Solution 1

Lokendra Jain

Base Query 1 (BQ1)

BQ1 involves the job fact, location dimension, and time dimension tables. In the following SELECT statement, the cross product join style is used with 3 tables in the FROM clause and 2 join conditions in the WHERE clause. Alternatively, the join operator style can be used with 2 join operations in the FROM clause. The GROUP BY clause must contain all non-aggregate columns (Location_Id, Location_Name, Sales_Class_Id, Sales_Class_Desc, Base_Price, Time_Year, and Time_Month).

```
-- Base query BG1 in the revenue/costs area
-- Location and sales class summary of job quantity and amount

SELECT W_Location_D.Location_Id, Location_Name,
       W_Sales_Class_D.Sales_Class_Id, Sales_Class_Desc,
       Base_Price, Time_Year, Time_Month,
       SUM ( QUANTITY_ORDERED ) AS Sum_Job_Qty,
       SUM ( QUANTITY_ORDERED * Unit_Price ) AS Sum_Job_Amount
 FROM W_JOB_F, W_Location_D, W_TIME_D, W_Sales_Class_D
 WHERE W_Location_D.Location_ID = W_Job_F.Location_Id
   AND W_JOB_F.CONTRACT_DATE = W_TIME_D.Time_ID
   AND W_Job_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id
 GROUP BY W_Location_D.Location_Id, Location_Name,
          W_Sales_Class_D.Sales_Class_Id, Sales_Class_Desc,
          Base_Price, Time_Year, Time_Month;
```

Base Query 2 (BQ2)

BQ2 involves 4 fact tables (job, subjob, shipment, and invoice line) and 2 dimension tables, location and time. In the following SELECT statement, the cross product join style is used with 6 tables in the FROM clause and 5 join conditions in the WHERE clause. Alternatively, the join operator style can be used with 5 join operations in the FROM clause. The GROUP BY clause must contain all non-aggregate columns (Job_Id, Location_Id, Location_Name, Quantity_Ordered, Unit-Price, Time_Year, and Time_Month). To facilitate formulation of analytic queries, the base query should be placed in a CREATE VIEW statement.

```
-- BQ2 in the revenue/costs area
-- Location invoice revenue summary
-- Use contract year and month

SELECT W_Sub_Job_F.Job_Id,
       W_Location_D.LOCATION_ID, W_LOCATION_D.LOCATION_NAME,
       Quantity_Ordered, Unit_Price,
       W_TIME_D.TIME_YEAR, W_TIME_D.TIME_MONTH,
       SUM (Invoice_Quantity) AS SumInvoiceQty,
       SUM (Invoice_Amount) AS SumInvoiceAmt
 FROM W_Job_Shipment_F, W_Sub_Job_F, W_Location_D, W_Time_D,
      W_InvoiceLine_F, W_Job_F
 WHERE W_Sub_Job_F.Sub_Job_Id = W_Job_Shipment_F.Sub_Job_Id
   AND W_Job_Shipment_F.Invoice_Id = W_InvoiceLine_F.Invoice_Id
   AND W_Time_D.Time_Id = Contract_Date
   AND W_Location_D.Location_Id = W_InvoiceLine_F.Location_Id
   AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
 GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.Location_Id,
          W_LOCATION_D.LOCATION_NAME, Quantity_Ordered, Unit_Price,
          W_Time_D.Time_Year, W_Time_D.Time_Month;

-- CREATE VIEW statement
CREATE VIEW LocRevenueSummary AS
 SELECT W_Sub_Job_F.Job_Id,
        W_Location_D.LOCATION_ID, W_LOCATION_D.LOCATION_NAME,
        Quantity_Ordered, Unit_Price,
        W_TIME_D.TIME_YEAR, W_TIME_D.TIME_MONTH,
        SUM (Invoice_Quantity) AS SumInvoiceQty,
        SUM (Invoice_Amount) AS SumInvoiceAmt
  FROM W_Job_Shipment_F, W_Sub_Job_F, W_Location_D, W_Time_D,
       W_InvoiceLine_F, W_Job_F
  WHERE W_Sub_Job_F.Sub_Job_Id = W_Job_Shipment_F.Sub_Job_Id
    AND W_Job_Shipment_F.Invoice_Id = W_InvoiceLine_F.Invoice_Id
    AND W_Time_D.Time_Id = Contract_Date
    AND W_Location_D.Location_Id = W_InvoiceLine_F.Location_Id
    AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
  GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.Location_Id,
           W_LOCATION_D.LOCATION_NAME, Quantity_Ordered, Unit_Price,
           W_Time_D.Time_Year, W_Time_D.Time_Month;
```

Base Query 3 (BQ3)

