Analysis Process Modeling Notation For Business Intelligence (APMN4BI)

Thomas Neuböck

October 2017

Contents

Chapter 1

Preliminaries for APMN4BI

1.1 Running Example

see Figure 1.1

1.2 Enriched Dimensional Fact Model (eDFM)

see Figure 1.2, 1.3, 1.4

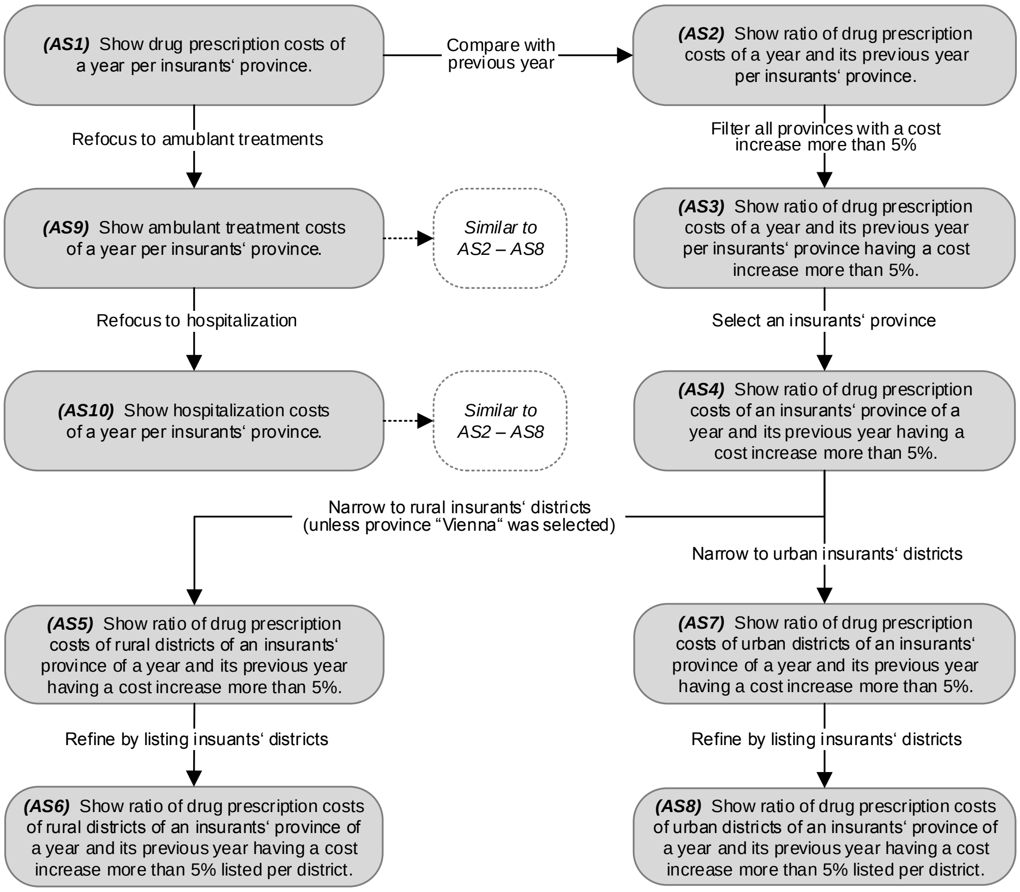


Figure 1.1: Health care use case – overview of an analysis process

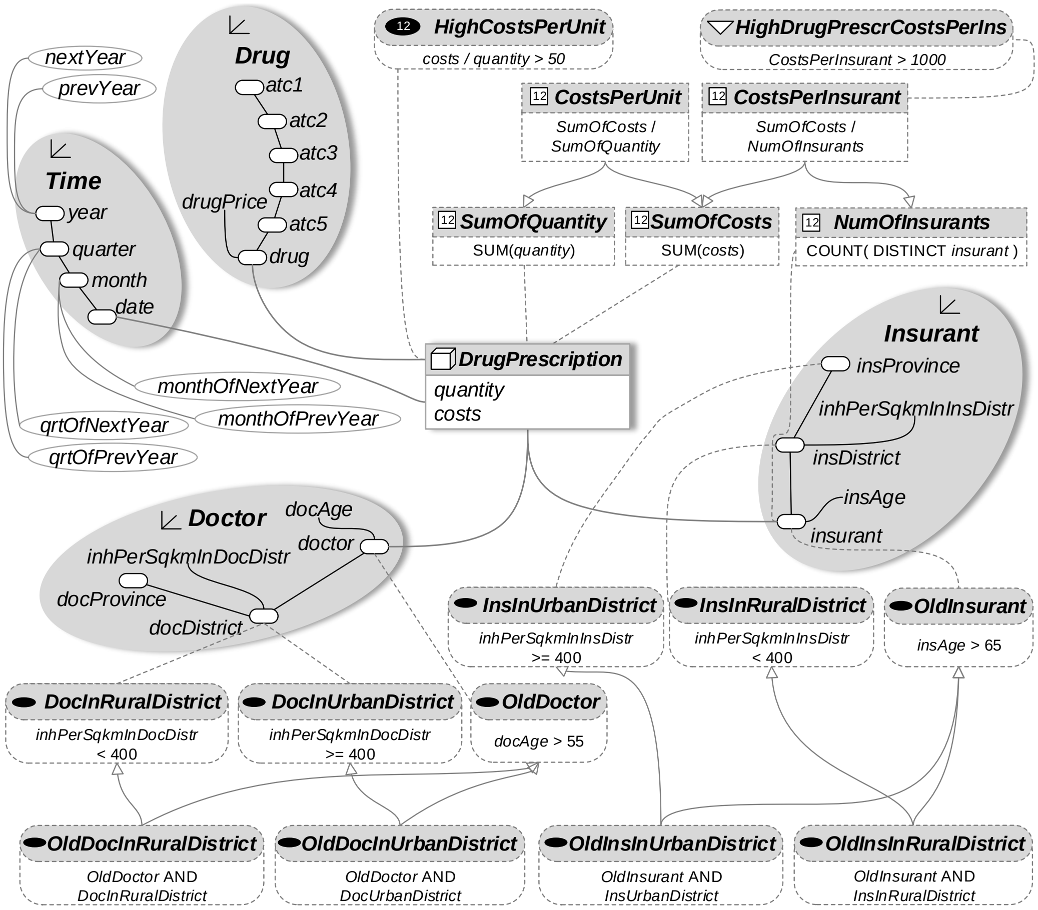


Figure 1.2: eDFM for Drug Prescriptions

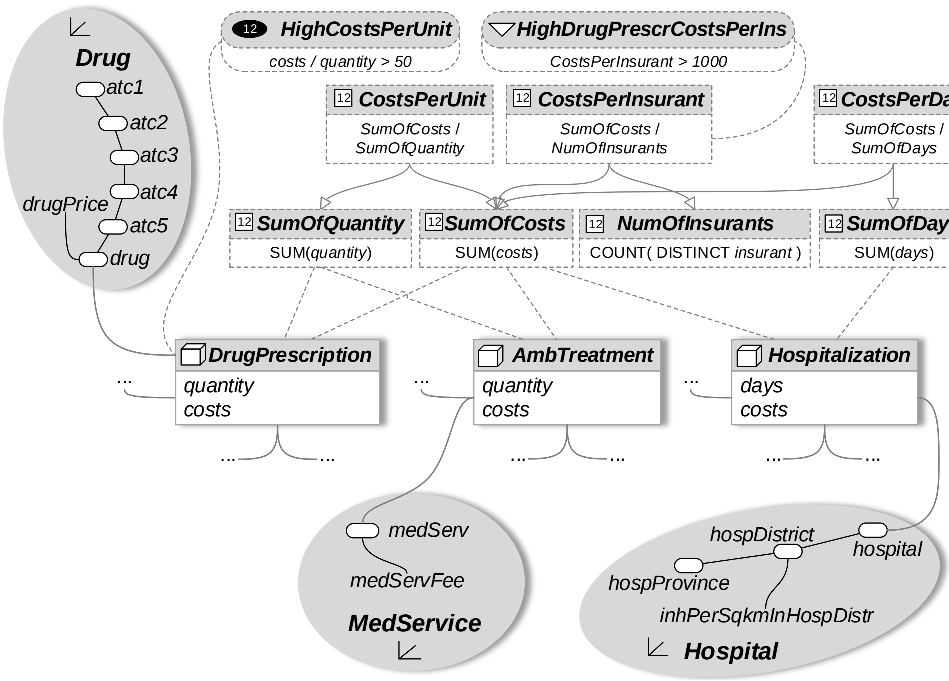
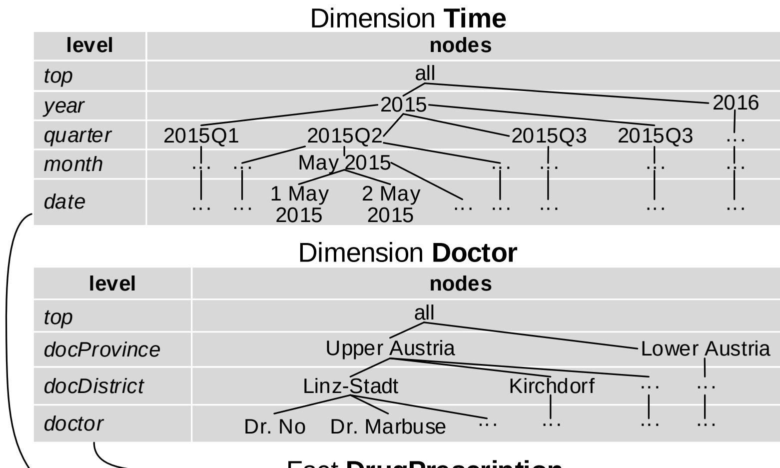


Figure 1.3: eDFM for Ambulant Treatment and Hospitalization



Fact **DrugPrescription**

date	*doctor*	*insurant*	*drug*	*quantity*	*costs*
1 May 2015	Dr. No	...	...	1	100
1 May 2015	Dr. Marbuse	...	...	2	50
2 May 2015	Dr. Marbuse	...	...	1	10
2 May 2015	Dr. No	...	...	3	5
2 May 2015	Dr. Marbuse	...	...	1	15
...	...	...	...	...	...

