Exam of Decision Support Systems / Decision Support Databases: Example 1

It is forbidden to consult any material during the test. Duration of written exam is 2h.

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

<table>
<thead>
<tr>
<th>Products</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>PkP</td>
<td>UnitPrice</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
</tr>
</tbody>
</table>

(a) **(1 point)** Write an SQL query to find the total sales revenue by product.

(b) **(1 point)** 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) **(1 point)** Modify the SQL query to find also the rank of the product total quantity sold (the highest is first)).

(d) **(1 point)** Show the instance of an index on the attribute FkP.

(e) **(1 point)** Show the instance of a Foreign Column Join Index on the attribute UnitPrice.

2. **(5 points)** Consider the FoodMart datawarehouse:
Consider a customer *customer id* in a day *time id* when she/he has made a purchase. The customer is *occasional* if she/he has made purchases for at most 10 euros in the previous 45 days. Give an analytic SQL query to output for every *customer id* and *time id* that are NOT occasional, the total sales in the previous 45 days.

3. See lecture notes Exercise A.4: Inventory.
   With respect to the above business scenario, answer the following questions:
   (a) *(6 points)* Design a conceptual schema for the data mart to support the business questions. Your schema should at least be able to satisfy the above mentioned analysis requirements. You may motivate other suitable attributes for the dimensions. Specify the fact granularity, and for each measure, if it is additive, semi-additive or non-additive, and for each dimensional attribute the type of updates of the attribute.
   (b) *(3 points)* Give a logical data mart design which is able to deal with updates of dimensional attributes.
   (c) *(1 point)* Write an SQL query to answer one business questions of your choice and write a physical query plan for the query using any index of your choice.

4. *(5 points)* 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 3 views to materialize, different from $N_1$, with the greedy algorithm HRU.

![View Lattice](attachment:view_lattice.png)

5. *(5 points)* Let us consider the logical schema of a data mart

   \begin{align*}
   \text{Customer}(PK\text{CustPhoneNo}, \text{CustName}, \text{CustCity}) \\
   \text{CallingPlans}(PK\text{PlanId}, \text{PlanName}) \\
   \text{Calls}(PK\text{CustPhoneNo}, FK\text{PlanId}, \text{Day}, \text{Month}, \text{Year}, \text{Duration}, \text{Charge})
   \end{align*}

   where PK\text{PlanId} and PlanName are two different keys, and the following query

   \begin{align*}
   Q: \quad & \text{SELECT} \quad \text{Year}, \text{PlanName}, \text{SUM(Charge)} \quad \text{AS} \quad TC \\
   & \text{FROM} \quad \text{Calls, CallingPlans} \\
   & \text{WHERE} \quad \text{FKPlanId} = \text{PKPlanId} \quad \text{AND} \quad \text{Year} \geq 2000 \quad \text{AND} \quad \text{Year} \leq 2005 \\
   & \text{GROUP BY} \quad \text{Year, PlanName} \\
   & \text{HAVING} \quad \text{SUM(Charge)} > 1000;
   \end{align*}

   Answer only one of the following questions:
   (a) Show if and how the GROUP BY can be brought forward on the table Calls.
   (b) Show if and how the query can be rewritten using the materialized view

   \begin{align*}
   V1: \quad & \text{SELECT} \quad \text{FKPlanId}, \text{Month}, \text{Year}, \text{SUM(Charge)} \quad \text{AS} \quad C \\
   & \text{FROM} \quad \text{Calls} \\
   & \text{WHERE} \quad \text{Year} \geq 2000 \\
   & \text{GROUP BY} \quad \text{FKPlanId}, \text{Month}, \text{Year};
   \end{align*}
Exam of Decision Support Databases: Solution Example 1 (Hints)

1. (a) The SQL query is:

```sql
SELECT PkP, SUM(UnitPrice*Qty) AS TR
FROM Sales, Products
WHERE FkP = PkP
GROUP BY PkP;
```

(b) The logical query plan is:

```
PkP \gamma SUM(UnitPrice*Qty) AS TR

\[\iff\]

FkP = PkP

Sales

Products

The type of the result is \{(PkP: int, TR: int)\}. The value of the result is:

<table>
<thead>
<tr>
<th>PkP</th>
<th>TR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>550</td>
</tr>
<tr>
<td>20</td>
<td>1100</td>
</tr>
<tr>
<td>30</td>
<td>600</td>
</tr>
</tbody>
</table>

The modified logical query plan is:

```
\sigma C > 5

PkP \gamma SUM(UnitPrice*Qty) AS TR, COUNT(*) AS C

\[\iff\]

FkP = PkP

Sales

\[\sigma UnitPrice > 5\]

