B, therefore duplicates are removed (hence, no need to use DISTINCT keyword)

*Next, a solution using the OUTER JOIN:*

SELECT DISTINCT CustomerState

FROM Customer\_T LEFT OUTER JOIN Salesperson\_T

ON CustomerState = SalespersonState

WHERE SalespersonState IS NULL

ORDER BY CustomerState;

1. Write an SQL query to produce a list of all the products (i.e., product ID and product description) and the number of times each product has been ordered.

SELECT Product\_T.ProductID, ProductDescription,

COUNT(OrderID) as TimesOrdered

FROM Product\_T LEFT OUTER JOIN OrderLine\_T

ON Product\_T.ProductID = OrderLine\_T.ProductID

GROUP BY Product\_T.ProductID, ProductDescription

ORDER BY Product\_T.ProductID;

**Note:** This query requires an outer join because some products may not have been ordered. Because many SQL systems do not have an outer join operator, this type of query can also use the UNION command. The following answer uses this second approach because it will work with almost any system. Also, note that the question wants the number of times a product has been ordered, not the total quantity ordered.

SELECT Product\_T.ProductID, ProductDescription,

COUNT(\*) AS TimesOrdered

FROM Product\_T,OrderLine\_T

WHERE Product\_T.ProductID = OrderLine\_T.ProductID

GROUP BY Product\_T.ProductID, ProductDescription

UNION

SELECT ProductID, ProductDescription, 0

FROM Product\_T

WHERE NOT EXISTS

(SELECT \* FROM OrderLine\_T

WHERE OrderLine\_T.ProductID =

Product\_T.ProductID);

1. Display the customer ID, name, and order ID for all customer orders. For those customers who do not have any orders, include them in the display once. Sort the results by customer ID.

SELECT CUST.CustomerID, CustomerName, OrderID

FROM Customer\_T CUST

LEFT OUTER JOIN Order\_T ORD

ON CUST.CustomerID = ORD.CustomerID

ORDER BY CUST.CustomerID;

1. Display the employee ID and employee name for those employees who do not possess the skill “Router”. Display the results in order by employee name.

SELECT EmployeeID, EmployeeName FROM Employee\_T

WHERE EmployeeID NOT IN

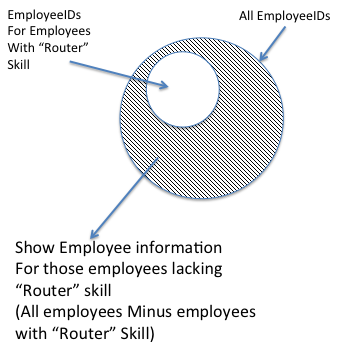
(SELECT ES.EmployeeID FROM EmployeeSkills\_T ES,

Skill\_T S

WHERE SkillDescription = 'Router'

AND ES.SkillID = S.SkillID)

ORDER BY EmployeeName;



1. Display the name of customer 16 and the names of all other customers that are in the same zip code as customer 16.

SELECT C1.CustomerName, C2.CustomerName, C1.CustomerPostalCode

FROM Customer\_T C1, Customer\_T C2

WHERE C1.CustomerID = 16

AND C1.CustomerPostalCode = C2.CustomerPostalCode

AND C2.CustomerID != 16;

1. Rewrite your answer to the previous question for each customer, not just customer 16.

SELECT C1.CustomerName, C2.CustomerName AS CName2,

C1.CustomerPostalCode

FROM Customer\_T C1, Customer\_T C2

WHERE C1.CustomerPostalCode = C2.CustomerPostalCode

AND C2.CustomerID != C1.CustomerID;

1. Display the customer ID, name, and order ID for all customer orders. For those customers who do not have any orders, include them in the display once, with a 0 value for OrderID.

SELECT Customer\_T.CustomerID, CustomerName, OrderID

FROM Customer\_T, Order\_T

WHERE Customer\_T.CustomerID = Order\_T.CustomerID

UNION

SELECT CustomerID, CustomerName, 0

FROM Customer\_T

WHERE NOT EXISTS

(SELECT \* FROM Order\_T

WHERE Order\_T.CustomerID = Customer\_T.CustomerID);

Or replace last three lines above with:

WHERE CustomerID NOT IN (SELECT CustomerID FROM Order\_T);

ALTERNATIVELY

SELECT Customer\_T.CustomerID, CustomerName, OrderID

FROM Customer\_T, Order\_T

WHERE Customer\_T.CustomerID = Order\_T.CustomerID

UNION

SELECT Customer\_T.CustomerID, CustomerName, 0

FROM Customer\_T LEFT OUTER JOIN Order\_T

ON Order\_T.CustomerID = Customer\_T.CustomerID

WHERE Order\_T.OrderID IS NULL;

**Note:** Query 20 is similar to Query 16. The only difference is that Query 16 displays a null value for order ID for customers who have not placed any order.