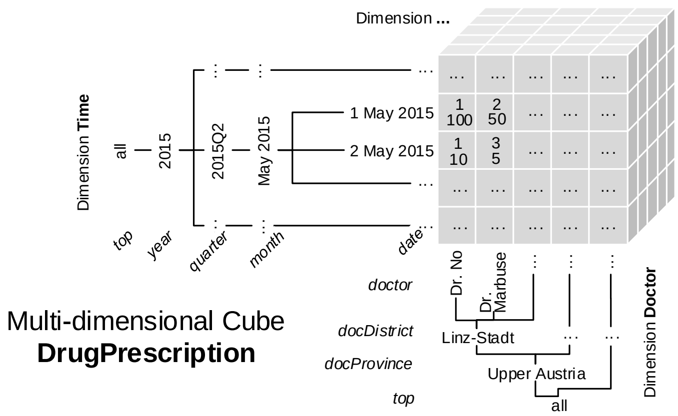


Figure 1.4: Cube Instance

Chapter 2

Analysis Situations

2.1 Non-comparative Analysis Situations

see Figure 2.1, 2.2, 2.3, 2.4, 2.5

2.2 Comparative Analysis Situations

see Figure 2.6, 2.7, 2.8, 2.9, 2.10

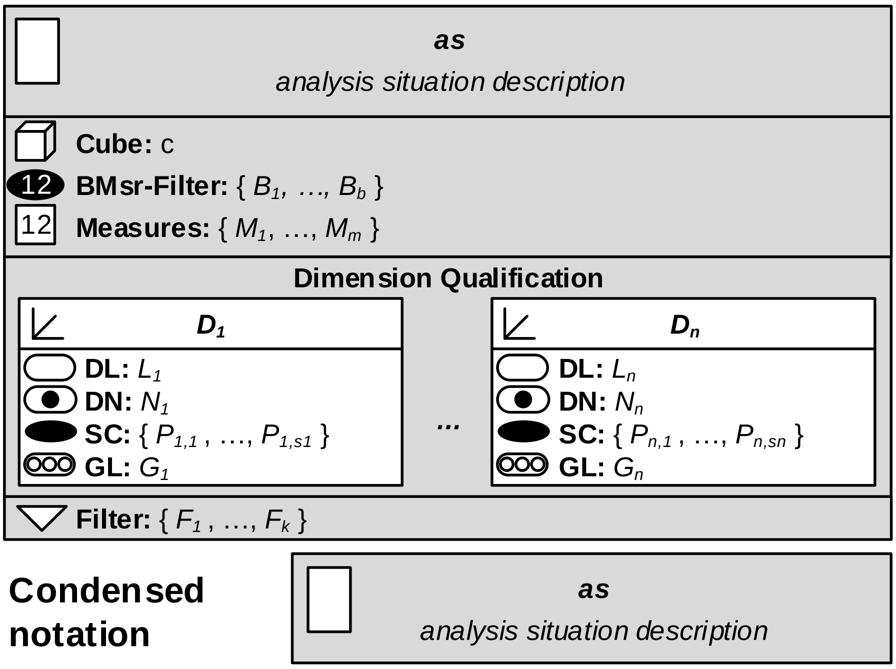


Figure 2.1: Graphical Template of a Non-comparative Analysis Situation

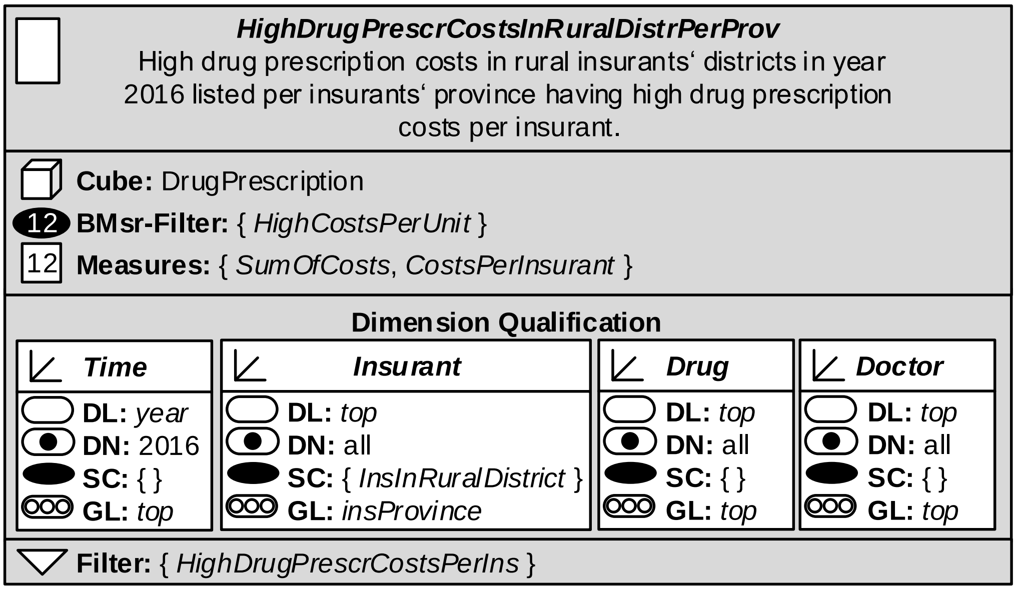


Figure 2.2: Example of a Graphical Representation of a Non-comparative Analysis Situation

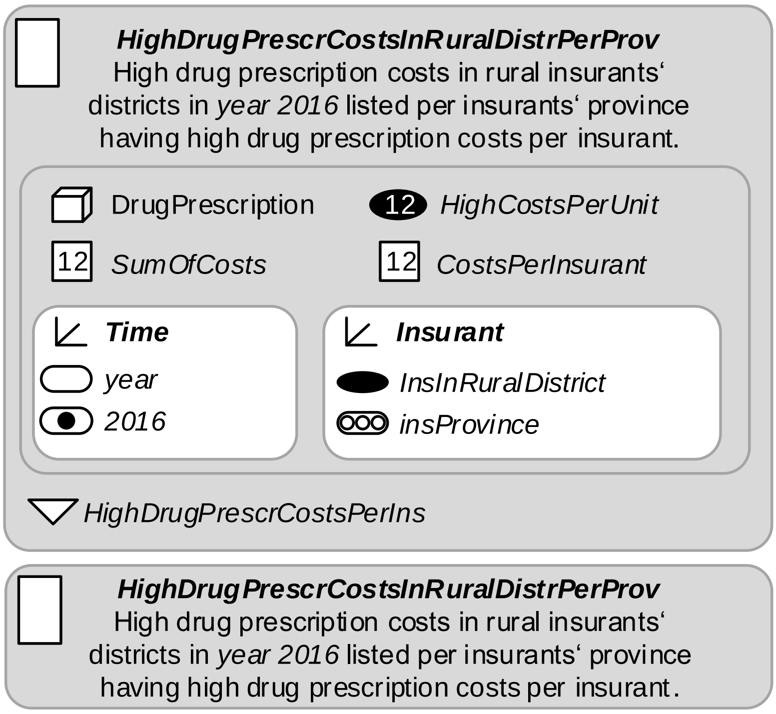


Figure 2.3: Example of an Alternative Graphical Representation of a Non-comparative Analysis Situation

SQL translation:

select	*G_1, …, G_n*,	
	M_1, …, M_m,	
from	c natural join D_1 natural join … natural join D_n	
where	B_1 and … and B_b	and
	$L_1 = N_1$ and … and $L_n = N_n$	and
	$P_{1,1}$ and … and $P_{1,s1}$	and
	…	and
	$P_{n,1}$ and … and $P_{n,sn}$	
group by	*G_1, …, G_n*	
having	F_1 and … and F_k	

Result set:

G_1	…	G_n	M_1	…	M_m
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…

Figure 2.4: SQL Translation

SQL translation:

select	*insProvince*,		
	SUM(*costs*)		as *SumOfCosts*,
	SUM(*costs*) / COUNT(DISTINCT *insurant*)		as *CostsPerInsurant*
from	DrugPrescription	natural join	
	Time	natural join	
	Insurant		
where	costs / quantity > 50 and		
	year = 2015 and		
	inhPerSqkmInInsDistr < 400		
group by	*insProvince*		
having	SUM(*costs*) / COUNT(DISTINCT *insurant*) > 1000		

Example result set:

insProvince	*SumOfCosts*	*CostsPerInsurant*
Upper Austria	117790612.01	2120.31
Lower Austria	197790612.45	2415.07
...	...	...

