Exam of *Decision Support Databases: Example 1*

1. **(5 points) (Mandatory)** Let us consider the following database, without null values:

Products

PkP	UnitPrice	...
10	5	...
20	10	...
30	20	...

Sales

FkP	Qty	...
10	50	...
20	10	...
30	20	...
10	30	...
20	100	...
30	10	...
10	30	...

   (a) Write an SQL query to find the total sales revenue by product.

   (b) Give a logical query plan for the SQL query, the type and the value of the result.
   Modify the logical query plan to consider only products with UnitPrice > 5 sold each of them more than 5 times.

   (c) Modify the SQL query to find also the rank of the product total quantity sold (the highest is first)).

   (d) Show the instance of an index on the attribute FkP.

   (e) Show the instance of a *Foreign Column Join Index* on the attribute UnitPrice.

2. See lecture notes Exercise A.4: Inventory.

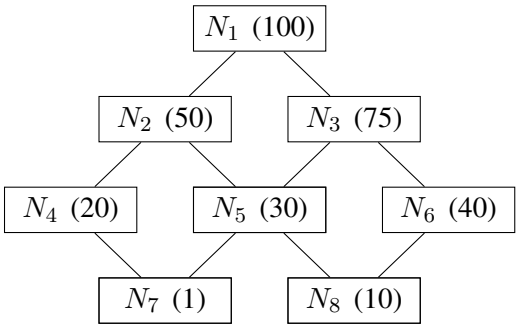
   With respect to the above business scenario, answer the following questions:

   (a) **(8 points)** Design a conceptual schema of a data mart to support at least the examples of the business questions given. Specify the fact granularity, and for each measure, if it is additive, semi-additive or non-additive.

   (b) **(5 points)** Design the logical schema of the data mart, and write the SQL queries for *two* business questions of your choice.

   (c) **(4 points)** Give the physical query plan to execute a query of the previous point using a *foreign column join index* among those needed to speed up the query.

3. **(3 points)** Explain briefly the problem of materialized views selection, without indexes. Let us consider the following lattice of possible candidate views to materialize. The numbers associated with the nodes represent the view size, measured in terms of the number of tuples in the view. Select 2 views to materialize, different from N_1, with the greedy algorithm HRU.



4. Let us consider the logical schema of a data mart

   Customer(PkCustPhoneNo, CustName, CustCity)
   CallingPlans(PkPlanId, PlanName)
   Calls(PkCustPhoneNo, FkPlanId, Day, Month, Year, Duration, Charge)

   where PkPlanId e PlanName are two different keys, and the following query

   Q: **SELECT** Year, PlanName, SUM(Charge) AS TC
   FROM Calls, CallingPlans
   WHERE FkPlanId = PkPlanId **AND** Year >= 2000 **AND** Year <=2005
   GROUP BY Year, PlanName
   HAVING SUM(Charge) > 1000;

   (a) **(3 points)** Show if and how the GROUP BY can be brought forward on the table Calls.

   (b) **(5 points)** Show if and how the query can be rewritten using the materialized view

   V1: **SELECT** FkPlanId, Month, Year, SUM(Charge) AS C
   FROM Calls
   WHERE Year >= 2000
   GROUP BY FkPlanId, Month, Year;

Exam of *Decision Support Databases: Solution Example 1 (Hints)*

1. See lecture notes.
2. (a) **Requirements specification and the conceptual design of a data mart for the inventory**. See lecture notes solution to Exercise A.4: Inventory.
   (b) **Logical design of the data mart and the SQL queries**. See lecture notes solution to Exercise A.4: Inventory.
   (c) **Physical query plan**. See lecture notes.
3. See lecture notes for the problem of materialized view selection.