1. Show the customer ID and name for all the customers who have ordered both products with IDs 3 and 4 on the same order. Display the same customer information only once.

SELECT DISTINCT C.CustomerID, CustomerName

FROM Customer\_T C, Order\_T O1, OrderLine\_T OL1

WHERE C.CustomerID = O1.CustomerID

AND O1.OrderID = OL1.OrderID

AND OL1.ProductID = 3

AND O1.OrderID IN

(SELECT OrderID FROM OrderLine\_T OL2

WHERE OL2.ProductID = 4);

Or here is another interesting approach using derived tables

SELECT DISTINCT CustomerID, CustomerName

FROM Customer\_T

WHERE CustomerID IN

(SELECT CustomerID

FROM

(SELECT p3.OrderID

FROM (SELECT OrderID FROM OrderLine\_T WHERE

ProductID = 3) p3,

(SELECT OrderID from OrderLine\_T WHERE

ProductID = 4) p4

WHERE p3.OrderID = p4.OrderID) p3p4, Order\_T

WHERE p3p4.OrderID = Order\_T.OrderID);

**Note:** Derived tables named p3 (which finds the orders that include product 3) and p4 (which finds the orders that include product 4) are joined together, and the resulting table is named p3p4 (which lists the orders that include both product 3 and product 4). Then, p3p4 table is joined with Order\_T to get the customer IDs associated with each order in p3p4. Finally, customer names are retrieved from the Customer\_T for each customer who has placed an order that included product 3 and product 4.

1. Display the customer names of all customers who have ordered (on the same or different orders) both products with IDs 3 and 4.

SELECT C.CustomerID, CustomerName

FROM Customer\_T C, Order\_T O1, OrderLine\_T OL1

WHERE C.CustomerID = O1.CustomerID

AND O1.OrderID = OL1.OrderID

AND OL1.ProductID = 3

INTERSECT

SELECT C.CustomerID, CustomerName

FROM Customer\_T C, Order\_T O1, OrderLine\_T OL1

WHERE C.CustomerID = O1.CustomerID

AND O1.OrderID = OL1.OrderID

AND OL1.ProductID = 4;

1. Write an SQL query to list the order number and order quantity for all customer orders for which the order quantity is greater than the average order quantity of that product: (Hint: This involves a correlated subquery).

SELECT x1.OrderID, x1.OrderedQuantity, x1.ProductID

FROM Product\_T, OrderLine\_T x1

WHERE Product\_T.ProductID = x1.ProductID

AND x1.OrderedQuantity >

(SELECT AVG(OrderedQuantity) FROM OrderLine\_T x2

WHERE x2.ProductID = x1.ProductID);

1. Display the product ID and total amount ordered of that product by the customer who has bought the most of that product. Display the results in product ID order.

SELECT F1.ProductID, MAX(F1.ProdCustTotal) as MaxOrdered

FROM

(SELECT ProductID, CustomerID, SUM(OrderedQuantity)

AS ProdCustTotal

FROM OrderLine\_T OL1, Order\_T O1

WHERE OL1.OrderID = O1.OrderID

GROUP BY ProductID, CustomerID) F1

GROUP BY F1.ProductID

ORDER BY F1.ProductID;

**Result*:***

|  |  |
| --- | --- |
| **PRODUCTID** | **TOTORDERED** |
| 1 | 9 |
| 2 | 26 |
| 3 | 12 |
| 4 | 4 |
| 5 | 10 |
| 6 | 4 |
| 7 | 4 |
| 8 | 2 |
| 10 | 9 |
| 13 | 2 |
| 14 | 10 |
| 17 | 5 |
| 20 | 1 |

1. Display the employee information for all the employees who were hired before the most recently hired person in that state.

SELECT \*

FROM Employee\_T E1

WHERE EmployeeDateHired <

(SELECT MAX(EmployeeDateHired)

FROM Employee\_T E2

WHERE E1.EmployeeState = E2.EmployeeState);

ALTERNATIVELY

SELECT \*

FROM Employee\_T E1, (SELECT EmployeeState, MAX(EmployeeDateHired) MaxDate FROM Employee\_T GROUP BY EmployeeState) E2

WHERE E1.EmployeeState = E2.EmployeeState

AND E1.EmployeeDateHired<E2.MaxDate;