Figure 2.5: Example of an SQL Translation

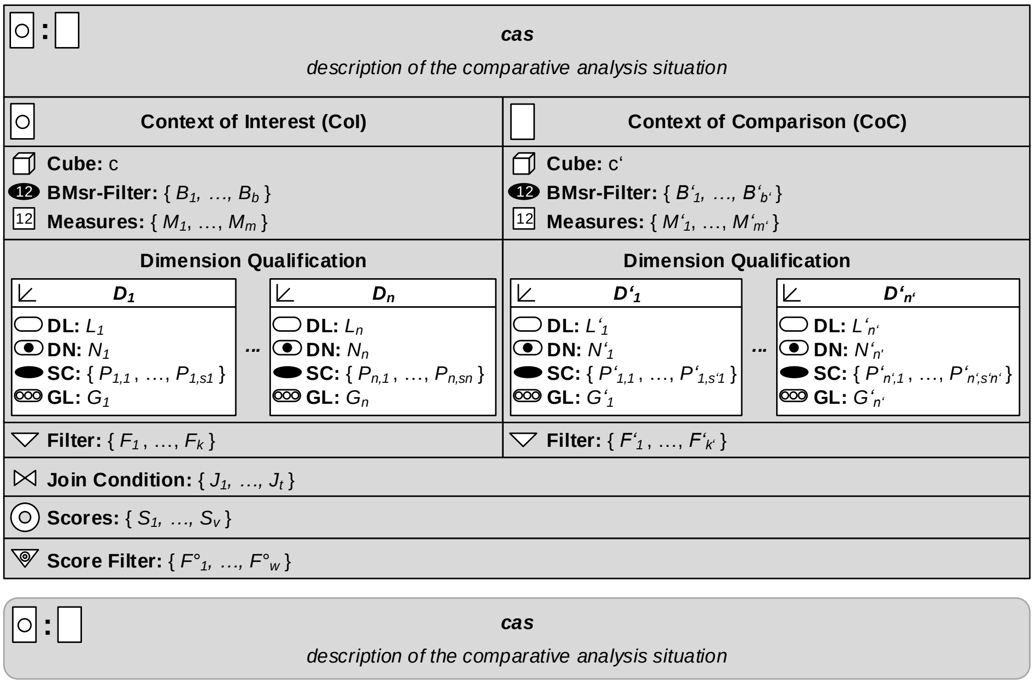


Figure 2.6: Graphical Template of a Comparative Analysis Situation

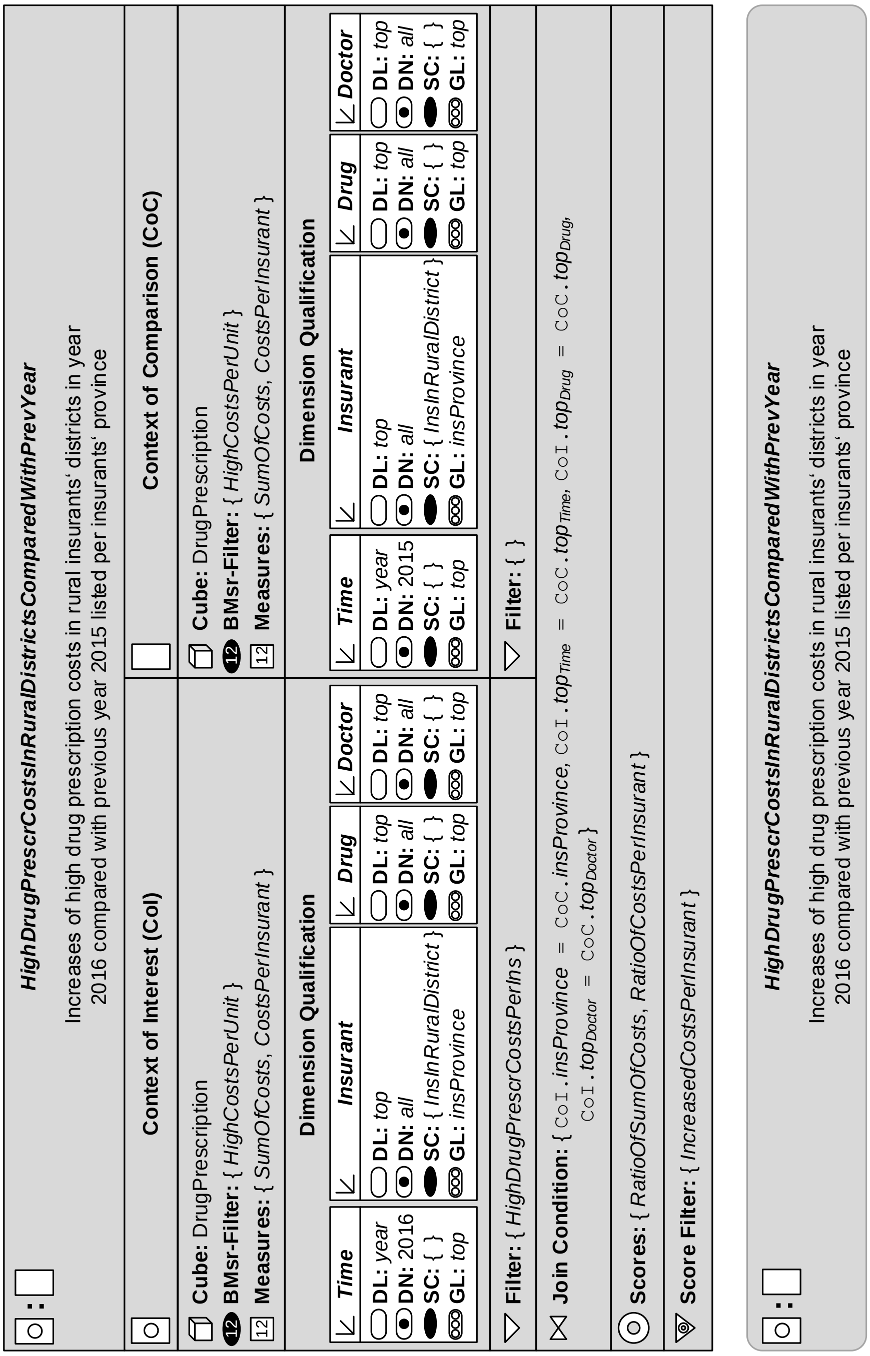


Figure 2.7: Example of a Graphical Representation of a Comparative Analysis Situation

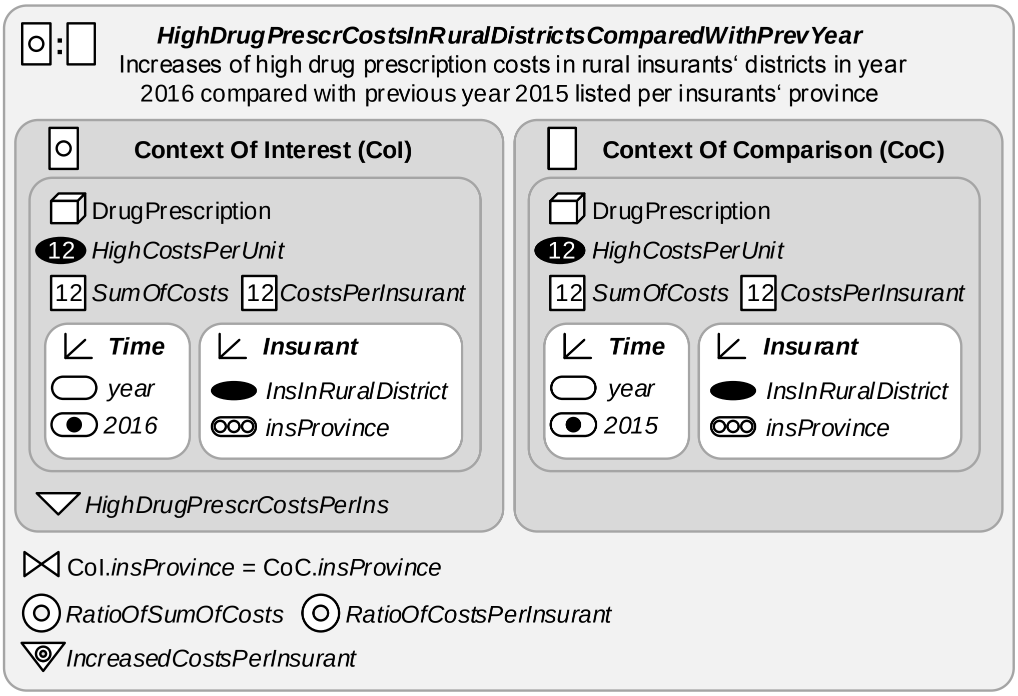


Figure 2.8: Example of an Alternative Graphical Representation of a Comparative Analysis Situation

SQL translation:

select Col.G_1, …, Col.G_n, Col.M_1, …, Col.M_m,
CoC.G'_1, …, CoC.$G'_{n'}$, CoC.M'_1, …, CoC.$M'_{m'}$,
S_1, …, S_v

from (select G_1, …, G_n, M_1, …, M_m
from c natural join D_1 natural join … natural join D_n
where B_1 and … and B_b and
$L_1 = N_1$ and … and $L_n = N_n$ and
$P_{1,1}$ and … and $P_{1,s1}$ and
… and
$P_{n,1}$ and … and $P_{n,sn}$
group by G_1, …, G_n
having F_1 and … and F_k
) Col

inner join (select G'_1, …, $G'_{n'}$, M'_1, …, $M'_{m'}$
from c' natural join D'_1 natural join … natural join D'_n
where B'_1 and … and $B'_{b'}$ and
$L'_1 = N'_1$ and … and $L'_{n'} = N'_{n'}$ and
$P'_{1,1}$ and … and $P'_{1,s'1}$ and
… and
$P'_{n',1}$ and … and $P'_{n',s'n'}$
group by G'_1, …, $G'_{n'}$
having F'_1 and … and $F'_{k'}$
) CoC

on J_1 and … and J_t

where $F°_1$ and … and $F°_w$

Result set:

Col.G_1	…	Col.G_n	Col.M_1	…	Col.M_n	CoC.G'_1	…	CoC.$G'_{n'}$	CoC.M'_1	…	CoC.$M'_{n'}$	S_1	…	S_t
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Figure 2.9: SQL Translation of a Comparative Analysis Situation

SQL translation:

```
select      CoI.insProvince, CoI.SumOfCosts, CoI.CostsPerInsurant,
            CoC.insProvince, CoC.SumOfCosts, CoC.CostsPerInsurant,
            CoI.SumOfCosts / CoC.SumOfCosts              as RatioOfSumOfCosts,
            CoI.CostsPerInsurant / CoC.CostsPerInsurant   as RatioOfCostsPerInsurant

from        (select     insProvince, SUM(costs) as SumOfCosts, SUM(costs) / COUNT( DISTINCT insurant ) as CostsPerInsurant
            from        DrugPrescription natural join Time natural join Insurant
            where       costs / quantity > 50 and
                        year = 2016 and
                        inhPerSqkmInInsDistr < 400
            group by    insProvince
            having      SUM(costs) / COUNT( DISTINCT insurant ) > 1000
            ) CoI

inner join  (select     insProvince, SUM(costs) as SumOfCosts, SUM(costs) / COUNT( DISTINCT insurant ) as CostsPerInsurant
            from        DrugPrescription natural join Time natural join Insurant
            where       costs / quantity > 50 and
                        year = 2015 and
                        inhPerSqkmInInsDistr < 400
            group by    insProvince
            ) CoC

on          CoI.insProvince = CoC.insProvince

where       CoI.CostsPerInsurant / CoC.CostsPerInsurant > 1
```

Example result set:

CoI. *insProvince*	CoI. *SumOfCosts*	CoI. *CostsPerInsurant*	CoC. *insProvince*	CoC. *SumOfCosts*	CoC. *CostsPerInsurant*	*RatioOfSumOf Costs*	*RatioOfCostsPer Insurant*
Upper Austria	117790612.01	2120.31	Upper Austria	107790612.01	1959.21	1,0928	1.0822
Lower Austria	197790612.45	2415.07	Lower Austria	187790612.45	2295.74	1.0533	1.0520
...	...	...	...	...	...	...	...