   Application of the HRU algorithm:

	First Choice	Second Choice
N_2	$\mathbf{50 \times 5 = 250}$	
N_3	$25 \times 5 = 125$	$25 \times 2 = 50$
N_4	$80 \times 2 = 160$	$30 \times 2 = 60$
N_5	$70 \times 3 = 210$	$20 \times 3 = 60$
N_6	$60 \times 2 = 120$	$\mathbf{60 + 10 = 70}$
N_7	$99 \times 1 = 99$	$49 \times 1 = 49$
N_8	$90 \times 1 = 90$	$40 \times 1 = 40$

4. Let us consider the logical schema of a data mart, without null values:

   Customer(PkCustPhoneNo, CustName, CustCity)
   CallingPlans(PkPlanId, PlanName)
   Calls(PkCustPhoneNo, FkPlanId, Day, Month, Year, Duration, Charge)

   where PkPlanId and PlanName are two different keys, and the following query

```
Q:  SELECT      Year, PlanName, SUM(Charge) AS TC
    FROM        Calls, CallingPlans
    WHERE       FkPlanId = PkPlanId  AND  Year >= 2000  AND  Year <=2005
    GROUP BY    Year, PlanName
    HAVING      SUM(Charge) > 1000;
```

   (a) Show if and how the **GROUP BY** can be brought forward on the table Calls.

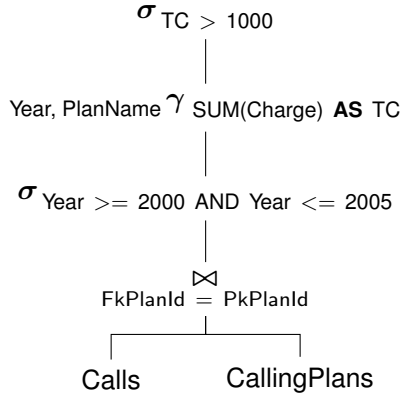


Figure 1: Logical query plan new

The selection on Year can be pushed on Calls below the join.
The group-by can be pushed on Calls below the join because the *invariant grouping* property holds: the aggregate function SUM(Charge) uses an attribute from Calls, and (Year, PlanName $\rightarrow$ FkPlanId) because

$$(\text{Year, PlanName})^{+} = \{\text{Year, PlanName, PkPlanId, FkPlanId}\}$$

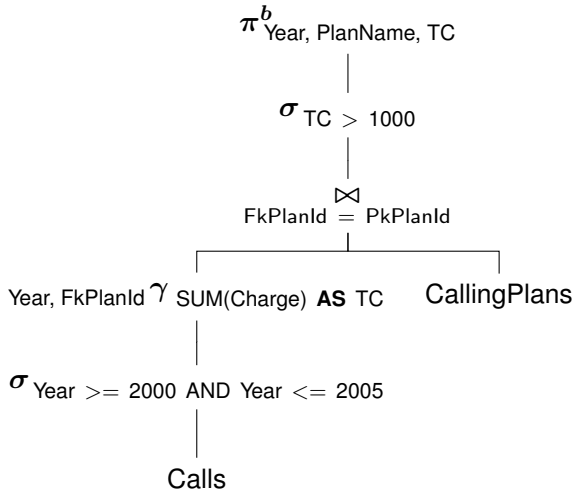


Figure 2: Logical query plan with the group-by pushed below the join

(b) Show if and how the query can be rewritten using the materialized view

V1: **SELECT** FkPlanId, Month, Year, SUM(Charge) AS C
FROM Calls
WHERE Year $>= 2000$
GROUP BY FkPlanId, Month, Year;

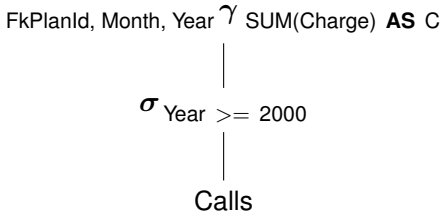


Figure 3: Logical view plan

Let us use the approach with a *transformation of the logical query plan* in Figure 2, which can be rewritten as follows with a subtree identical to V:

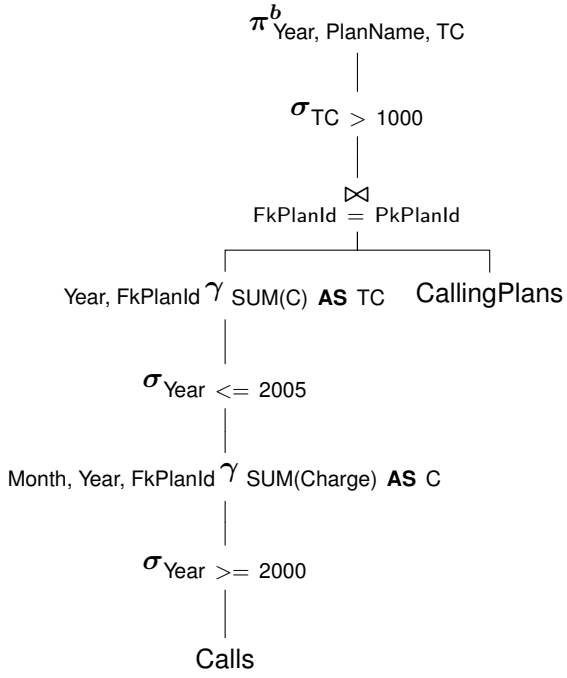


Figure 4: Logical query plan with the logical view subplan

The logical query plan with the logical view subplan is then rewritten as follows to translate it in SQL. With the rewriting of the group-by over the join, the attribute PlanName of CallingPlans, used by the projection, can be added to grouping attributes because (FkPlanId = PkPlanId) and PkPlanId $\rightarrow$ PlanName.

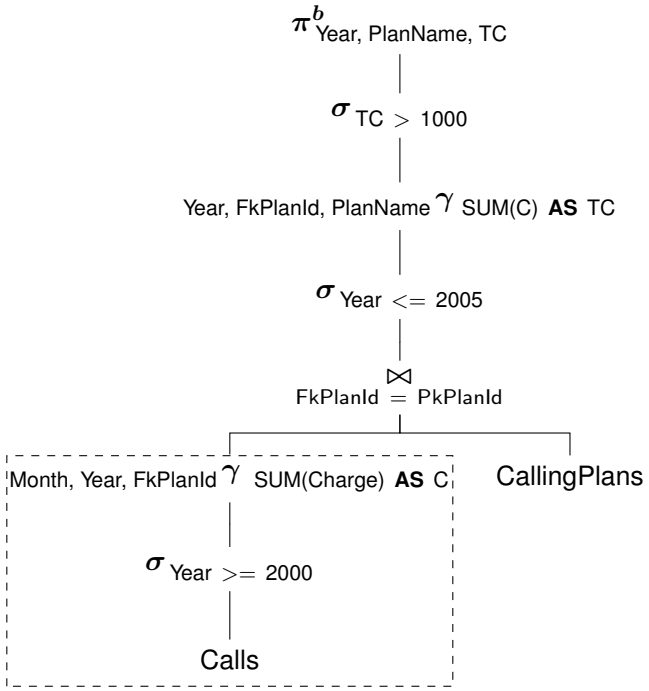


Figure 5: Final version of the logical query plan with the logical view subplan

So the rewriting of Q succeeds:

```
Q1": SELECT     Year, PlanName, SUM(C) AS TC
     FROM       V1, CallingPlans
     WHERE      FkPlanId = PkPlanId AND Year <=2005
     GROUP BY   Year, FkPlanId, PlanName
     HAVING     SUM(C) > 1000;
```

Exam of *Decision Support Databases: Example 2*

1. **(5 points) (Mandatory)** Let us consider the following database, without null values (F(Fk :int, B : int, C :int) and D(Pk :int, E :string)) and the query:

```
SELECT    Fk, COUNT(*) AS Cn
FROM      F, D
WHERE     Fk = Pk AND E <> 'd3'
GROUP BY  Fk
HAVING    SUM(C) < 100;
```

D

Pk	E
1	d1
2	d2
3	d3

F

Fk	B	C
1	10	60
1	20	20
2	30	80
2	20	25
3	30	80

   (a) Give a logical query plan for the SQL query, the type and the value of the result.

   (b) Modify the SQL query to find also the rank of the product total quantity sold (the highest is first)).

   (c) Show the instance of an index on the attribute Fk.

   (d) Show the instance of a *Foreign Column Join Index* on the attribute E.

   (e) Explain the meaning of a "semi-additive" measure. Let F(FkD1, FkD2, M) be a table with the "semi-additive" measure M with respect to the dimension D1. Give a correct and a wrong query on the schema with the aggregation SUM(M).

2. See lecture notes Exercise A.5: Hotels.

   With respect to the above business scenario, answer the following questions:

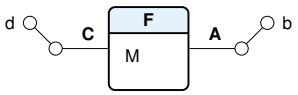
   (a) **(8 points)** Give a conceptual data mart design to support at least the examples of the business questions given. Specify the fact granularity, and for each measure, if it is additive, semi-additive or non-additive.