BQ3 involves 2 fact tables (job and subjob) and 3 dimension tables (location, time, and machine type). In the following SELECT statement, the cross product join style is used with 5 tables in the FROM clause and 4 join conditions in the WHERE clause. Alternatively, the join operator style can be used with 4 join operations in the FROM clause. The GROUP BY clause must contain all non-aggregate columns (Job_Id, Location_Id, Location_Name, Time_Year, and Time_Month). To facilitate formulation of analytic queries, the base query should be placed in a CREATE VIEW statement.

```
-- BQ3 in the revenue/costs area
-- Location subjob cost summary
-- Use contract year and month to match revenues/costs

SELECT W_Sub_Job_F.Job_Id,
       W_Location_D.LOCATION_ID ,W_LOCATION_D.LOCATION_NAME,
       W_TIME_D.TIME_YEAR,  W_TIME_D.TIME_MONTH,
       SUM(Cost_Labor) AS SumLaborCosts,
       SUM(Cost_Material) AS SumMaterialCosts,
       SUM(Cost_Overhead) AS SumOvrhdCosts,
       SUM(Machine_Hours * Rate_Per_Hour) AS SumMachineCosts,
       SUM(Quantity_Produced) AS SumQtyProduced,
       SUM(Cost_Labor + Cost_Material + Cost_Overhead +
            (Machine_Hours * Rate_Per_Hour) ) AS TotalCosts,
       SUM( Cost_Labor + Cost_Material + Cost_Overhead + (Machine_Hours *
            Rate_Per_Hour) ) / SUM(Quantity_Produced)  AS UnitCosts
 FROM W_Job_F, W_Sub_Job_F, W_Location_D, W_Time_D, W_Machine_Type_D
 WHERE W_Job_F.Location_Id = W_Location_D.Location_Id
   AND W_Sub_Job_F.Machine_Type_Id = W_Machine_Type_D.Machine_Type_Id
   AND W_Time_D.Time_Id = Contract_Date
   AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
 GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
          W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR,
          W_TIME_D.TIME_MONTH;

CREATE VIEW LocCostSummary AS
SELECT W_Sub_Job_F.Job_Id,
       W_Location_D.LOCATION_ID ,W_LOCATION_D.LOCATION_NAME,
       W_TIME_D.TIME_YEAR,  W_TIME_D.TIME_MONTH,
       SUM(Cost_Labor) AS SumLaborCosts,
       SUM(Cost_Material) AS SumMaterialCosts,
       SUM(Cost_Overhead) AS SumOvrhdCosts,
       SUM(Machine_Hours * Rate_Per_Hour) AS SumMachineCosts,
       SUM(Quantity_Produced) AS SumQtyProduced,
       SUM(Cost_Labor + Cost_Material + Cost_Overhead +
            (Machine_Hours * Rate_Per_Hour) ) AS TotalCosts,
       SUM( Cost_Labor + Cost_Material + Cost_Overhead + (Machine_Hours *
            Rate_Per_Hour) ) / SUM(Quantity_Produced)  AS UnitCosts
 FROM W_Job_F, W_Sub_Job_F, W_Location_D, W_Time_D, W_Machine_Type_D
 WHERE W_Job_F.Location_Id = W_Location_D.Location_Id
   AND W_Sub_Job_F.Machine_Type_Id = W_Machine_Type_D.Machine_Type_Id
   AND W_Time_D.Time_Id = Contract_Date
   AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
 GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
          W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR,
          W_TIME_D.TIME_MONTH;
```

Base Query 4 (BQ4)

BQ4 involves 1 fact table (invoice line) and 3 dimension tables (location, sales class, and time). In the following SELECT statement, the join operator style is used with 3 join operations in the FROM clause. Alternatively, the cross product style could be used with 4 tables in the

FROM clause and 3 join conditions in the WHERE clause. The GROUP BY clause must contain all non-aggregate columns (Location_Id, Location_Name, Sales_Class_Id, Sales_Class_Desc, Time_Year, and Time_Month). The WHERE clause must contain the condition that the quantity shipped is larger than the quantity invoiced. Note the calculation of return amount in the computed column *SumReturnAmt* involves a calculation of unit price (invoice_amount / invoice_quantity).

```
-- BQ4 in the quality control area
-- Return quantity and amount by location and sales class
-- Calculate unit price as invoice_amount/invoice_quantity

SELECT
  W_Location_D.Location_Id, Location_Name,
  W_Sales_Class_D.Sales_Class_Id, Sales_Class_Desc,
  Time_Year, Time_Month,
  SUM ( quantity_shipped - invoice_quantity ) as SumReturnQty,
  SUM ( (quantity_shipped - invoice_quantity) *
      (invoice_amount/invoice_quantity) ) AS SumReturnAmt
 FROM W_INVOICELINE_F INNER JOIN W_TIME_D
     ON W_INVOICELINE_F.INVOICE_SENT_DATE = W_TIME_D.TIME_ID
   INNER JOIN W_Location_D
     ON W_INVOICELINE_F.Location_Id = W_Location_D.Location_Id
   INNER JOIN W_Sales_Class_D
     ON W_INVOICELINE_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id
 WHERE quantity_shipped > invoice_quantity
 GROUP BY W_Location_D.Location_Id, Location_Name,
  W_Sales_Class_D.Sales_Class_Id, Sales_Class_Desc, Time_Year, Time_Month;
```

