**Solution 1**

**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 th

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,

) X1,

W\_LOCATION\_D.LOCATION\_NAME, W\_TIME\_D.TIME\_YEAR, W\_TIME\_D.TIME\_MONTH

(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;