Products

(c) The modified SQL query is:

```sql
SELECT PkP, SUM(UnitPrice*Qty) AS TR, RANK() OVER (ORDER BY SUM(Qty) DESC) AS Rk
FROM Sales, Products
WHERE FkP = PkP
GROUP BY PkP;
```
(d) and (e) These are the instances of the indexes:

<table>
<thead>
<tr>
<th>Index on FkP</th>
<th>FCJ on UnitPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>FkP</td>
<td>UnitPrice</td>
</tr>
<tr>
<td>RID</td>
<td>RID</td>
</tr>
<tr>
<td>10 1</td>
<td>5 1</td>
</tr>
<tr>
<td>10 4</td>
<td>5 4</td>
</tr>
<tr>
<td>10 7</td>
<td>5 7</td>
</tr>
<tr>
<td>20 2</td>
<td>10 2</td>
</tr>
<tr>
<td>20 5</td>
<td>10 5</td>
</tr>
<tr>
<td>30 3</td>
<td>20 3</td>
</tr>
<tr>
<td>30 6</td>
<td>20 6</td>
</tr>
</tbody>
</table>

2. **WITH** `tmp` **AS** (  
   ```  
   SELECT customer_id, time_id,  
   SUM(SUM(store_sales)) OVER (  
      PARTITION BY customer_id  
      ORDER BY time_id  
      RANGE BETWEEN 45 PRECEDING AND CURRENT ROW) AS tot  
   FROM sales_fact  
   GROUP BY customer_id, time_id  
   )  
   SELECT *  
   FROM tmp  
   WHERE tot>10  
   ORDER BY customer_id, time_id;  
   ```

3. (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.

4. The application of the HRU algorithm yields:

<table>
<thead>
<tr>
<th>First Choice</th>
<th>Second Choice</th>
<th>Third Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>N₂</td>
<td>50 × 5 = 250</td>
<td></td>
</tr>
<tr>
<td>N₃</td>
<td>25 × 5 = 125</td>
<td></td>
</tr>
<tr>
<td>N₄</td>
<td>80 × 2 = 160</td>
<td></td>
</tr>
<tr>
<td>N₅</td>
<td>70 × 3 = 210</td>
<td></td>
</tr>
<tr>
<td>N₆</td>
<td>60 × 2 = 120</td>
<td></td>
</tr>
<tr>
<td>N₇</td>
<td>99 × 1 = 99</td>
<td></td>
</tr>
<tr>
<td>N₈</td>
<td>90 × 1 = 90</td>
<td></td>
</tr>
</tbody>
</table>

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

   \[(Year, \text{PlanName})^+ = \{\text{Year, PlanName, PkPlanId, FkPlanId}\}\]
(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 $V$:

$$
\pi_{Year, PlanName, TC}^b \\
\sigma_{TC > 1000} \\
FkPlanId = PkPlanId \\
\gamma_{Year < 2005} \sum(C) \text{AS} \ TC \ \text{CallingPlans} \\
\sigma_{Year <= 2005} \\
\gamma_{Month, Year, FkPlanId} \sum(Charge) \text{AS} \ C \\
\sigma_{Year >= 2000} \\
\text{Calls}
$$

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 → PlanName.

\[
\begin{array}{c}
\pi_{\text{Year}, \text{PlanName}, \text{TC}} \\
\sigma_{\text{TC} > 1000} \\
\text{Year, FkPlanId, PlanName} & \gamma & \text{SUM}(C) \ AS \ TC \\
\sigma_{\text{Year} \leq 2005} \\
\text{FkPlanId} = \ PkPlanId
\end{array}
\]

\[
\begin{array}{c}
\text{Month, Year, FkPlanId} & \gamma & \text{SUM}(\text{Charge}) \ AS \ C' \\
\sigma_{\text{Year} \geq 2000} \\
\text{Calls}
\end{array}
\]

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

So the rewriting of \( Q \) succeeds:

\[
\begin{aligned}
Q1^* & : \text{SELECT} \quad \text{Year, PlanName, SUM(C) AS TC} \\
& \text{FROM} \quad \text{V1, CallingPlans} \\
& \text{WHERE} \quad \text{FkPlanId = PkPlanId AND Year <= 2005} \\
& \text{GROUP BY} \quad \text{Year, FkPlanId, PlanName} \\
& \text{HAVING} \quad \text{SUM(C) > 1000;}
\end{aligned}
\]
1. (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:

<table>
<thead>
<tr>
<th>SELECT</th>
<th>Fk, COUNT( ) AS Cn</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM</td>
<td>F, D</td>
</tr>
<tr>
<td>WHERE</td>
<td>Fk = Pk AND E &lt;&gt; 'd3'</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>SUM(C) &lt; 100;</td>
</tr>
<tr>
<td>HAVING</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pk</td>
<td>E</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>1</td>
<td>d1</td>
</tr>
<tr>
<td>2</td>
<td>d2</td>
</tr>
<tr>
<td>3</td>
<td>d3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>25</td>
</tr>
</tbody>
</table>