   (b) **(5 points)** Give a logical data mart design, and write the SQL queries for the analysis (1) and (4);

   (c) **(4 points)** Give the physical query plan to execute the first query of the previous point using (a) the physical operator HashGroupby for the grouping operator, and (b) a *foreign column join index* among those needed to speed up the query. Give a brief description of a *bitmapped foreign column join index*.

   (d) **(1 point)** Give a brief description of the TableAccess operator.

3. **(2 points)** Give the lattice of possible candidate views to materialize for the conceptual design in the following figure with dimensions A(b), C(d). Explain how the benefit of a view is computed with the greedy algorithm HRU.



4. Let us consider the database without null values:

   Customer(PKCustomer, CName, CCity)
   Order(PKOrder, FKCustomer, ODate)
   Product(PKProduct, PName, PCost)
   OrderLine(LineNo, FKOrder, FKProduct, Quantity, ExtendedPrice, Discount, Revenue)

   and the query

   Q: **SELECT** CCity, AVG(Revenue) **AS** avgR
   FROM OrderLine, Order, Customer
   WHERE FKOrder = PKOrder **AND** FKCustomer = PKCustomer
   GROUP BY CCity, FKCustomer
   HAVING SUM(Revenue) $>$ 1000;

   (a) **(2 points)** Show if and how the **GROUP BY** can be pushed on the join (OrderLine $\underset{\text{FKOrder = PKOrder}}{\bowtie}$ Order).

   (b) **(2 points)** Show if and how the **GROUP BY** can be pushed on the relation OrderLine.

   (c) **(4 points)** Show if and how the query Q can be rewritten using the materialized view V

   V: **SELECT** FKCustomer, SUM(Revenue) **AS** TR, COUNT(*) **AS** Cnt
   FROM OrderLine, Order
   WHERE FKOrder = PKOrder
   GROUP BY FKCustomer;

Exam of *Decision Support Databases: Solution Example 2 (Hints)*

1. See lecture notes.
2. (a) **Requirements specification and the conceptual design of a data mart for hotel room type utilization**. See lecture notes solution to Exercise A.5: Hotels.

   (b) **Logical design of the data mart and the SQL queries**. See lecture notes solution to Exercise A.5: Hotels.

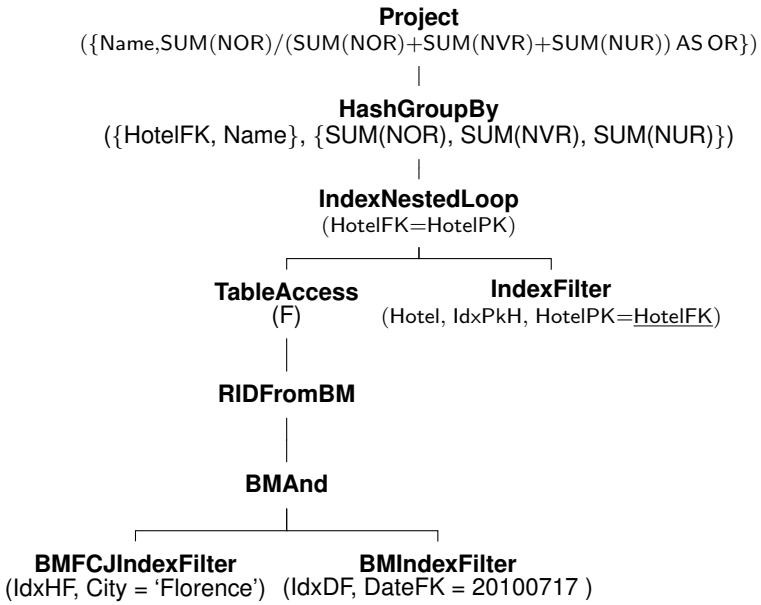
   (c) **Physical query plan** for the query

```
SELECT    H.Name
          , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                     SUM(F.NVacantRooms) +
                                     SUM(F.NUnavailableRooms) )
              AS OccupancyRate
FROM      RoomTypeUtilization F, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = 20100717
          AND H.City = 'Florence'
GROUP BY  F.HotelFK, H.Name;
```

   Let us assume that there is (a) a bitmap index on the fact table attribute DateFK, (b) a bitmapped foreign column join index on the Hotel dimensional attribute City, and (c) an index on the primary key of the dimensional tables.
   Let us use the following abbreviations:

   - F for RoomTypeUtilization.
   - NOR for NOccupiedRooms.
   - NVR for NVacantRooms.
   - NUR for NUnavailableRooms.
   - OR for OccupancyRate.

