Exam of *Decision Support Databases: Example 1*

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

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

(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

\begin{itemize}
\item Customer(PkCustPhoneNo, CustName, CustCity)
\item CallingPlans(PkPlanId, PlanName)
\item Calls(PkCustPhoneNo, FkPlanId, Day, Month, Year, Duration, Charge)
\end{itemize}

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

\begin{verbatim}
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;
\end{verbatim}

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

(b) \textbf{(5 points)} Show if and how the query can be rewritten using the materialized view

\begin{verbatim}
V1: SELECT FkPlanId, Month, Year, SUM(Charge) AS C
    FROM Calls
    WHERE Year >= 2000
    GROUP BY FkPlanId, Month, Year;
\end{verbatim}
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:

<table>
<thead>
<tr>
<th></th>
<th>First Choice</th>
<th>Second Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_2$</td>
<td>$50 \times 5 = 250$</td>
<td></td>
</tr>
<tr>
<td>$N_3$</td>
<td>$25 \times 5 = 125$</td>
<td>$25 \times 2 = 50$</td>
</tr>
<tr>
<td>$N_4$</td>
<td>$80 \times 2 = 160$</td>
<td>$30 \times 2 = 60$</td>
</tr>
<tr>
<td>$N_5$</td>
<td>$70 \times 3 = 210$</td>
<td>$20 \times 3 = 60$</td>
</tr>
<tr>
<td>$N_6$</td>
<td>$60 \times 2 = 120$</td>
<td>$60 + 10 = 70$</td>
</tr>
<tr>
<td>$N_7$</td>
<td>$99 \times 1 = 99$</td>
<td>$49 \times 1 = 49$</td>
</tr>
<tr>
<td>$N_8$</td>
<td>$90 \times 1 = 90$</td>
<td>$40 \times 1 = 40$</td>
</tr>
</tbody>
</table>

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

   Customer(PkCustPhoneNo, CustName, CustCity)
   CallingPlans(PkPlanId, PlanName)
   Calls(PkCustPhoneNo, PkPlanId, 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.
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

\[(Year, PlanName)^+ = \{Year, PlanName, PkPlanId, FkPlanId\}\]

Figure 1: Logical query plan new

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;

FkPlanId, Month, Year \( \gamma \) SUM(Charge) AS C
\[ \sigma \text{Year} \geq 2000 \] Calls

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 \( V \):

\( \pi \text{Year, PlanName, TC} \)
\[ \sigma \text{TC} > 1000 \]
\[ \bowtie \exists \text{FkPlanId = PkPlanId} \]

Year, FkPlanId \( \gamma \) SUM(C) AS TC CallingPlans
\[ \sigma \text{Year} \leq 2005 \]
Month, Year, FkPlanId \( \gamma \) SUM(Charge) AS C
\[ \sigma \text{Year} \geq 2000 \]
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.

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

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

So the rewriting of \( Q \) succeeds:

\[
\begin{align*}
\text{Q1}: \quad \text{SELECT} \quad \text{Year, PlanName, SUM(C) AS TC} \\
\text{FROM} \quad \text{V1, CallingPlans} \\
\text{WHERE} \quad \text{FkPlanId} = \text{PkPlanId AND Year } \leq 2005 \\
\text{GROUP BY} \quad \text{Year, FkPlanId, PlanName} \\
\text{HAVING} \quad \text{SUM(C) } > 1000; \\
\end{align*}
\]
Exam of *Decision Support Databases: Example 2*

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

\[
\begin{align*}
\text{SELECT} & \quad Fk, \text{COUNT}(\ast) \text{ AS } Cn \\
\text{FROM} & \quad F, D \\
\text{WHERE} & \quad Fk = Pk \text{ AND } E \neq 'd3' \\
\text{GROUP BY} & \quad Fk \\
\text{HAVING} & \quad \text{SUM}(C) < 100;
\end{align*}
\]

(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 \( \text{SUM}(M) \).

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 ⊘◁FKOrder = PKOrder 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

   ```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

   ```plaintext
   Project ([Name, SUM(NOR)/(SUM(NOR)+SUM(NVR)+SUM(NUR)) AS OR])
   HashGroupBy ([HotelFK, Name], {SUM(NOR), SUM(NVR), SUM(NUR)})
   IndexNestedLoop (HotelFK=HotelPK)
   TableAccess (F)
   IndexFilter (Hotel, IdxPkH, HotelPK=HotelFK)
   RIDFromBM
   BMAnd
   BMFCJIndexFilter (IdxHF, City = 'Florence')
   BMIndexFilter (IdxDF, DateFK = 20100717)
   ```
3. See lecture notes.

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

\[
\begin{align*}
\text{Customer}(&PKCustomer, CName, CCity) \\
\text{Order}(&PKOrder, FKCustomer, ODate) \\
\text{Product}(&PKProduct, PName, PCost) \\
\text{OrderLine}(&\text{LineNo}, FKOrder, FKProduct, Quantity, ExtendedPrice, Discount, Revenue)
\end{align*}
\]

and the query

\[
Q: \quad \text{SELECT} \quad \text{CCity, AVG(Revenue) AS avgR} \\
\text{FROM} \quad \text{OrderLine, Order, Customer} \\
\text{WHERE} \quad \text{FKOrder = PKOrder AND FKCustomer = PKCustomer} \\
\text{GROUP BY} \quad \text{CCity, FKCustomer} \\
\text{HAVING} \quad \text{SUM(Revenue) > 1000;}
\]

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

\[
\begin{align*}
&\pi^b_{\text{CCity, avgR}} \\
&\sigma_{SR > 1000} \\
&\text{CCity, FKCustomer} \bowtie \text{SUM(Revenue) AS SR, AVG(Revenue) AS avgR} \\
&\text{FKCustomer = PKCustomer} \\
&\text{OrderLine} \bowtie \text{FKOrder = PKOrder Customer} \\
&\text{Order}
\end{align*}
\]

Figure 6: Logical query plan

The group-by can be pushed on the join (OrderLine $\bowtie$FKOrder = PKOrder Order) because the \textit{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`
(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.
The join operations do not match. Therefore, a compensation is added as shown in the figure:

```
πᵇ_{CCity, SR/C} AS avgR
σ_{SR > 1000}
CCity ⊘[πᵇ_{SUM(Revenue) AS SR, COUNT(*) AS C}]
FKCustomer ⊘[FKCustomer = PKCustomer]
FKOrder = PKOrder
OrderLine ⊘ Order
(a) A_Q
```

```
FKCustomer ⊘[FKCustomer ⊘[SUM(Revenue) AS SR, COUNT(*) AS C]]
FKOrder = PKOrder
OrderLine ⊘ Order
(b) A_V
```

Figure 10: Query and view logical query plans

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:

![Diagram of compensation](image1)

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

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

![Diagram of other compensations](image2)

Figure 13: Other compensations

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

QR: \[
\text{SELECT } \text{CCity, TR/Cnt AS avgR,}
\text{FROM } V, \text{Customer}
\text{WHERE FKCustomer = PKCustomer AND TR > 1000;}
\]
Let us use the approach with a transformation of the logical query plan. The rewriting of $Q$ with the $\gamma$ pushed below the first join

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

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

Rewriting of $Q$:

$$\begin{align*}
QR: & \quad \text{SELECT} \quad \text{CCity, TR/Cnt AS } \text{avgR} \\
   & \quad \text{FROM} \quad \text{V, Customer} \\
   & \quad \text{WHERE} \quad \text{FKCustomer} = \text{PKCustomer} \quad \text{AND} \quad \text{TR} > 1000;
\end{align*}$$

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