(a) (1 point) Give a logical query plan for the SQL query, the type and the value of the result.
(b) (1 point) Show the instance of a bitmap index on the attribute Fk.
(c) (1 point) Show the instance of a Bitmap Foreign Column Join Index on the attribute E.
(d) (2 points) 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. (5 points) Consider the FoodMart datawarehouse:

A customer is called a commuter if she buys at least 70% of her purchases (total store sales) in stores outside her city. Give an SQL query to find the number of commuters by customer city. The query must return 0 for cities with no commuter.
   With respect to the above business scenario, answer the following questions:
   (a) **(6 points)** Design a conceptual schema for the data mart to support the business questions. Your
       schema should at least be able to satisfy the above mentioned analysis requirements. You may moti-
       vate other suitable attributes for the dimensions. Specify the fact granularity, and for each measure,
       if it is additive, semi-additive or non-additive, and for each dimensional attribute the type of updates
       of the attribute.
   (b) **(3 points)** Give a logical data mart design which is able to deal with updates of dimensional attributes.
   (c) **(1 point)** Write an SQL query to answer one business questions of your choice and write a physical
       query plan for the query using any index of your choice.

4. **(5 points)** 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 3 views to materialize, different from $N_1$, with the greedy algorithm HRU. Determine the various
   possible results of HRU on the basis of the unknown value $X$.

5. **(5 points)** 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;

   Answer only one of the following questions:
   (a) Show if and how the GROUP BY can be pushed on the join
       \( \text{OrderLine} \Join \text{Order} \),
       and show if and how the GROUP BY can be pushed on the relation OrderLine.
   (b) 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;
1. (a) The logical query plan is:

\[
\begin{align*}
\pi_{Fk, \text{COUNT}()} & \text{ AS } Cn \\
\sigma_{\text{SUM}(C) < 100} & \\
Fk \bowtie \gamma \text{ COUNT}(), \text{SUM}(C) & \\
\sigma_{Fk = Pk} & \\
F & \sigma_{E <> 'd3'} & D
\end{align*}
\]

The type of the result is \{(Fk: int, Cn: int)\}. The value of the result is:

<table>
<thead>
<tr>
<th>Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fk</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

(b) and (c) These are the instances of the indexes are:

<table>
<thead>
<tr>
<th>Bitmap Index on Fk</th>
<th>BFCJI on E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fk</td>
<td>Bitmap</td>
</tr>
<tr>
<td>----</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>11000</td>
</tr>
<tr>
<td>2</td>
<td>00110</td>
</tr>
<tr>
<td>3</td>
<td>00001</td>
</tr>
</tbody>
</table>

(d) A measure \(M\) is semi-additive w.r.t. a dimension \(D\) when it cannot be summed up for different values of dimension \(D\). E.g., the monthly balance cannot be summed for different months, but it can be summed for different account holder at the same month. Here there are a correct and wrong query:

Correct: SELECT FkD1, SUM(M) FROM F GROUP BY FkD1;

Wrong: SELECT FkD2, SUM(M) FROM F GROUP BY FkD2;
2. WITH base AS (
    SELECT F.customer_id, C.city,
    CASE WHEN S.store_city=C.city THEN 0 ELSE 1 END AS other,
    SUM(store_sales) AS purchases
    FROM sales_fact F, customer C, store S
    WHERE F.customer_id = C.customer_id AND F.store_id = S.store_id
    GROUP BY F.customer_id, C.city, CASE WHEN S.store_city=C.city THEN 0 ELSE 1 END
  ), tmp AS (
    SELECT customer_id, city, other,
    100*SUM(purchases) OVER (PARTITION BY customer_id, other) /
    SUM(purchases) OVER (PARTITION BY customer_id) AS Pct
    FROM base
  )
SELECT city, SUM(CASE WHEN other=1 AND Pct >= 70 THEN 1 ELSE 0 END) AS N
FROM tmp
GROUP BY city;