   The physical query plan for the first data analysis is



(d) See lecture notes.

3. See lecture notes.

4. Let us consider the logical schema of a data mart, without null values:

   Customer(PKCustomer, CName, CCity)
   Order(PKOrder, FKCustomer, ODate)
   Product(PKProduct, PName, PCost)
   OrderLine(LineNo, FKOrder, FKProduct, Quantity, ExtendedPrice, Discount, Revenue)

   and the query

   Q: **SELECT** CCity, AVG(Revenue) **AS** avgR
   FROM OrderLine, Order, Customer
   WHERE FKOrder = PKOrder **AND** FKCustomer = PKCustomer
   GROUP BY CCity, FKCustomer
   HAVING SUM(Revenue) $>$ 1000;

   (a) Show if and how the **GROUP BY** can be pushed on the join (OrderLine $\overset{\bowtie}{_{\text{FKOrder = PKOrder}}}$ Order).

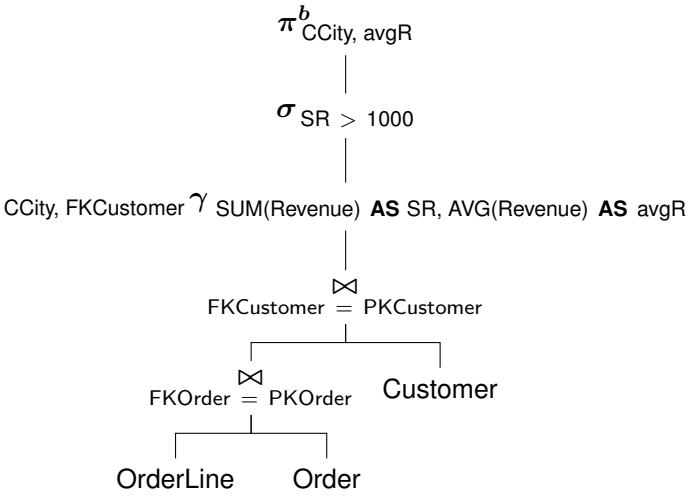


Figure 6: Logical query plan

The group-by can be pushed on the join (OrderLine $\overset{\bowtie}{_{\text{FKOrder = PKOrder}}}$ Order) because the *invariant grouping* property holds: the aggregate functions use an attribute from the join result, and (CCity, FKCustomer $\rightarrow$ FKCustomer).

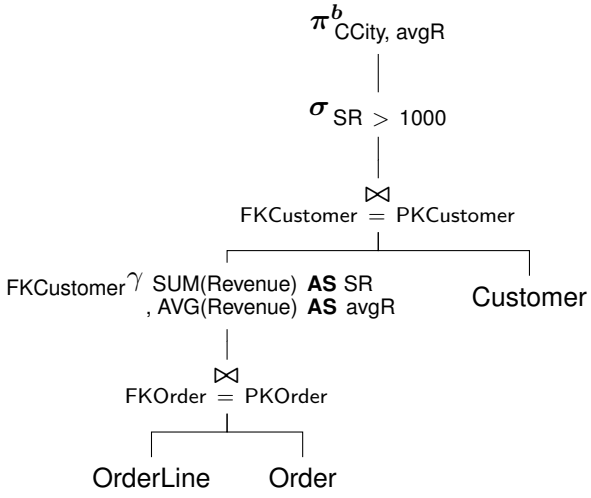


Figure 7: Logical query plan: the **GROUP BY** is pushed below the first join with the invariant grouping

(b) Show if and how the **GROUP BY** can be pushed on the relation OrderLine.

OrderLine does not have the invariant grouping property because Condition 1 does not hold: FKCustomer $\not\rightarrow$ FKOrder. The double grouping can be applied, with the rewriting of the not decomposable aggregation function AVG(Revenue) as SUM(Revenue) / COUNT(Revenue), equivalent to SUM(Revenue) / COUNT($*$) because the data mart is without null values.