Figure 2.10: Example of an SQL Translation of a Comparative Analysis Situation

Chapter 3

Navigation Operators

3.1 Navigation Step

see Figure 3.1

3.2 Operators Not Involving Comparison

3.2.1 Operators Changing Granularity Level

see Table 3.1

3.2.2 Operators Changing Dice Node

see Table 3.2, 3.3

3.2.3 Operators Changing Slice Conditions

see Table 3.4

3.2.4 Operators Changing Base Measure Conditions

see Table 3.5

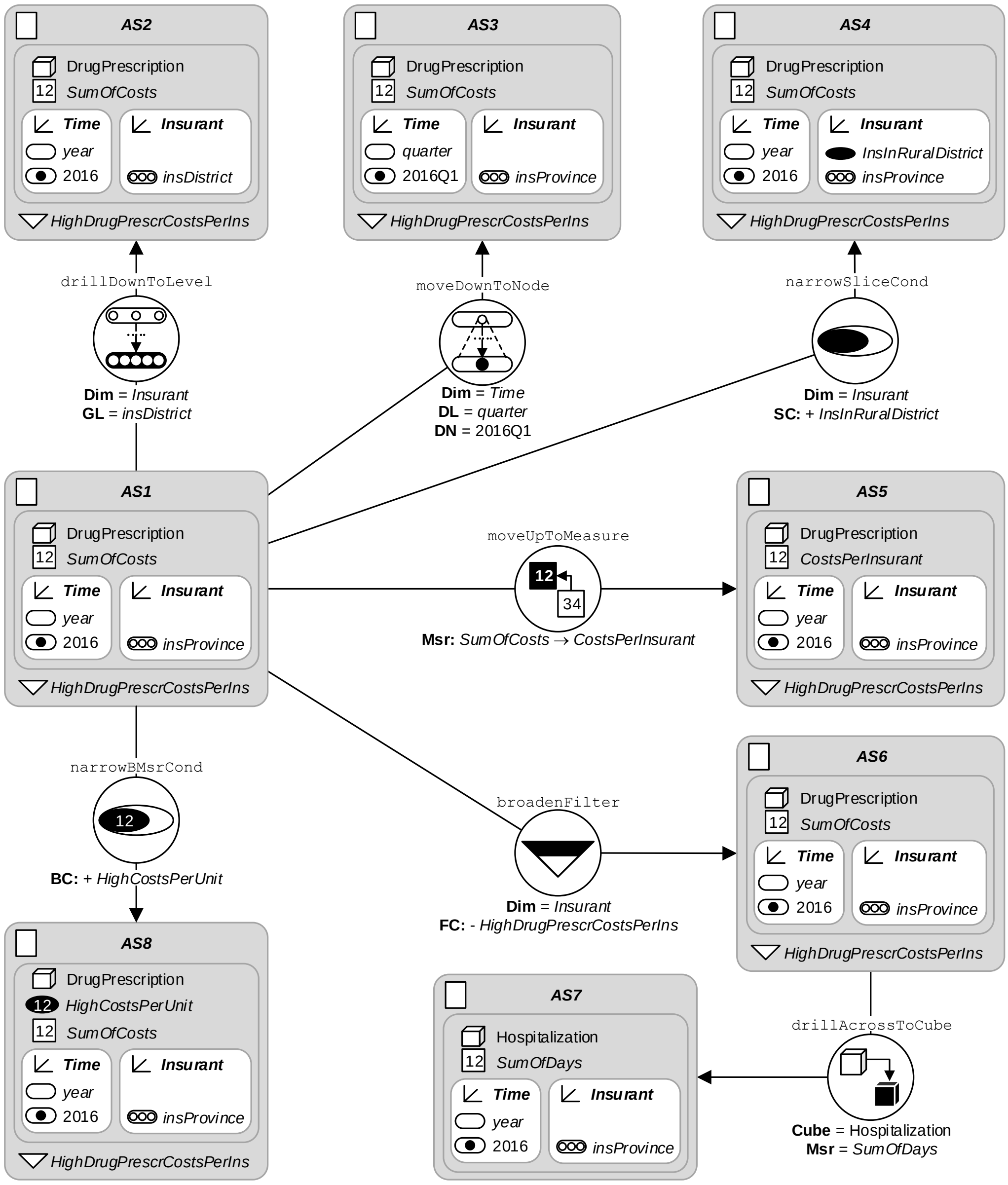


Figure 3.1: Navigation operator examples

Table 3.1: Operators `drillDownOneLevel`, `drillDownToLevel`, `rollUpOne-Level`, and `rollUpToLevel`

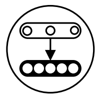
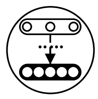
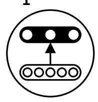
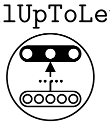
Operator Definition	**Symbol**
`drillDownOneLevel(`D`)` **Precondition:** $GranLvl_{src}(D) \neq base_D$ **Postcondition:** $GranLvl_{trg}(D) \rightarrow GranLvl_{src}(D)$	`drillDownOneLevel`  **Dim** $= D$
`drillDownToLevel(`D`,`G`)` **Precondition:** $G \twoheadrightarrow GranLvl_{src}(D)$ **Postcondition:** $GranLvl_{trg}(D) = G$	`drillDownToLevel`  **Dim** $= D$ **GL** $= G$
`rollUpOneLevel(`D`)` **Precondition:** $GranLvl_{src}(D) \neq top_D$ **Postcondition:** $GranLvl_{src}(D) \rightarrow GranLvl_{src}(D)$	`rollUpOneLevel`  **Dim** $= D$
`rollUpToLevel(`D`,`G`)` **Precondition:** $GranLvl_{src}(D) \twoheadrightarrow G$ **Postcondition:** $GranLvl_{trg}(D) = G$	`rollUpToLevel`  **Dim** $= D$ **GL** $= G$

Table 3.2: Operators `moveDownToNode`, `moveUpToNode`, `moveAsideToNode`, and `moveToNode`

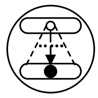
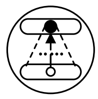
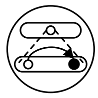
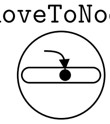
Operator Definition	**Symbol**
`moveDownToNode(`D`,`L`,`N`)` **Precondition:** $L \in Lvls_D$, $L \neq base_D$, $L \twoheadrightarrow DiceLvl_{src}(D)$, $N \in Nodes_d(L)$ with $d = DimInstance_{src}(D)$, and $N \twoheadrightarrow DiceNode_{src}(D)$ **Postcondition:** $L = DiceLvl_{trg}(D)$, $N = DiceNode_{trg}(D)$	`moveDownToNode`  **Dim** $= D$ **DL** $= L$ **DN** $= N$
`moveUpToNode(`D`,`L`)` **Precondition:** $L \in Lvls_D$, $L \neq top_D$, $DiceLvl_{src}(D) \twoheadrightarrow L$, $DiceNode_{src}(D) \twoheadrightarrow N$ for the unique existing $N \in Nodes_d(L)$, with $d = DimInstance_{src}(D)$ **Postcondition:** $L = DiceLvl_{trg}(D)$, $N = DiceNode_{trg}(D)$	`moveUpToNode`  **Dim** $= D$ **DL** $= L$
`moveAsideToNode(`D`,`N`)` **Precondition:** $DiceLvl_{src}(D) \neq top_D$, $N \neq DiceNode_{src}(D)$ and, for the unique existing $\hat{N}$ with $DiceNode_{src}(D) \rightarrow \hat{N}$: $N \rightarrow \hat{N}$ **Postcondition:** $DiceNode_{trg}(D) = N$	`moveAsideToNode`  **Dim** $= D$ **DN** $= N$
`moveToNode(`D`,`L`,`N`)` **Precondition:** $L \neq DiceLvl_{src}(D)$, $L \in Lvls_D$, and $N \in Nodes_d(L)$, with $d = DimInstance_{src}(D)$ **Postcondition:** $DiceLvl_{trg}(D) = L$, $DiceNode_{trg}(D) = N$	`moveToNode`  **Dim** $= D$ **DL** $= L$ **DN** $= N$

Table 3.3: Operators `moveDownToFirstNode`, `moveDownToLastNode`, `moveToNextNode`, and `moveToPrevNode`

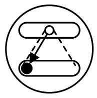
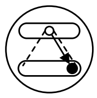
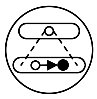
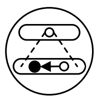
Operator Definition	**Symbol**
`moveDownToFirstNode(`D`)` **Precondition:** $DiceLvl_{src}(D) \neq base_D$ **Postcondition:** $DiceLvl_{trg}(D) \to DiceLvl_{src}(D)$, $DiceNode_{trg}(D) = min_{\prec}\{N \mid N \to DiceNode_{src}(D)\}$	`moveDownToFirstNode`  **Dim** $= D$
`moveDownToLastNode(`D`)` **Precondition:** $DiceLvl_{src}(D) \neq base_D$ **Postcondition:** $DiceLvl_{trg}(D) \to DiceLvl_{src}(D)$, $DiceNode_{trg}(D) = max_{\prec}\{N \mid N \to DiceNode_{src}(D)\}$	`moveDownToLastNode`  **Dim** $= D$
`moveToNextNode(`D`)` **Precondition:** $DiceLvl_{src}(D) \neq top_D$ and, for the unique existing N with $DiceNode_{src}(D) \to N$, there exits an $N' \to N$, such that $N' = next_{\prec}(DiceNode_{src}(D))$ **Postcondition:** $DiceNode_{trg}(D) = N'$	`moveToNextNode`  **Dim** $= D$
`moveToPrevNode(`D`)` **Precondition:** $DiceLvl_{src}(D) \neq top_D$ and, for the unique existing N with $DiceNode_{src}(D) \to N$, there exits an $N' \to N$, such that $N' = prev_{\prec}(DiceNode_{src}(D))$ **Postcondition:** $DiceNode_{trg}(D) = N'$	`moveToPrevNode`  **Dim** $= D$

Table 3.4: Operators `narrowSliceCond`, `broadenSliceCond`, and `refocus-SliceCond`