Base Query 5 (BQ5)

BQ5 involves a nested query in the FROM clause as shown in the assignment on page 4. The outer query contains 3 base tables (job fact table along with location and sales class dimension tables) and a nested query in the FROM clause. The WHERE clause contains 2 join conditions for the tables in the outer query, a join condition with the nested query, and a condition comparing the date promised to the last shipment date. The SELECT clause in the outer query should use the *GetBusDaysDiff* function to calculate the difference in business days. The outer query should not contain a GROUP BY clause. To facilitate formulation of analytic queries, the base query should be placed in a CREATE VIEW statement.

```
-- BQ5 in the quality control area
-- Jobs with delays in the last shipment date (Date_Promised)
-- Nested query in the FROM clause to determine last shipment date

SELECT W_JOB_F.job_ID,
  W_JOB_F.SALES_CLASS_ID, Sales_Class_Desc,
  W_JOB_F.LOCATION_ID, Location_Name,
  Date_Promised, Last_Shipment_Date,
  QUANTITY_ORDERED, SumDelayShipQty,
  GetBusDaysDiff ( date_promised, Last_Shipment_Date ) AS BusDaysDiff
FROM W_JOB_F , W_Location_D, W_Sales_Class_D,
  (SELECT W_SUB_JOB_F.JOB_ID,
    MAX(actual_ship_Date)   AS Last_Shipment_Date,
    SUM ( actual_Quantity ) AS SumDelayShipQty
  FROM W_JOB_SHIPMENT_F, W_SUB_JOB_F, W_Job_F
  WHERE W_SUB_JOB_F.SUB_JOB_ID = W_JOB_SHIPMENT_F.SUB_JOB_ID
    AND W_Job_F.Job_Id = W_SUB_JOB_F.JOB_ID
    AND Actual_Ship_Date > Date_Promised
  GROUP BY W_SUB_JOB_F.JOB_ID
  ) X1
WHERE date_promised < X1.Last_Shipment_Date
  AND W_JOB_F.JOB_ID  = X1.Job_Id
  AND W_Job_F.Location_Id = W_Location_D.Location_Id
  AND W_Job_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id;

-- CREATE VIEW statement using the base query
CREATE VIEW LastShipmentDelays AS
 SELECT W_JOB_F.job_ID ,
  W_JOB_F.SALES_CLASS_ID, Sales_Class_Desc,
  W_JOB_F.LOCATION_ID, Location_Name,
  Date_Promised, Last_Shipment_Date,
  QUANTITY_ORDERED, SumDelayShipQty,
  GetBusDaysDiff ( date_promised, Last_Shipment_Date ) AS BusDaysDiff
FROM W_JOB_F , W_Location_D, W_Sales_Class_D,
  (SELECT W_SUB_JOB_F.JOB_ID,
    MAX(actual_ship_Date)   AS Last_Shipment_Date,
    SUM ( actual_Quantity ) AS SumDelayShipQty
  FROM W_JOB_SHIPMENT_F, W_SUB_JOB_F, W_Job_F
  WHERE W_SUB_JOB_F.SUB_JOB_ID = W_JOB_SHIPMENT_F.SUB_JOB_ID
    AND W_Job_F.Job_Id = W_SUB_JOB_F.JOB_ID
    AND Actual_Ship_Date > Date_Promised
  GROUP BY W_SUB_JOB_F.JOB_ID
  ) X1
WHERE date_promised < X1.Last_Shipment_Date
  AND W_JOB_F.JOB_ID  = X1.Job_Id
  AND W_Job_F.Location_Id = W_Location_D.Location_Id
  AND W_Job_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id;
```

Base Query 6 (BQ6)