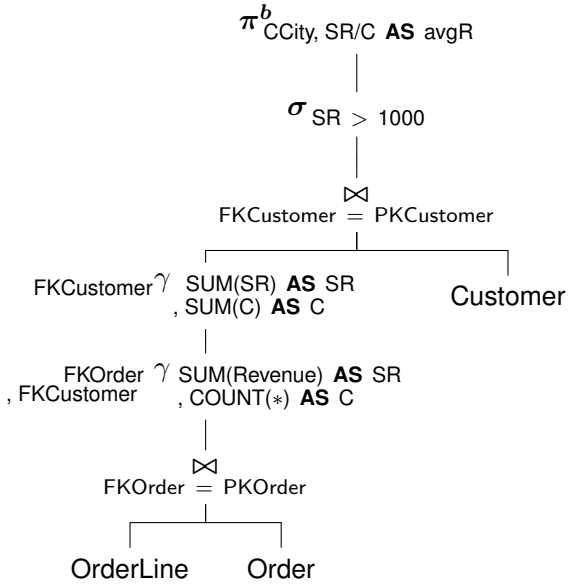


Figure 8: Logical query plan: the **GROUP BY** is rewritten with the double grouping and the rewriting of AVG

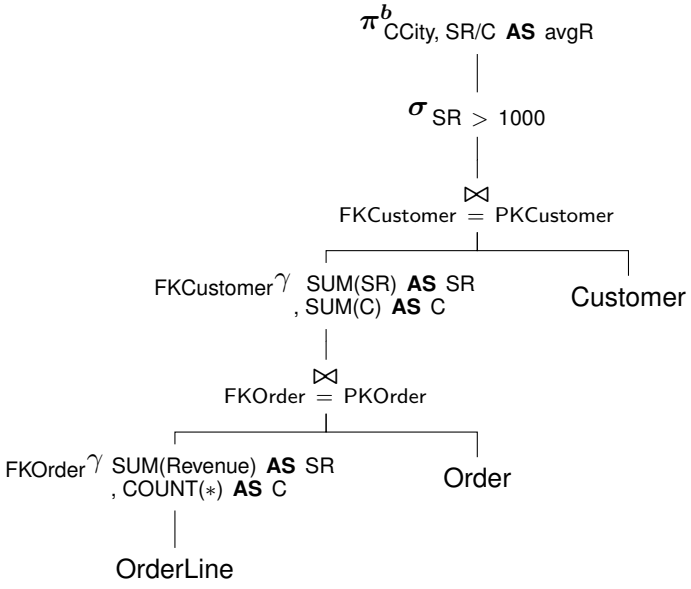


Figure 9: Logical query plan: the second **GROUP BY** is pushed below the second join with the invariant grouping

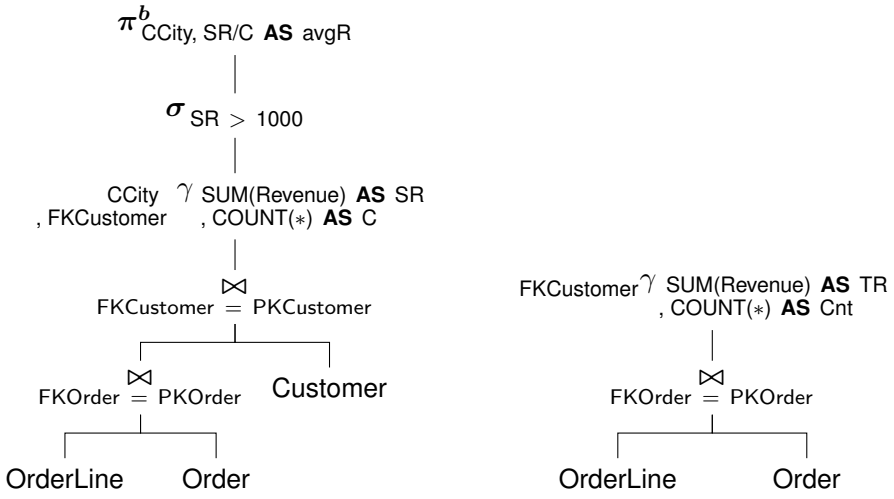
(c) Show if and how the query Q can be rewritten using the materialized view V

Q: **SELECT** CCity, AVG(Revenue) **AS** avgR
FROM OrderLine, Order, Customer
WHERE FKOrder = PKOrder **AND** FKCustomer = PKCustomer
GROUP BY CCity, FKCustomer
HAVING SUM(Revenue) $>$ 1000;

V: **SELECT** FKCustomer, SUM(Revenue) **AS** TR, COUNT(*) **AS** Cnt
FROM OrderLine, Order
WHERE FKOrder = PKOrder
GROUP BY FKCustomer;

Let us use the approach with a *compensation on the view*.