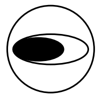
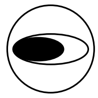
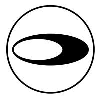
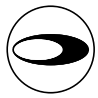
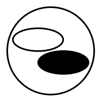
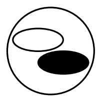
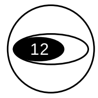
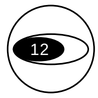
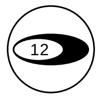
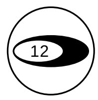
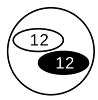
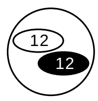
Operator Definition	Symbol
`narrowSliceCond(`D`,+`P_1`,`$\cdots$`,+`P_s`)` **Precondition:** $P_1, \cdots, P_s \in \mathit{DimPredicates}_D$, $\{P_1, \cdots, P_s\} \nsubseteq \mathit{SliceConds}_{src}(D)$ **Postcondition:** $\mathit{SliceConds}_{trg}(D) =$ $\mathit{SliceConds}_{src}(D) \cup \{P_1, \cdots, P_s\}$	`narrowSliceCond`  **Dim** $= D$ **SC**: $+P_1, \cdots, +P_s$
`narrowSliceCond(`D`,`P_{old} `->` P_{new}`)` **Precondition:** $P_{old}, P_{new} \in \mathit{DimPredicates}_D$, $P_{old} \in \mathit{SliceConds}_{src}(D)$, $P_{new} \notin \mathit{SliceConds}_{src}(D)$, $P_{new} \Rightarrow P_{old}$ **Postcondition:** $\mathit{SliceConds}_{trg}(D) =$ $\mathit{SliceConds}_{src}(D) \backslash \{P_{old}\} \cup \{P_{new}\}$	`narrowSliceCond`  **Dim** $= D$ **SC**: $P_{old} \rightarrow P_{new}$
`broadenSliceCond(`D`,-`P_1`,`$\cdots$`,-`P_s`)` **Precondition:** $P_1, \cdots, P_s \in \mathit{DimPredicates}_D$, $\{P_1, \cdots, P_s\} \nsubseteq \mathit{SliceConds}_{src}(D)$ **Postcondition:** $\mathit{SliceConds}_{trg}(D) =$ $\mathit{SliceConds}_{src}(D) \backslash \{P_1, \cdots, P_s\}$	`broadenSliceCond`  **Dim** $= D$ **SC**: $-P_1, \cdots, -P_s$
`broadenSliceCond(`D`,`P_{old} `->` P_{new}`)` **Precondition:** $P_{old}, P_{new} \in \mathit{DimPredicates}_D$, $P_{old} \in \mathit{SliceConds}_{src}(D)$, $P_{new} \notin \mathit{SliceConds}_{src}(D)$, $P_{old} \Rightarrow P_{new}$ **Postcondition:** $\mathit{SliceConds}_{trg}(D) =$ $\mathit{SliceConds}_{src}(D) \backslash \{P_{old}\} \cup \{P_{new}\}$	`broadenSliceCond`  **Dim** $= D$ **SC**: $P_{old} \rightarrow P_{new}$
`refocusSliceCond(`D`,`P_1`,`$\cdots$`,`P_s`)` **Precondition:** $P_1, \cdots, P_s \in \mathit{DimPredicates}_D$, $\{P_1, \cdots, P_s\} \neq \mathit{SliceConds}_{src}(D)$ **Postcondition:** $\mathit{SliceConds}_{trg}(D) =$ $\{P_1, \cdots, P_s\}$	`refocusSliceCond`  **Dim** $= D$ **SC**: $P_1, \cdots, P_s$
`refocusSliceCond(`D`,`P_{old} `->` P_{new}`)` **Precondition:** $P_{old}, P_{new} \in \mathit{DimPredicates}_D$, $P_{old} \in \mathit{SliceConds}_{src}(D)$, $P_{new} \notin \mathit{SliceConds}_{src}(D)$ **Postcondition:** $\mathit{SliceConds}_{trg}(D) =$ $\mathit{SliceConds}_{src}(D) \backslash \{P_{old}\} \cup \{P_{new}\}$	`refocusSliceCond`  **Dim** $= D$ **SC**: $P_{old} \rightarrow P_{new}$

Table 3.5: Operators `narrowBMsrCond`, `broadenBMsrCond`, and `refocus-BMsrCond`

Operator Definition	**Symbol**
`narrowBMsrCond(+`B_1`,`$\cdots$`,+`B_b`)` **Precondition:** $B_1, \cdots, B_b \in BMsrPredicates_C$ with $C = CubeSchema_{src}$, $\{B_1, \cdots, B_b\} \nsubseteq BMsrConds_{src}$ **Postcondition:** $BMsrConds_{trg} = BMsrConds_{src} \cup \{B_1, \cdots, B_b\}$	`narrowBMsrCond`  **BC**: $+B_1, \cdots, +B_b$
`narrowBMsrCond(`B_{old} `->` B_{new}`)` **Precondition:** $B_{new} \in BMsrPredicates_C$ with $C = CubeSchema_{src}$, $B_{old} \in BMsrConds_{src}$, $B_{new} \notin BMsrConds_{src}$, $B_{new} \Rightarrow B_{old}$ **Postcondition:** $BMsrConds_{trg} = BMsrConds_{src} \backslash \{B_{old}\} \cup \{B_{new}\}$	`narrowBMsrCond`  **BC**: $B_{old} \rightarrow B_{new}$
`broadenBMsrCond(-`B_1`,`$\cdots$`,-`B_b`)` **Precondition:** $\{B_1, \cdots, B_b\} \subseteq BMsrConds_{src}$ **Postcondition:** $BMsrConds_{trg} = BMsrConds_{src} \backslash \{B_1, \cdots, B_b\}$	`broadenBMsrCond`  **BC**: $-B_1, \cdots, -B_b$
`broadenBMsrCond(`B_{old} `->` B_{new}`)` **Precondition:** $B_{new} \in BMsrPredicates_C$ with $C = CubeSchema_{src}$, $B_{old} \in BMsrConds_{src}$, $B_{new} \notin BMsrConds_{src}$, $B_{old} \Rightarrow B_{new}$ **Postcondition:** $BMsrConds_{trg} = BMsrConds_{src} \backslash \{B_{old}\} \cup \{B_{new}\}$	`broadenBMsrCond`  **BC**: $B_{old} \rightarrow B_{new}$
`refocusBMsrCond(`B_1`,`$\cdots$`,`B_b`)` **Precondition:** $B_1, \cdots, B_b \in BMsrPredicates_C$ with $C = CubeSchema_{src}$ $BMsrConds_{src} \neq \{B_1, \cdots, B_b\}$ **Postcondition:** $BMsrConds_{trg} = \{B_1, \cdots, B_b\}$	`refocusBMsrCond`  **BC**: $B_1, \cdots, B_b$
`refocusBMsrCond(`B_{old} `->` B_{new}`)` **Precondition:** $B_{new} \in BMsrPredicates_C$ with $C = CubeSchema_{src}$, $B_{old} \in BMsrConds_{src}$, $B_{new} \notin BMsrConds_{src}$ **Postcondition:** $BMsrConds_{trg} = BMsrConds_{src} \backslash \{B_{old}\} \cup \{B_{new}\}$	`refocusBMsrCond`  **BC**: $B_{old} \rightarrow B_{new}$

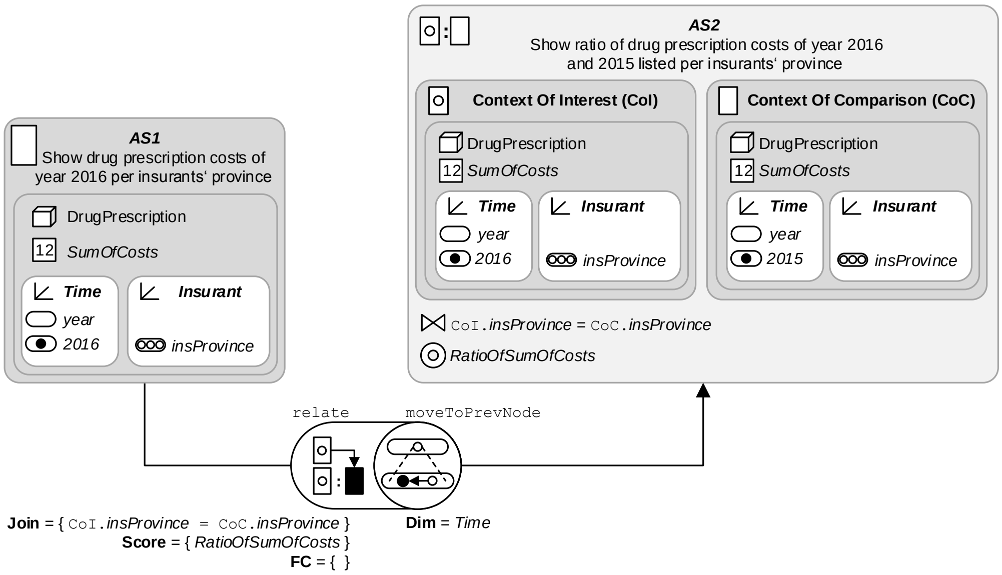


Figure 3.2: Example of a comparative navigation step

3.2.5 Operators Changing Measures

see Table 3.6

3.2.6 Operators Changing Filters

see Table 3.7

3.2.7 Operators Changing Cube Access

see Table 3.8

3.3 Operators Involving Comparison

see Figure 3.2

3.3.1 Operators Introducing Comparison

see Table 3.9

Table 3.6: Operators `addMeasure`, `dropMeasure`, `refocusMeasure`, `move-DownToMeasure`, and `moveUpToMeasure`

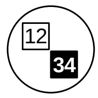
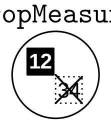
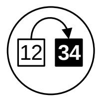
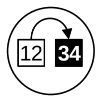
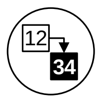
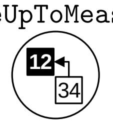
Operator Definition	**Symbol**
`addMeasure(`$+M_1, \cdots, +M_m$`)` **Precondition:** $M_1, \cdots, M_m \in Msrs_C$ with $C = CubeSchema_{src}$, $\{M_1, \cdots, M_m\} \nsubseteq Msrs_{src}$ **Postcondition:** $Msrs_{trg} = Msrs_{src} \cup \{M_1, \cdots, M_m\}$	`addMeasure`  **Msr**: $+M_1, \cdots, +M_m$
`dropMeasure(`$-M_1, \cdots, -M_m$`)` **Precondition:** $\{M_1, \cdots, M_m\} \subseteq Msrs_{src}$ **Postcondition:** $Msrs_{trg} = Msrs_{src} \backslash \{M_1, \cdots, M_m\}$	`dropMeasure`  **Msr**: $-M_1, \cdots, -M_m$
`refocusMeasure(`$M_1, \cdots, M_m$`)` **Precondition:** $M_1, \cdots, M_m \in Msrs_C$ with $C = CubeSchema_{src}$, $\{M_1, \cdots, M_m\} \neq Msrs_{src}$ **Postcondition:** $Msrs_{trg} = \{M_1, \cdots, M_m\}$	`refocusMeasure`  **Msr** $= M_1, \cdots, M_m$
`refocusMeasure(`M_{old} `->` M_{new}`)` **Precondition:** $M_{new} \in Msrs_C$ with $C = CubeSchema_{src}$, $M_{old} \in Msrs_{src}$, $M_{new} \notin Msrs_{src}$ **Postcondition:** $Msrs_{trg} =$ $Msrs_{src} \backslash \{M_{old}\} \cup \{M_{new}\}$	`refocusMeasure`  **Msr**: $M_{old} \rightarrow M_{new}$
`moveDownToMeasure(`M_{old} `->` M_{new}`)` **Precondition:** $M_{new} \in Msrs_C$ with $C = CubeSchema_{src}$, $M_{new} \notin Msrs_{src}$, $M_{new} \twoheadrightarrow M_{old}$ **Postcondition:** $Msrs_{trg} = Msrs_{src} \backslash \{M_{old}\} \cup \{M_{new}\}$	`moveDownToMeasure`  **Msr**: $M_{old} \rightarrow M_{new}$
`moveUpToMeasure(`M_{old} `->` M_{new}`)` **Precondition:** $M_{new} \in Msrs_C$ with $C = CubeSchema_{src}$, $M_{new} \notin Msrs_{src}$, $M_{old} \twoheadrightarrow M_{new}$ **Postcondition:** $Msrs_{trg} = Msrs_{src} \backslash \{M_{old}\} \cup \{M_{new}\}$	`moveUpToMeasure`  **Msr**: $M_{old} \rightarrow M_{new}$