BQ6 involves a nested query in the FROM clause as shown in the assignment on page 4. The outer query contains 3 base tables (job fact table and location and sales class dimension tables) and a nested query in the FROM clause. The WHERE clause contains 2 join conditions for the tables in the outer query, a join condition with the nested query, and a condition comparing the shipped by date to the first shipment date. The SELECT clause in the outer query should use the *GetBusDaysDiff* function to calculate the difference in business days. The outer query should not contain a GROUP BY clause. To facilitate formulation of analytic queries, the base query should be placed in a CREATE VIEW statement.

```
-- BQ6 in the quality control area
-- Jobs with delays in the first shipment date (Date_Ship_By)
-- Requires a nested query in the FROM clause to determine first shipment
date

SELECT W_JOB_F.job_ID,
  W_JOB_F.SALES_CLASS_ID, Sales_Class_Desc,
  W_JOB_F.LOCATION_ID, Location_Name,
  Date_Ship_By,
  FirstShipDate,
  GetBusDaysDiff ( date_ship_By, FirstShipDate ) AS BusDaysDiff
FROM W_JOB_F, W_Location_D, W_Sales_Class_D,
  (SELECT W_SUB_JOB_F.JOB_ID, MIN(actual_ship_Date) as FirstShipDate
   FROM W_JOB_SHIPMENT_F, W_SUB_JOB_F
   WHERE W_SUB_JOB_F.SUB_JOB_ID = W_JOB_SHIPMENT_F.SUB_JOB_ID
   GROUP BY W_SUB_JOB_F.JOB_ID
   ) X1
WHERE date_ship_By < X1.FirstShipDate
  AND W_JOB_F.JOB_ID = X1.Job_Id
  AND W_Job_F.Location_Id = W_Location_D.Location_Id
  AND W_Job_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id;

-- CREATE VIEW statement using the base query
CREATE VIEW FirstShipmentDelays AS
 SELECT W_JOB_F.job_ID,
  W_JOB_F.SALES_CLASS_ID, Sales_Class_Desc,
  W_JOB_F.LOCATION_ID, Location_Name,
  Date_Ship_By,
  FirstShipDate,
  GetBusDaysDiff ( date_ship_By, FirstShipDate ) AS BusDaysDiff
FROM W_JOB_F , W_Location_D, W_Sales_Class_D,
  (SELECT W_SUB_JOB_F.JOB_ID, MIN(actual_ship_Date) as FirstShipDate
   FROM W_JOB_SHIPMENT_F, W_SUB_JOB_F
   WHERE W_SUB_JOB_F.SUB_JOB_ID = W_JOB_SHIPMENT_F.SUB_JOB_ID
   GROUP BY W_SUB_JOB_F.JOB_ID
   ) X1
WHERE date_ship_By < X1.FirstShipDate AND W_JOB_F.JOB_ID = X1.Job_Id
  AND W_Job_F.Location_Id = W_Location_D.Location_Id
  AND W_Job_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id;
```

Analytic Query Formulation

Analytic queries involve base queries or views defined with using base queries. Here are important details of each analytic query along with SELECT statements.

Analytic Query 1 (AQ1)

AQ1 extends BQ1 with a window comparison involving the cumulative sum of the order amount, partitioned for each combination of location name and year. Note the analytic function specification for the CumSumAmt computed column.

```
-- Analytic Query 1 (AQ1)
-- Cumulative amount ordered by location, year, and month

SELECT
  Location_Name, Time_Year, Time_Month,
  SUM ( QUANTITY_ORDERED * Unit_Price ) AS SumJobAmt,
  SUM ( SUM ( QUANTITY_ORDERED * Unit_Price ) )
    OVER ( PARTITION BY Location_Name, Time_Year
      ORDER BY Time_Month
      ROWS UNBOUNDED PRECEDING ) AS CumSumAmt
FROM W_JOB_F, W_Location_D, W_TIME_D
WHERE W_Location_D.Location_ID = W_Job_F.Location_Id
  AND W_JOB_F.CONTRACT_DATE = W_TIME_D.Time_ID
GROUP BY Location_Name, Time_Year, Time_Month;
```

Analytic Query 2 (AQ2)

AQ2 extends BQ1 with a window comparison involving the moving average of the average order amount, partitioned by location name with criteria of year and month. The moving average is calculated over the current row and 11 preceding rows. In the SELECT statement, note the analytic function specification for the MovAvgAmtOrdered computed column.

```
-- Analytic query 2 (AQ2)
-- Moving average over current row and 11 preceding rows of average amount
-- Partitioned by location name
-- Ordering criteria by year and month

SELECT Location_Name, Time_Year, Time_Month,
  AVG( QUANTITY_ORDERED * Unit_Price ) AS AvgJobAmount ,
  AVG( AVG( QUANTITY_ORDERED * Unit_Price ) )
   OVER ( PARTITION BY  Location_Name
   ORDER BY Time_Year, Time_Month
   ROWS BETWEEN 11 PRECEDING AND CURRENT ROW ) AS MovAvgAmtOrdered
 FROM W_JOB_F, W_Location_D, W_TIME_D
 WHERE W_Location_D.Location_ID = W_Job_F.Location_Id
```

```
   AND W_JOB_F.CONTRACT_DATE = W_TIME_D.Time_ID
 GROUP BY Location_Name, Time_Year, Time_Month;
```

Analytic Query 3 (AQ3)