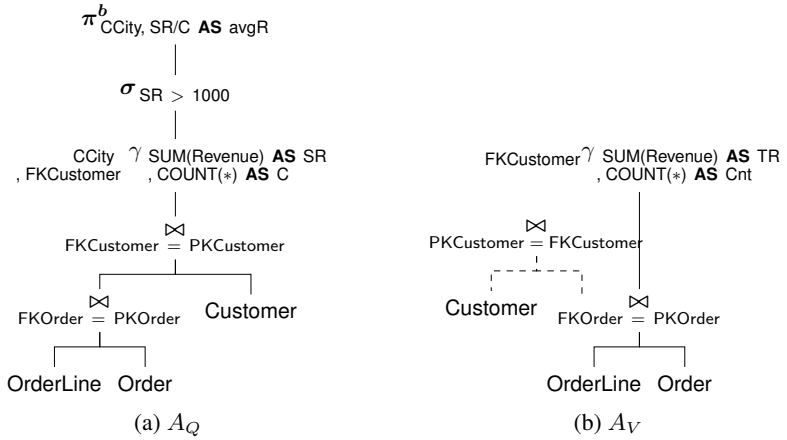
Since the approach requires that the **SELECT** and **HAVING** clauses may contain only the aggregate functions MIN, MAX, SUM and COUNT, the AVG function in Q is rewritten to compute it from given values for SUM and COUNT.



(a) A_Q (b) A_V

Figure 10: Query and view logical query plans

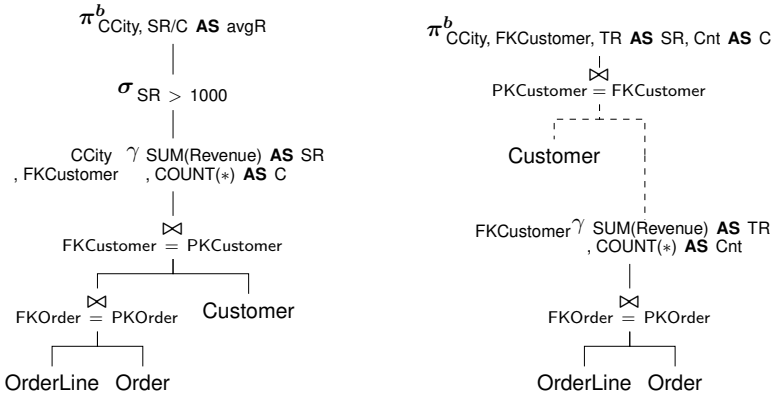
The join operations do not match. Therefore, a compensation is added as shown in the figure:



(a) A_Q (b) A_V

Figure 11: Join compensation

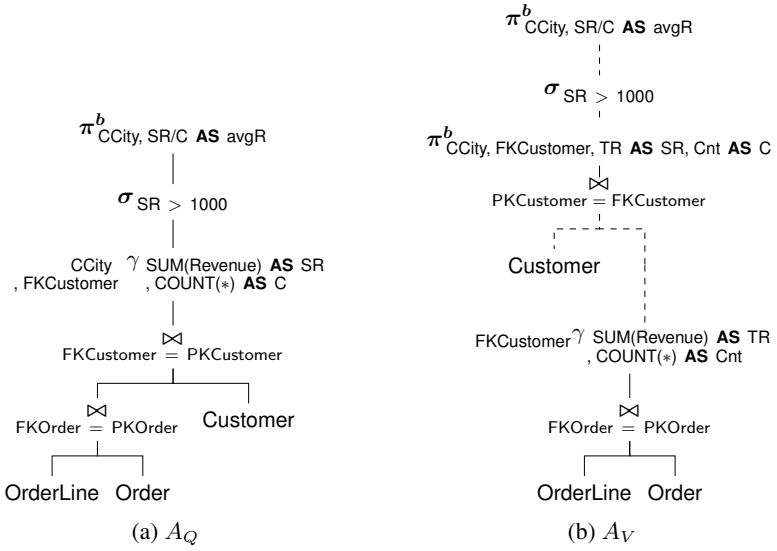
To match the groupings, the compensation on the operand of γ_V floats, and since $g(Q) \rightarrow g(V) \wedge g(V) \rightarrow g(Q)$, the rewriting does not require a grouping compensation, but an aggregate compensation only with a project, as shown in the figure:



(a) A_Q (b) A_V

Figure 12: The float of the join compensation and the groupings compensation

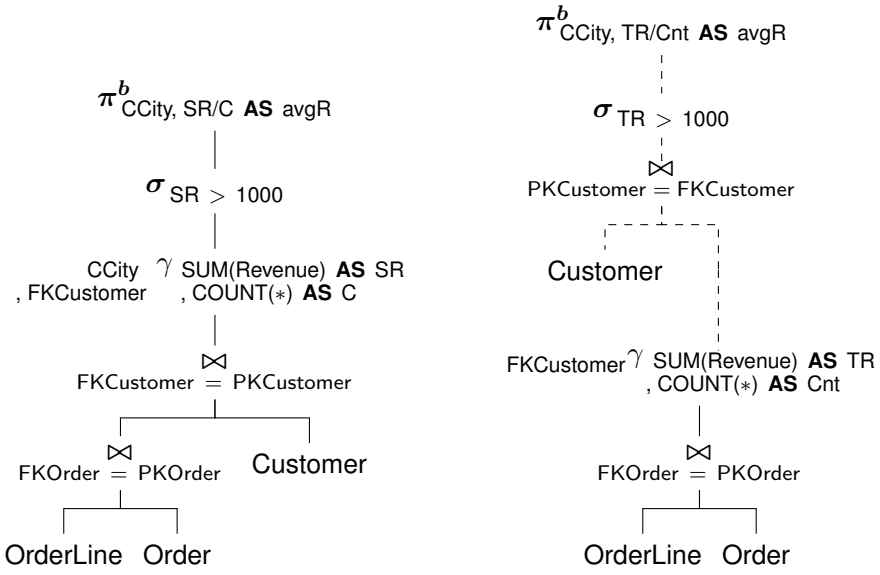
The other compensations required for the σ and π^b in Q are shown in the figure:



(a) A_Q (b) A_V

Figure 13: Other compensations

Since the internal π^b of the compensation is useless, the final solution is the following:



(a) A_Q (b) A_V

Figure 14: Final solution

Rewriting of Q:

```
QR: SELECT   CCity, TR/Cnt AS avgR
    FROM     V, Customer
    WHERE    FKCustomer = PKCustomer AND TR > 1000;
```

Let us use the approach with a *transformation of the logical query plan*.

The rewriting of Q with the γ pushed below the first join

$$(\text{OrderLine} \underset{\text{FKOrder = PKOrder}}{\bowtie} \text{Order})$$

produces a logical tree that, with the rewriting of AVG and the changing of SR and C in the γ with TR and Cnt, has as subtree the tree of the view.

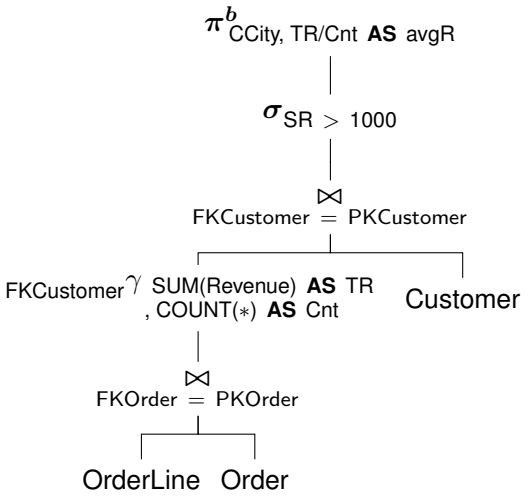


Figure 15: The rewriting of Q to have V as a subtree

Rewriting of Q:

```
QR: SELECT    CCity, TR/Cnt AS avgR
    FROM      V, Customer
    WHERE     FKCustomer = PKCustomer AND TR > 1000;
```