Table 3.7: Operator `narrowFilter`

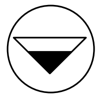
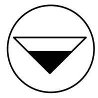
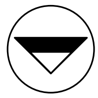
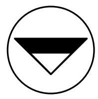
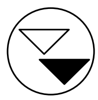
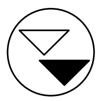
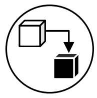
Operator Definition	**Symbol**
`narrowFilter(+`F_1`,`$\cdots$`,+`F_k`)` **Precondition:** $F_1, \cdots, F_k \in \mathit{MsrPredicates}_C$ with $C = \mathit{CubeSchema}_{src}$, $\{F_1, \cdots, F_k\} \nsubseteq \mathit{FilterConds}_{src}$ **Postcondition:** $\mathit{FilterConds}_{trg} = \mathit{FilterConds}_{src} \cup \{F_1, \cdots, F_k\}$	`narrowFilter`  **FC**: $+F_1, \cdots, +F_k$
`narrowFilter(`F_{old} `->` F_{new}`)` **Precondition:** $F_{new} \in \mathit{MsrPredicates}_C$ with $C = \mathit{CubeSchema}_{src}$, $F_{old} \in \mathit{FilterConds}_{src}$, $F_{new} \notin \mathit{FilterConds}_{src}$, $F_{new} \Rightarrow F_{old}$ **Postcondition:** $\mathit{FilterConds}_{trg} = \mathit{FilterConds}_{src} \backslash \{F_{old}\} \cup \{F_{new}\}$	`narrowFilter`  **FC**: $F_{old} \rightarrow F_{new}$
`broadenFilter(-`F_1`,`$\cdots$`,-`F_k`)` **Precondition:** $\{F_1, \cdots, F_k\} \subseteq \mathit{FilterConds}_{src}$ **Postcondition:** $\mathit{FilterConds}_{trg} = \mathit{FilterConds}_{src} \backslash \{F_1, \cdots, F_k\}$	`broadenFilter`  **FC**: $-F_1, \cdots, -F_k$
`broadenFilter(`F_{old} `->` F_{new}`)` **Precondition:** $F_{new} \in \mathit{MsrPredicates}_C$ with $C = \mathit{CubeSchema}_{src}$, $F_{old} \in \mathit{FilterConds}_{src}$, $F_{new} \notin \mathit{FilterConds}_{src}$, $F_{old} \Rightarrow F_{new}$ **Postcondition:** $\mathit{FilterConds}_{trg} = \mathit{FilterConds}_{src} \backslash \{F_{old}\} \cup \{F_{new}\}$	`broadenFilter`  **FC**: $F_{old} \rightarrow F_{new}$
`refocusFilter(`F_1`,`$\cdots$`,`F_k`)` **Precondition:** $F_1, \cdots, F_k \in \mathit{MsrPredicates}_C$ with $C = \mathit{CubeSchema}_{src}$ $\mathit{FilterConds}_{src} \neq \{F_1, \cdots, F_k\}$ **Postcondition:** $\mathit{FilterConds}_{trg} = \{F_1, \cdots, F_k\}$	`refocusFilter`  **FC**: $F_1, \cdots, F_k$
`refocusFilter(`F_{old} `->` F_{new}`)` **Precondition:** $F_{new} \in \mathit{MsrPredicates}_C$ with $C = \mathit{CubeSchema}_{src}$, $F_{old} \in \mathit{FilterConds}_{src}$, $F_{new} \notin \mathit{FilterConds}_{src}$ **Postcondition:** $\mathit{FilterConds}_{trg} = \mathit{FilterConds}_{src} \backslash \{F_{old}\} \cup \{F_{new}\}$	`refocusFilter`  **FC**: $F_{old} \rightarrow F_{new}$

Table 3.8: Operator `drillAcrossToCube`

Operator Definition	**Symbol**
`drillAcrossToCube(`C`,`B`,`M`,`F`)` **Precondition:** C is a cube, $B \subseteq \mathit{BMsrPredicates}_C$, $M \subseteq \mathit{Msrs}_C$, $M \neq \emptyset$, and $F \subseteq \mathit{MsrPredicates}_C$ **Postcondition:** $\mathit{CubeInstance}_{trg} = C$, $\mathit{BMsrConds}_{trg} = B$, and if $D \in \mathit{DimSchemas}_{src} \cap \mathit{DimSchemas}_C$, then $\mathit{DimQual}_{src}(D) \in \mathit{DimQuals}_{trg}$, if $D \in \mathit{DimSchemas}_C \wedge D \notin \mathit{DimSchemas}_{src}$, then $\mathit{DiceLvl}_{dq} = \mathit{top}_D$, $\mathit{DiceNode}_{dq} = \mathit{all}_D$, $\mathit{SliceConds}_{dq} = \emptyset$, and $\mathit{GranLvl}_{dq} = \mathit{top}_D$ with $dq = \mathit{DimQual}_{trg}(D)$, and moreover, $\mathit{Msrs}_{trg} = M$ and $\mathit{FilterConds}_{trg} = F$	`drillAcrossToCube`  **Cube** $= C$ **BC** $= B$ **Msr** $= M$ **FC** $= F$

3.3.2 Operators Changing Comparison

see Table 3.10, 3.11, 3.12, 3.13, 3.14

3.3.3 Operators Dropping Comparison

see Table 3.15

3.4 Use of Analysis Situations as Cubes

see Table 3.16

see Figure 3.3, 3.4

Table 3.9: Operators `relate` and `target`

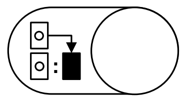
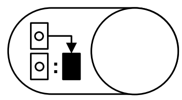
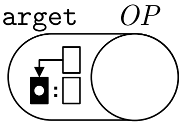
Operator Definition	**Symbol**
<u>`relate(`OP`(`p_1`, ⋯ , `p_q`),`J`,`S`,`F°`)`</u> **Precondition:** J is a non-empty set of join conditions between *src* and *src*`.`OP`(`$p_1, \cdots, p_q$`)`, S is a non-empty set of scores over *src* and *src*`.`OP`(`$p_1, \cdots, p_q$`)`, and F° is a possibly empty set of score filters over *src* and *src*`.`OP`(`$p_1, \cdots, p_q$`)` with scores of S. **Postcondition:** $CoI_{trg} = src$, $CoC_{trg} = src$`.`OP`(`$p_1, \cdots, p_q$`)`, $JoinConds_{trg} = J$, $Scores_{trg} = S$, $ScoreFilters_{trg} = F^\circ$	`relate` OP  **Join** $= J$ p_1 **Score** $= S$ $\cdots$ **FC** $= F^\circ$ p_q
<u>`relate(`J`,`S`,`F°`)`</u> **Precondition:** J is a non-empty set of join conditions between *src* and *src*, S is a non-empty set of scores over *src* and *src*, and F° is a possibly empty set of score filters over *src* and *src* with scores of S. **Postcondition:** $CoI_{trg} = src$, $CoC_{trg} = src$, $JoinConds_{trg} = J$, $Scores_{trg} = S$, $ScoreFilters_{trg} = F^\circ$	`relate`  **Join** $= J$ **Score** $= S$ **FC** $= F^\circ$
<u>`target(`OP`(`p_1`, ⋯ , `p_q`),`J`,`S`,`F°`)`</u> **Precondition:** J is a non-empty set of join conditions between *src* and *src*`.`OP`(`$p_1, \cdots, p_q$`)`, S is a non-empty set of scores over *src* and *src*`.`OP`(`$p_1, \cdots, p_q$`)`, and F° is a possibly empty set of score filters over *src* and *src*`.`OP`(`$p_1, \cdots, p_q$`)` with scores of S. **Postcondition:** $CoC_{trg} = src$, $CoI_{trg} = src$`.`OP`(`$p_1, \cdots, p_q$`)`, $JoinConds_{trg} = J$, $Scores_{trg} = S$, $ScoreFilters_{trg} = F^\circ$	`target` OP  **Join** $= J$ p_1 **Score** $= S$ $\cdots$ **FC** $= F^\circ$ p_q

Table 3.10: Operator `rerelate`

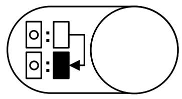
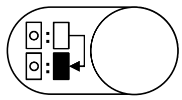
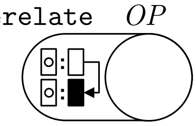
Operator Definition	**Symbol**
`rerelate(`OP`(`$p_1, \cdots, p_q$`),`J`)` **Precondition:** J is a non-empty set of join conditions between CoI_{src} and CoC_{src} . OP`(`$p_1, \cdots, p_q$`)` **Postcondition:** $CoI_{trg} = CoI_{src}$, $CoC_{trg} = CoC_{src}$. OP`(`$p_1, \cdots, p_q$`)`, $JoinConds_{trg} = J$, $Scores_{trg} = Scores_{src}$, $ScoreFilters_{trg} = ScoreFilters_{src}$	`rerelate` OP  **Join** $= J$ p_1 $\cdots$ p_q
`rerelate(`OP`(`$p_1, \cdots, p_q$`),`J`,`S`,`F°`)` **Precondition:** J is a non-empty set of join conditions between CoI_{src} and CoC_{src} . OP`(`$p_1, \cdots, p_q$`)`, S is a non-empty set of scores over CoI_{src} and CoC_{src} . OP`(`$p_1, \cdots, p_q$`)`, and F° is a possibly empty set of score filters over CoI_{src} and CoC_{src} . OP`(`$p_1, \cdots, p_q$`)` with scores of S. **Postcondition:** $CoI_{trg} = CoI_{src}$, $CoC_{trg} = CoC_{src}$. OP`(`$p_1, \cdots, p_q$`)`, $JoinConds_{trg} = J$, $Scores_{trg} = S$, $ScoreFilters_{trg} = F^\circ$	`rerelate` OP  **Join** $= J$ p_1 **Score** $= S$ $\cdots$ **FC** $= F^\circ$ p_q
`rerelate(`OP`(`$p_1, \cdots, p_q$`))` **Precondition:** $JoinConds_{src}$ are also join conditions between CoI_{src} and CoC_{src} . OP`(`$p_1, \cdots, p_q$`)`, $Scores_{src}$ are also scores over CoI_{src} and CoC_{src} . OP`(`$p_1, \cdots, p_q$`)`, and $ScoreFilters_{src}$ are also scores over filters over CoI_{src} and CoC_{src} . OP`(`$p_1, \cdots, p_q$`)` with scores $Scores_{src}$. **Postcondition:** $CoI_{trg} = CoI_{src}$, $CoC_{trg} = CoC_{src}$. OP`(`$p_1, \cdots, p_q$`)`, $JoinConds_{trg} = JoinConds_{src}$, $Scores_{trg} = Scores_{src}$, $ScoreFilters_{trg} = ScoreFilters_{src}$	`rerelate` OP  p_1 $\cdots$ p_q