AQ3 extends BQ2 and BQ3 with ranking of locations by sum of profit. Ranking starts over for each contract year. Two SELECT statement solutions are shown. The first and simpler solution uses views containing SELECT statements for BQ2 and BQ3. The WHERE clause contains a join condition on Job_Id combining the views for BQ2 and BQ3. The second and more complex solution uses base queries for BQ2 and BQ3 in the FROM clause instead of views. The WHERE clause contains a join condition on Job_Id combining the nested queries for BQ2 and BQ3. In both solutions, profit is computed AS SumInvoiceAmt – TotalCosts.

```
-- Analytic query AQ3
-- Rank locations by descending sum of annual profit
-- Extends BQ2 and BQ3
-- Using views for location revenue and location cost summaries

SELECT X1.Location_Name, X1.Time_Year,
       SUM(SumInvoiceAmt - TotalCosts) AS SumLocProfit,
       RANK() OVER ( PARTITION BY X1.Time_Year
        ORDER BY ( SUM(SumInvoiceAmt - TotalCosts) ) DESC ) AS RankProfitSum
 FROM LocCostSummary X1, LocRevenueSummary X2
 WHERE X1.Job_Id = X2.Job_Id
 GROUP BY X1.Location_Name, X1.Time_Year;

-- Using base queries for location revenue and location cost summaries

SELECT X1.Location_Name, X1.Time_Year,
       SUM(SumInvoiceAmt - TotalCosts) AS SumLocProfit,
       RANK() OVER ( PARTITION BY X1.Time_Year
        ORDER BY ( SUM(SumInvoiceAmt - TotalCosts) ) DESC )  AS RankProfitSum
FROM
(
 SELECT W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
        W_LOCATION_D.LOCATION_NAME,
        W_TIME_D.TIME_YEAR,  W_TIME_D.TIME_MONTH,
        SUM (Invoice_Quantity) AS SumInvoiceQty,
        SUM (Invoice_Amount) AS SumInvoiceAmt
  FROM W_Job_Shipment_F, W_Sub_Job_F, W_Location_D, W_Time_D,
       W_InvoiceLine_F, W_Job_F
  WHERE W_Sub_Job_F.Sub_Job_Id = W_Job_Shipment_F.Sub_Job_Id
    AND W_Job_Shipment_F.Invoice_Id = W_InvoiceLine_F.Invoice_Id
    AND W_Time_D.Time_Id = Contract_Date
    AND W_Location_D.Location_Id = W_InvoiceLine_F.Location_Id
    AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
  GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
           W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR,
           W_TIME_D.TIME_MONTH
 ) X1,
 (
 SELECT W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
        W_LOCATION_D.LOCATION_NAME,
        W_TIME_D.TIME_YEAR,  W_TIME_D.TIME_MONTH,
        SUM(Cost_Labor) AS SumLaborCosts,
        SUM(Cost_Material) AS SumMaterialCosts,
        SUM(Cost_Overhead) AS SumOvrhdCosts,
        SUM(Machine_Hours * Rate_Per_Hour) AS SumMachineCosts,
        SUM(Quantity_Produced) AS SumQtyProduced,
        SUM(Cost_Labor + Cost_Material + Cost_Overhead +
           (Machine_Hours * Rate_Per_Hour)) AS TotalCosts
  FROM W_Job_F, W_Sub_Job_F, W_Location_D, W_Time_D, W_Machine_Type_D
  WHERE W_Job_F.Location_Id = W_Location_D.Location_Id
    AND W_Sub_Job_F.Machine_Type_Id = W_Machine_Type_D.Machine_Type_Id
    AND W_Time_D.Time_Id = Contract_Date
    AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
  GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
           W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR,
           W_TIME_D.TIME_MONTH
 ) X2
 WHERE X1.Job_Id = X2.Job_Id
 GROUP BY X1.Location_Name, X1.Time_Year;
```

Analytic Query 4 (AQ4)