3. (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

```sql
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
4. First, we observe that $10 \leq X \leq 75$. The application of the HRU algorithm yields:

<table>
<thead>
<tr>
<th>First Choice</th>
<th>Second Choice</th>
<th>Third Choice ($X \leq 55$)</th>
<th>Third Choice ($X &gt; 55$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_2$</td>
<td>$50 \times 4 = 200$</td>
<td>$25 \times 3 = 75$</td>
<td>$25 \times 1 = 25$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>$25 \times 4 = 100$</td>
<td>$30 \times 2 = 60$</td>
<td>$30 \times 2 = 60$</td>
</tr>
<tr>
<td>$N_4$</td>
<td>$80 \times 2 = 160$</td>
<td>$20 \times 1 = 20$</td>
<td>$20 \times 1 = 20$</td>
</tr>
<tr>
<td>$N_5$</td>
<td>$70 \times 1 = 70$</td>
<td>$(100 - X) \times 2$</td>
<td>$(100 - X) \times 1$</td>
</tr>
<tr>
<td>$N_6$</td>
<td>$(100 - X) \times 2$</td>
<td>$49 \times 1 = 49$</td>
<td>$49 \times 1 = 49$</td>
</tr>
<tr>
<td>$N_7$</td>
<td>$99 \times 1 = 99$</td>
<td>$10 \times 1 = 10$</td>
<td>$X - 10$</td>
</tr>
<tr>
<td>$N_8$</td>
<td>$90 \times 1 = 90$</td>
<td>$90 \times 1 = 90$</td>
<td>$X - 10$</td>
</tr>
</tbody>
</table>

Observe that:
- At the first choice, since $10 \leq X$ then $(100 - X) \times 2 \leq 180$. Hence $N_2$ is better than $N_6$.
- At the second choice, $N_6$ is selected if $(100 - X) \times 2 \geq 90$, which holds if $X \leq 55$. Otherwise $N_8$ is selected.
- At the third choice with $X \leq 55$, $N_4$ is selected because $X - 10 \leq 45$.
- At the third choice with $X > 55$, $N_4$ is selected because $(100 - X) < 45$.

5. (a - first part) Show if and how the **GROUP BY** can be pushed on the join

(OrderLine $\bowtie$ FKOrder = PKOrder Order).

The group-by can be pushed on the join (OrderLine $\bowtie$ 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 6: Logical query plan

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

(a - second part) 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

(b) 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
```
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 \( \text{SUM} \) and \( \text{COUNT} \).

\[
\begin{align*}
\pi_{\text{CCity}, \text{SUM(Revenue)} \ AS \ SR, \ \text{COUNT(\ast)} \ AS \ C} \\
\sigma_{\text{SR} > 1000} \\
\text{CCity} \\
\gamma \text{SUM(Revenue)} \ AS \ SR, \ \text{COUNT(\ast)} \ AS \ C \\
FKCustomer = PKCustomer \\
FKOrder = PKOrder
\end{align*}
\]

Figure 10: Query and view logical query plans

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

\[
\begin{align*}
\pi_{\text{CCity}, \text{SUM(Revenue)} \ AS \ TR, \ \text{COUNT(\ast)} \ AS \ Cnt} \\
\sigma_{\text{SR} > 1000} \\
\text{CCity} \\
\gamma \text{SUM(Revenue)} \ AS \ TR, \ \text{COUNT(\ast)} \ AS \ Cnt \\
FKCustomer = PKCustomer \\
FKOrder = PKOrder
\end{align*}
\]

Figure 11: Join compensation
To match the groupings, the compensation on the operand of $\gamma_V$ floats, and since $g(Q) \rightarrow g(V) \land g(V) \rightarrow g(Q)$, the rewriting does not require a grouping compensation, but an aggregate compensation only with a project, as shown in the figure:

The other compensations required for the $\sigma$ and $\pi^b$ in $Q$ are shown in the figure:

Since the internal $\pi^b$ of the compensation is useless, the final solution is the following:
Rewriting of $Q$:

$$\begin{align*}
\text{QR: } & \text{SELECT } \text{CCity, TR/Cnt AS avgR} \\
& \text{FROM } \text{V, Customer} \\
& \text{WHERE } \text{FKCustomer} = \text{PKCustomer AND TR} > 1000;
\end{align*}$$
Let us use the approach with a *transformation of the logical query plan*.

The rewriting of $Q$ with the $\gamma$ pushed below the first join

\[
(\text{OrderLine} \searrow \text{FKOrder} = \text{PKOrder} \nearrow \text{Order})
\]

produces a logical tree that, with the rewriting of $\text{AVG}$ and the changing of $\text{SR}$ and $\text{C}$ in the $\gamma$ with $\text{TR}$ and $\text{Cnt}$, has as subtree the tree of the view.

\[
\pi_{\text{CCity, TR/Cnt AS avgR}} \sigma_{\text{SR} > 1000} \bigtriangleup \text{FKCustomer} = \text{PKCustomer} \bigtriangledown \text{FKCustomer} \gamma \text{SUM(Revenue)} \ AS \text{TR} \comma \text{COUNT(\ast)} \ AS \text{Cnt} \bigtriangledown \text{FKOrder} = \text{PKOrder} \bigtriangledown \text{OrderLine} \searrow \text{Order}
\]

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

Rewriting of $Q$:

QR:  
\[
\text{SELECT CCity, TR/Cnt AS avgR}
\comma
\text{FROM V, Customer}
\comma
\text{WHERE FKCustomer = PKCustomer AND TR > 1000;}
\]