Table 3.11: Operator `retarget`

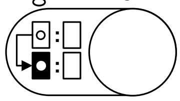
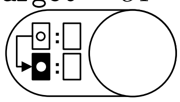
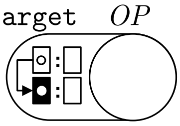
Operator Definition	**Symbol**
$\underline{\texttt{retarget(}OP\texttt{(}p_1,\ \cdots,\ p_q\texttt{)},J\texttt{)}}$ **Precondition:** J is a non-empty set of join conditions between $CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$ and CoC_{src} **Postcondition:** $CoI_{trg} = CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$, $CoC_{trg} = CoC_{src}$, $JoinConds_{trg} = J$, $Scores_{trg} = Scores_{src}$, $ScoreFilters_{trg} = ScoreFilters_{src}$	`retarget` OP  **Join** $= J$ p_1 $\cdots$ p_q
$\underline{\texttt{retarget(}OP\texttt{(}p_1,\ \cdots,\ p_q\texttt{)},J\texttt{,}S\texttt{,}F^\circ\texttt{)}}$ **Precondition:** J is a non-empty set of join conditions between $CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$ and CoC_{src}, S is a non-empty set of scores over $CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$ and CoC_{src}, and F° is a possibly empty set of score filters over $CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$ and CoC_{src} with scores of S. **Postcondition:** $CoI_{trg} = CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$, $CoC_{trg} = CoC_{src}$, $JoinConds_{trg} = J$, $Scores_{trg} = S$, $ScoreFilters_{trg} = F^\circ$	`retarget` OP  **Join** $= J$ p_1 **Score** $= S$ $\cdots$ **FC** $= F^\circ$ p_q
$\underline{\texttt{retarget(}OP\texttt{(}p_1,\ \cdots,\ p_q\texttt{))}}$ **Precondition:** $JoinConds_{src}$ are also join conditions between $CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$ and CoC_{src}, $Scores_{src}$ are also scores over and $CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$ and CoC_{src}, and $ScoreFilters_{src}$ are also scores over filters over $CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$ and CoC_{src} with scores $Scores_{src}$. **Postcondition:** $CoI_{trg} = CoI_{src}\texttt{.}OP\texttt{(}p_1, \cdots, p_q\texttt{)}$, $CoC_{trg} = CoC_{src}$, $JoinConds_{trg} = JoinConds_{src}$, $Scores_{trg} = Scores_{src}$, $ScoreFilters_{trg} = ScoreFilters_{src}$	`retarget` OP  p_1 $\cdots$ p_q

Table 3.12: Operator `correlate`

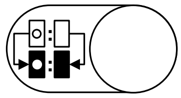
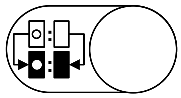
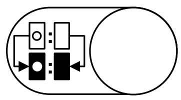
Operator Definition	**Symbol**
`correlate(`OP`(`$p_1, \cdots, p_q$`),`J`)` **Precondition:** J is a non-empty set of join conditions between $CoI_{src}.OP(p_1, \cdots, p_q)$ and $CoC_{src}.OP(p_1, \cdots, p_q)$ **Postcondition:** $CoI_{trg} = CoI_{src}.OP(p_1, \cdots, p_q)$, $CoC_{trg} = CoC_{src}.OP(p_1, \cdots, p_q)$, $JoinConds_{trg} = J$, $Scores_{trg} = Scores_{src}$, $ScoreFilters_{trg} = ScoreFilters_{src}$	`correlate` OP  **Join** $= J$ p_1 $\cdots$ p_q
`correlate(`OP`(`$p_1, \cdots, p_q$`),`J`,`S`,`F°`)` **Precondition:** J is a non-empty set of join conditions between $CoI_{src}.OP(p_1, \cdots, p_q)$ and $CoC_{src}.OP(p_1, \cdots, p_q)$, S is a non-empty set of scores over $CoI_{src}.OP(p_1, \cdots, p_q)$ and $CoC_{src}.OP(p_1, \cdots, p_q)$, and F° is a possibly empty set of score filters over $CoI_{src}.OP(p_1, \cdots, p_q)$ and $CoC_{src}.OP(p_1, \cdots, p_q)$ with scores of S. **Postcondition:** $CoI_{trg} = CoI_{src}.OP(p_1, \cdots, p_q)$, $CoC_{trg} = CoC_{src}.OP(p_1, \cdots, p_q)$, $JoinConds_{trg} = J$, $Scores_{trg} = S$, $ScoreFilters_{trg} = F^\circ$	`correlate` OP  **Join** $= J$ p_1 **Score** $= S$ $\cdots$ **FC** $= F^\circ$ p_q
`correlate(`OP`(`$p_1, \cdots, p_q$`))` **Precondition:** $JoinConds_{src}$ are also join conditions between $CoI_{src}.OP(p_1, \cdots, p_q)$ and $CoC_{src}.OP(p_1, \cdots, p_q)$, $Scores_{src}$ are also scores over $CoI_{src}.OP(p_1, \cdots, p_q)$ and $CoC_{src}.OP(p_1, \cdots, p_q)$, and $ScoreFilters_{src}$ are also scores over filters over $CoI_{src}.OP(p_1, \cdots, p_q)$ and $CoC_{src}.OP(p_1, \cdots, p_q)$ with scores $Scores_{src}$. **Postcondition:** $CoI_{trg} = CoI_{src}.OP(p_1, \cdots, p_q)$, $CoC_{trg} = CoC_{src}.OP(p_1, \cdots, p_q)$, $JoinConds_{trg} = JoinConds_{src}$, $Scores_{trg} = Scores_{src}$, $ScoreFilters_{trg} = ScoreFilters_{src}$	`correlate` OP  p_1 $\cdots$ p_q

Table 3.13: Operators `narrowFilter`, `broadenFilter`, and `refocusFilter` for score filters

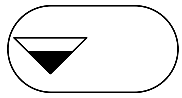
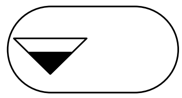
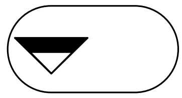
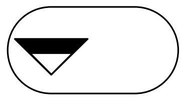
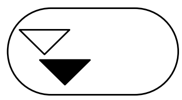
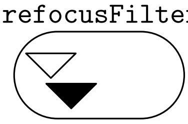
Operator Definition	**Symbol**
`narrowFilter(+`F_1°`,`$\cdots$`,+`F_w°`)` **Precondition:** $F_1^\circ, \cdots, F_w^\circ$ are score filters over CoI_{src} and CoC_{src} with scores in $Scores_{src}$, and $\{F_1^\circ, \cdots, F_w^\circ\} \nsubseteq ScoreFilters_{src}$. **Postcondition:** $ScoreFilters_{trg} = ScoreFilters_{src} \cup \{F_1^\circ, \cdots, F_w^\circ\}$	`narrowFilter`  **FC**: $+F_1^\circ, \cdots, +F_w^\circ$
`narrowFilter(`F_{old}°` -> `F_{new}°`)` **Precondition:** F_{new}° is a score filter over CoI_{src} and CoC_{src} with scores in $Scores_{src}$, $F_{old}^\circ \in ScoreFilters_{src}$, $F_{new}^\circ \notin ScoreFilters_{src}$, and $F_{new}^\circ \Rightarrow F_{old}^\circ$. **Postcondition:** $ScoreFilters_{trg} = ScoreFilters_{src} \setminus \{F_{old}^\circ\} \cup \{F_{new}^\circ\}$	`narrowFilter`  **FC**: $F_{old}^\circ \rightarrow F_{new}^\circ$
`broadenFilter(-`F_1°`,`$\cdots$`,-`F_w°`)` **Precondition:** $\{F_1^\circ, \cdots, F_w^\circ\} \subseteq ScoreFilters_{src}$. **Postcondition:** $ScoreFilters_{trg} = ScoreFilters_{src} \setminus \{F_1^\circ, \cdots, F_w^\circ\}$	`broadenFilter`  **FC**: $-F_1^\circ, \cdots, -F_w^\circ$
`broadenFilter(`F_{old}°` -> `F_{new}°`)` **Precondition:** F_{new}° is a score filter over CoI_{src} and CoC_{src} with scores in $Scores_{src}$, and $F_{old}^\circ \in ScoreFilters_{src}$, $F_{new}^\circ \notin ScoreFilters_{src}$, and $F_{old}^\circ \Rightarrow F_{new}^\circ$. **Postcondition:** $ScoreFilters_{trg} = ScoreFilters_{src} \setminus \{F_{old}^\circ\} \cup \{F_{new}^\circ\}$	`broadenFilter`  **FC**: $F_{old}^\circ \rightarrow F_{new}^\circ$
`refocusFilter(`F_1°`,`$\cdots$`,`F_w°`)` **Precondition:** $F_1^\circ, \cdots, F_w^\circ$ are score filters over CoI_{src} and CoC_{src} with scores in $Scores_{src}$, and $\{F_1^\circ, \cdots, F_w^\circ\} \neq ScoreFilters_{src}$. **Postcondition:** $ScoreFilters_{trg} = \{F_1^\circ, \cdots, F_w^\circ\}$	`refocusFilter`  **FC**: $F_1^\circ, \cdots, F_w^\circ$
`refocusFilter(`F_{old}°` -> `F_{new}°`)` **Precondition:** F_{new}° is a score filter over CoI_{src} and CoC_{src} with scores in $Scores_{src}$, $F_{old}^\circ \in ScoreFilters_{src}$, and $F_{new}^\circ \notin ScoreFilters_{src}$. **Postcondition:** $ScoreFilters_{trg} = ScoreFilters_{src} \setminus \{F_{old}^\circ\} \cup \{F_{new}^\circ\}$	`refocusFilter`  **FC**: $F_{old}^\circ \rightarrow F_{new}^\circ$