AQ4 extends BQ2 and BQ3 with ranking of locations by annual profit margin. Ranking starts over for each contract year. Two SELECT statement solutions are shown. The first and simpler solution uses views containing SELECT statements for BQ2 and BQ3. The WHERE clause contains a join condition on Job_Id combining the views for BQ2 and BQ3. The second and more complex solution uses base queries for BQ2 and BQ3 in the FROM clause instead of views. The WHERE clause contains a join condition on Job_Id combining the nested queries for BQ2 and BQ3. In both solutions, annual profit margin is computed AS SUM(SumInvoiceAmt – TotalCosts) / SUM (SumInvoiceAmt).

```
-- Analytic query AQ4
-- Rank locations by descending annual profit margin
-- Extends BQ2 and BQ3
-- Using views for location revenue and location cost summaries

SELECT X1.Location_Name, X1.Time_Year,
       SUM (SumInvoiceAmt - TotalCosts) / SUM(SumInvoiceAmt) AS ProfitMargin,
       RANK() OVER ( PARTITION BY X1.Time_Year
        ORDER BY ( SUM (SumInvoiceAmt - TotalCosts) / SUM(SumInvoiceAmt) )
DESC )  AS RankProfitMargin
FROM LocCostSummary X1, LocRevenueSummary X2
WHERE X1.Job_Id = X2.Job_Id
GROUP BY X1.Location_Name, X1.Time_Year;

-- Base queries for location revenue and location cost summaries

SELECT X1.Location_Name, X1.Time_Year,
       SUM (SumInvoiceAmt - TotalCosts) / SUM(SumInvoiceAmt) AS ProfitMargin,
       RANK() OVER ( PARTITION BY X1.Time_Year
        ORDER BY ( SUM (SumInvoiceAmt - TotalCosts) / SUM(SumInvoiceAmt) )
          DESC )  AS RankProfitMargin
 FROM
 (
  SELECT W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
         W_LOCATION_D.LOCATION_NAME,  W_TIME_D.TIME_YEAR,
         W_TIME_D.TIME_MONTH, SUM (Invoice_Quantity) AS SumInvoiceQty,
         SUM (Invoice_Amount) AS SumInvoiceAmt
  FROM W_Job_Shipment_F, W_Sub_Job_F, W_Location_D, W_Time_D,
      _InvoiceLine_F, W_Job_F
  WHERE W_Sub_Job_F.Sub_Job_Id = W_Job_Shipment_F.Sub_Job_Id
    AND W_Job_Shipment_F.Invoice_Id = W_InvoiceLine_F.Invoice_Id
    AND W_Time_D.Time_Id = Contract_Date
    AND W_Location_D.Location_Id = W_InvoiceLine_F.Location_Id
    AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
  GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
           W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR,
           W_TIME_D.TIME_MONTH
  ) X1,
  (
  SELECT W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
         W_LOCATION_D.LOCATION_NAME,
         W_TIME_D.TIME_YEAR,  W_TIME_D.TIME_MONTH,
         SUM(Cost_Labor) AS SumLaborCosts,
         SUM(Cost_Material) AS SumMaterialCosts,
         SUM(Cost_Overhead) AS SumOvrhdCosts,
         SUM(Machine_Hours * Rate_Per_Hour) AS SumMachineCosts,
         SUM(Quantity_Produced) AS SumQtyProduced,
         SUM(Cost_Labor + Cost_Material + Cost_Overhead +
            (Machine_Hours * Rate_Per_Hour)) AS TotalCosts
   FROM W_Job_F, W_Sub_Job_F, W_Location_D, W_Time_D, W_Machine_Type_D
   WHERE W_Job_F.Location_Id = W_Location_D.Location_Id
     AND W_Sub_Job_F.Machine_Type_Id = W_Machine_Type_D.Machine_Type_Id
     AND W_Time_D.Time_Id = Contract_Date
     AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
   GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
            W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR,
            W_TIME_D.TIME_MONTH
  ) X2
 WHERE X1.Job_Id = X2.Job_Id
 GROUP BY X1.Location_Name, X1.Time_Year;
```

Analytic Query 5 (AQ5)

AQ5 extends BQ2 and BQ3 with percent ranking of jobs by profit margins. A single percent ranking is computed without partitioning. Two SELECT statement solutions are shown. The first and simpler solution uses views containing SELECT statements for BQ2 and BQ3. The WHERE clause contains a join condition on Job_Id combining the views for BQ2 and BQ3. The second and more complex solution uses base queries for BQ2 and BQ3 in the FROM clause instead of views. The WHERE clause contains a join condition on Job_Id combining the nested queries for BQ2 and BQ3. In both solutions, profit margin is computed AS (SumInvoiceAmt – TotalCosts) / SumInvoiceAmt.

```
-- Analytic query AQ5
-- Percent rank jobs by annual profit margin
-- Extends BQ2 and BQ3
-- Using views for location revenue and location cost summaries

SELECT X1.Job_Id, X1.Location_Name, X1.Time_Year, X1.Time_Year,
       (SumInvoiceAmt - TotalCosts) / SumInvoiceAmt AS ProfitMargin,
       PERCENT_RANK() OVER (
        ORDER BY ( (SumInvoiceAmt - TotalCosts) / SumInvoiceAmt ) )
         AS PercentRankProfitMargin
 FROM LocCostSummary X1, LocRevenueSummary X2
 WHERE X1.Job_Id = X2.Job_Id;

-- Using base queries for location revenue and location cost summaries

SELECT X1.Job_Id, X1.Location_Name, X1.Time_Year, X1.Time_Month,
      (SumInvoiceAmt - TotalCosts) / SumInvoiceAmt AS ProfitMargin,
      PERCENT_RANK() OVER (
        ORDER BY ( (SumInvoiceAmt - TotalCosts) / SumInvoiceAmt ) )
         AS PercentRankProfitMargin
 FROM
(
 SELECT W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
        W_LOCATION_D.LOCATION_NAME,
        W_TIME_D.TIME_YEAR,  W_TIME_D.TIME_MONTH,
        SUM (Invoice_Quantity) AS SumInvoiceQty,
        SUM (Invoice_Amount) AS SumInvoiceAmt
  FROM W_Job_Shipment_F, W_Sub_Job_F, W_Location_D, W_Time_D,
       W_InvoiceLine_F, W_Job_F
  WHERE W_Sub_Job_F.Sub_Job_Id = W_Job_Shipment_F.Sub_Job_Id
    AND W_Job_Shipment_F.Invoice_Id = W_InvoiceLine_F.Invoice_Id
    AND W_Time_D.Time_Id = Contract_Date
    AND W_Location_D.Location_Id = W_InvoiceLine_F.Location_Id
    AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
  GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
     W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR, W_TIME_D.TIME_MONTH
 ) X1,
 (
 SELECT W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
        W_LOCATION_D.LOCATION_NAME,
        W_TIME_D.TIME_YEAR,  W_TIME_D.TIME_MONTH,
        SUM(Cost_Labor) AS SumLaborCosts,
        SUM(Cost_Material) AS SumMaterialCosts,
        SUM(Cost_Overhead) AS SumOvrhdCosts,
        SUM(Machine_Hours * Rate_Per_Hour) AS SumMachineCosts,
        SUM(Quantity_Produced) AS SumQtyProduced,
        SUM(Cost_Labor + Cost_Material + Cost_Overhead +
           (Machine_Hours * Rate_Per_Hour)) AS TotalCosts
  FROM W_Job_F, W_Sub_Job_F, W_Location_D, W_Time_D, W_Machine_Type_D
  WHERE W_Job_F.Location_Id = W_Location_D.Location_Id
    AND W_Sub_Job_F.Machine_Type_Id = W_Machine_Type_D.Machine_Type_Id
    AND W_Time_D.Time_Id = Contract_Date
    AND W_Job_F.Job_Id = W_Sub_Job_F.Job_Id
  GROUP BY W_Sub_Job_F.Job_Id, W_Location_D.LOCATION_ID,
    W_LOCATION_D.LOCATION_NAME, W_TIME_D.TIME_YEAR, W_TIME_D.TIME_MONTH
  ) X2
 WHERE X1.Job_Id = X2.Job_Id;
```

Analytic Query 6 (AQ6)

AQ6 extends AQ5 directly (and BQ2 and BQ3 indirectly) with the top 5% of job profit margins. The WHERE clause in the outer query contains a condition on the percent rank computed in the nested query in the FROM clause. AQ5 is used in the FROM clause.

```
-- Analytic query AQ6
-- Top performers of percent rank of job profit margins
-- Using SELECT statement of AQ5 in the FROM clause

SELECT Job_Id, Location_Name, Time_Year, Time_Month,
       ProfitMargin, PercentRankProfitMargin
 FROM (
  SELECT X1.Job_Id, X1.Location_Name, X1.Time_Year, X1.Time_Month,
         (SumInvoiceAmt - TotalCosts) / SumInvoiceAmt AS ProfitMargin,
         PERCENT_RANK() OVER (
         ORDER BY ( (SumInvoiceAmt - TotalCosts) / SumInvoiceAmt ) )
          AS PercentRankProfitMargin
  FROM LocCostSummary X1, LocRevenueSummary X2
  WHERE X1.Job_Id = X2.Job_Id )
 WHERE PercentRankProfitMargin > 0.95;
```

Analytic Query 7 (AQ7)

AQ6 extends BQ4 with ranking of sales classes by the sum of the return quantity. The ranking restarts on every year. The WHERE clause contains a condition that quantity shipped is greater than invoice quantity. This condition comes from the base query, BQ4.

```
-- Analytic query AQ7
-- Rank sales class by sum of return quantities
-- Partition rank by year

SELECT Sales_Class_Desc, Time_Year,
  SUM ( quantity_shipped - invoice_quantity ) as ReturnSum ,
  RANK() over ( PARTITION BY Time_Year
    ORDER BY SUM ( quantity_shipped - invoice_quantity ) DESC )
     AS RankReturnSum
 FROM W_INVOICELINE_F INNER JOIN W_TIME_D
     ON W_INVOICELINE_F.INVOICE_SENT_DATE = W_TIME_D.TIME_ID
  INNER JOIN W_Sales_Class_D
     ON W_INVOICELINE_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id
 WHERE quantity_shipped > invoice_quantity
 GROUP BY Sales_Class_Desc, Time_Year;
```

Analytic Query 8 (AQ8)

AQ8 extends BQ4 with ratio to report of sales classes by the sum of the return quantity. The ranking restarts on every year. The WHERE clause contains a condition that quantity shipped is greater than invoice quantity. This condition comes from the base query, BQ4. The ORDER BY clause ensures a convenient ordering by year and return quantity.

```
-- Analytic query AQ8
-- Ratio to report for sales classes on sum of return quantity
-- Partition ratio to report by year

SELECT Time_Year, Sales_Class_Desc,
  SUM ( quantity_shipped - invoice_quantity ) as SumReturnQty,
  Ratio_To_Report(SUM ( quantity_shipped - invoice_quantity ))
    OVER ( PARTITION BY Time_Year ) AS RatioReturnSum
 FROM W_INVOICELINE_F INNER JOIN W_TIME_D
      ON W_INVOICELINE_F.INVOICE_SENT_DATE = W_TIME_D.TIME_ID
   INNER JOIN W_Sales_Class_D
      ON W_INVOICELINE_F.Sales_Class_Id = W_Sales_Class_D.Sales_Class_Id
 WHERE quantity_shipped > invoice_quantity
 GROUP BY Sales_Class_Desc, Time_Year
 ORDER BY Time_Year, SUM( quantity_shipped - invoice_quantity );
```

Analytic Query 9 (AQ9)

AQ9 extends BQ6 with ranking of locations on the sum of the business days delayed. BQ6 involves delays on the first shipment date compared to the date shipped by in the job. Both ranking functions should be used. The ranking restarts on each year of the date promised. The FROM clause combines the view for BQ6 (FirstShipmentDelays) and the time dimension table.

The WHERE clause contains a join condition on Time_Id of the time dimension table with the Date_Promised from the view.

```
-- Analytic query AQ9
-- Rank locations by sum of business days delayed
-- Partition ranking by year of date promised
-- Use both rank and dense_rank functions
-- Uses FirstShipmentDelays view (based on BQ6)

SELECT Location_Name, W_Time_D.Time_Year,
       SUM(BusDaysDiff) as SumDelayDays,
  RANK() OVER ( PARTITION BY W_Time_D.Time_Year
    ORDER BY SUM(BusDaysDiff) DESC) AS RankSumDelayDays,
  DENSE_RANK() OVER ( PARTITION BY W_Time_D.Time_Year
    ORDER BY SUM(BusDaysDiff) DESC) AS RankSumDelayDays
 FROM FirstShipmentDelays, W_Time_D
 WHERE W_Time_D.Time_Id = FirstShipmentDelays.Date_Ship_By
 GROUP BY Location_Name, W_Time_D.Time_Year;
```

Analytic Query 9 (AQ9)

AQ9 extends BQ6 with ranking of locations on the sum of the business days delayed for locations. BQ5 involves delays on the first shipment date compared to the date shipped by in the job. Both ranking functions should be used. The ranking restarts on each year of the date promised. The FROM clause combines the view for BQ6 (LastShipmentDelays) and time dimension table. The WHERE clause contains a join condition on Time_Id of the time dimension table with the Date_Ship_By of the view.

```
-- Analytic query AQ9
-- Rank locations by sum of business days delayed
-- Partition ranking by year of shipped by date
-- Use both rank and dense_rank functions
-- Uses FirstShipmentDelays view (based on BQ6)

SELECT Location_Name, W_Time_D.Time_Year,
       SUM(BusDaysDiff) as SumDelayDays,
  RANK() OVER ( PARTITION BY W_Time_D.Time_Year
    ORDER BY SUM(BusDaysDiff) DESC) AS RankSumDelayDays,
  DENSE_RANK() OVER ( PARTITION BY W_Time_D.Time_Year
    ORDER BY SUM(BusDaysDiff) DESC) AS RankSumDelayDays
 FROM FirstShipmentDelays, W_Time_D
 WHERE W_Time_D.Time_Id = FirstShipmentDelays.Date_Ship_By
 GROUP BY Location_Name, W_Time_D.Time_Year;
```

Analytic Query 10 (AQ10)

AQ10 extends BQ5 with ranking of locations on the delay rate. BQ5 involves delays on the last shipment date compared to the date promised in the job. The ranking restarts on each year of the date promised. The FROM clause combines the view for BQ5 (LastShipmentDelays) and the time dimension table. The WHERE clause contains a join condition on Time_Id of the time dimension table with the Date_Promised of the view.

```
-- Analytic query AQ9
-- Rank locations by delay rate for the contract promised date
-- Partition ranking by year of date promised
-- Delay rate calculated as SUM(Quantity_Ordered - SumDelayShipQty) /
--                          SUM(Quantity_Ordered)
-- Uses LastShipmentDelays view (based on BQ5)

SELECT Location_Name, W_Time_D.Time_Year,
       COUNT(*) AS NumJobs,
       SUM(BusDaysDiff) as SumDelayDays,
       SUM(Quantity_Ordered - SumDelayShipQty) / SUM(Quantity_Ordered)
        AS PromisedDelayRate,
  RANK() OVER ( PARTITION BY W_Time_D.Time_Year
    ORDER BY SUM(Quantity_Ordered - SumDelayShipQty) /
             SUM(Quantity_Ordered) DESC) AS RankDelayRate
 FROM LastShipmentDelays, W_Time_D
 WHERE W_Time_D.Time_Id = LastShipmentDelays.Date_Promised
 GROUP BY Location_Name, W_Time_D.Time_Year;
```