Table 3.14: Operator `rejoin`

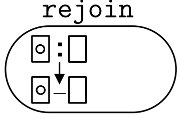
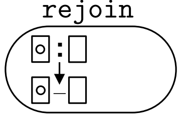
Operator Definition	**Symbol**
`rejoin(`J`)` **Precondition:** J is a set of join conditions between CoI_{src} and CoC_{src}, **Postcondition:** $JoinConds_{trg} = J, Scores_{trg} = Scores_{src}$, $ScoreFilters_{trg} = ScoreFilters_{src}$	`rejoin`  **Join** $= J$
`rejoin(`J`,`S`,`F°`)` **Precondition:** J is a set of join conditions between CoI_{src} and CoC_{src}, S is a set of scores over CoI_{src} and CoC_{src}, and F are score filters over CoI_{src} and CoC_{src} with scores in S. **Postcondition:** $JoinConds_{trg} = J, Scores_{trg} = S, ScoreFilters_{trg} = F$	`rejoin`  **Join** $= J$ **Score** $= S$ **FC** $= F^\circ$

Table 3.15: Operators `unrelate` and `untarget`

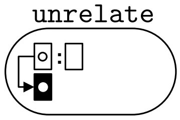
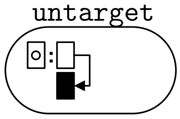
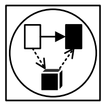
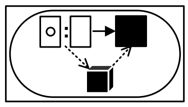
Operator Definition	**Symbol**
`unrelate()` **Precondition:** *No additional preconditons* **Postcondition:** $trg = CoC_{src}$	`unrelate` 
`untarget()` **Precondition:** *No additional preconditons* **Postcondition:** $trg = CoI_{src}$	`untarget` 

Table 3.16: Operator `useAsCube`

Operator Definition	**Symbol**
<u>`useAsCube(`C`, `$\mathcal{B}$`, `M`)`</u> **Precondition:** *src* is a non-comparative analysis situation, C is a cube schema derived from *src* such that $BMsrs_C = \mathcal{B}(Msrs_{src})$ where $\mathcal{B}$ is a base measure renaming from *src* to C, and M are measures applicable to $BMsrs_C$. **Postcondition:** *trg* is a non-comparative analysis, situation, $CubeSchema_{trg} = C$, $Msrs_{trg} = M$, for all $D \in CubeSchema_C$, $DiceLvl_{trg}(D) = top_D$, $DiceNode_{trg}(D) = all_D$, $SliceConds_{trg}(D) = \emptyset$, $GranLvl_{trg}(D) = base_D$, and $FilterConds_{trg} = \emptyset$.	`useAsCube`  **Cube** $= C$ **BMsrs** $= \mathcal{B}$ **Msrs** $= M$
<u>`useAsCube(`C`, `$\mathcal{B}$`, `$\mathcal{D}$`, `$\mathcal{L}$`, `$\mathcal{A}$`, `M`)`</u> **Precondition:** *src* is a comparative analysis situation, C is a cube schema derived from *src* such that $BMsrs_C = \mathcal{B}(Msrs_{CoI_{src}} \dot{\cup} Msrs_{CoI_{src}} \dot{\cup} Scores_{src})$ where $\mathcal{B}$ is a base measure renaming from *src* to C, and M are measures applicable to $BMsrs_C$. Moreover, dimension schemas of $DimSchemas_C$ are renamed by $\mathcal{D}$, the levels are renamed by $\mathcal{L}$, and descriptive attributes by $\mathcal{A}$. **Postcondition:** *trg* is a non-comparative analysis, situation, $CubeSchema_{trg} = C$, $Msrs_{trg} = M$, for all $D \in CubeSchema_C$, $DiceLvl_{trg}(D) = top_D$, $DiceNode_{trg}(D) = all_D$, $SliceConds_{trg}(D) = \emptyset$, $GranLvl_{trg}(D) = base_D$, and $FilterConds_{trg} = \emptyset$.	`useAsCube`  **Cube** $= C$ **BMsrs** $= \mathcal{M}$ **Dims** $= \mathcal{D}$ **Lvls** $= \mathcal{L}$ **Attrs** $= \mathcal{A}$ **Msrs** $= M$

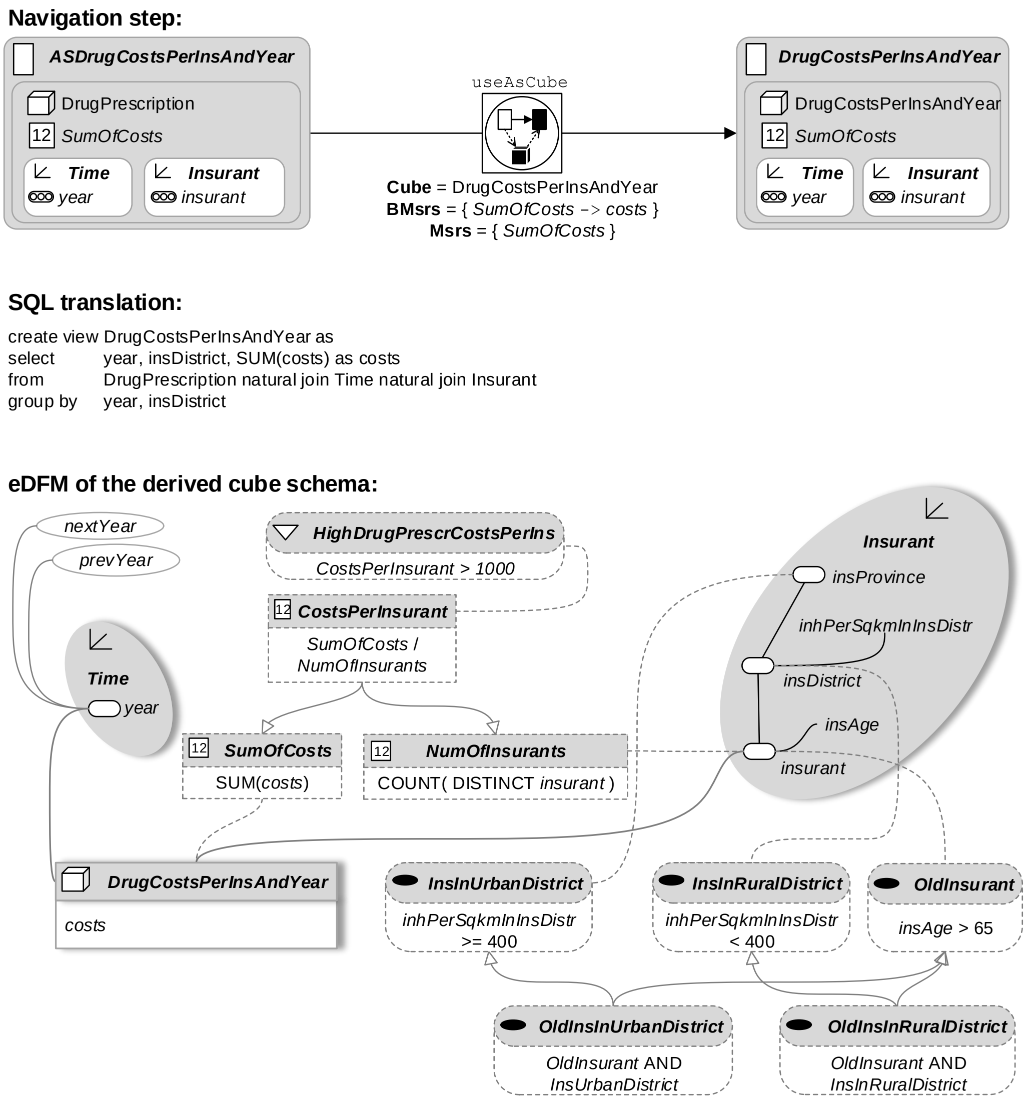
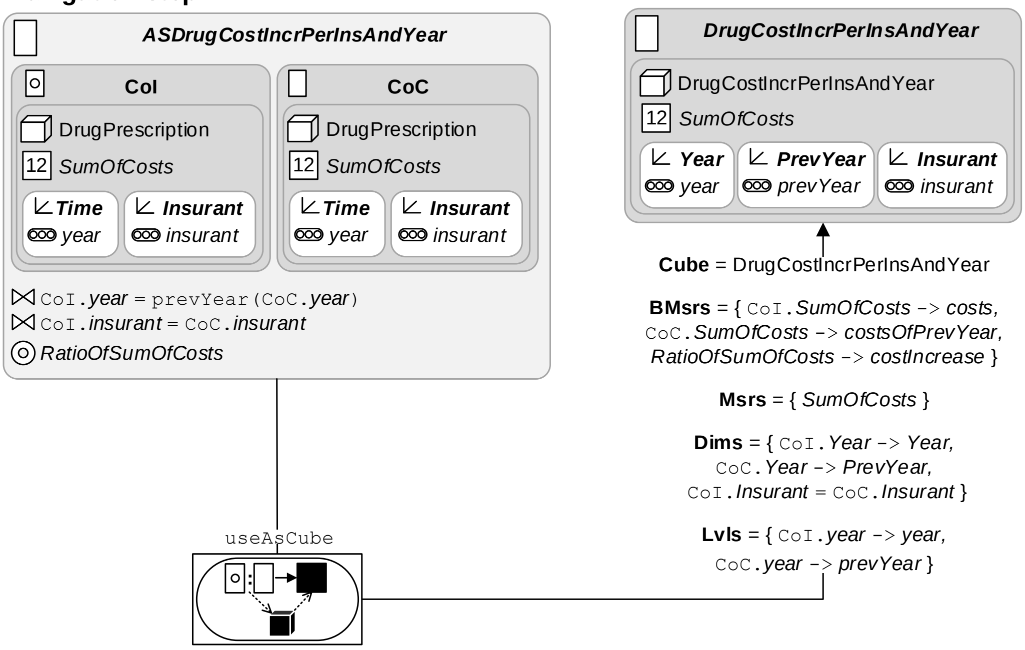


Figure 3.3: Operator `useAsCube` (non-comparative analysis situation)

Navigation step:



SQL translation:

```
create view DrugCostIncrPerInsAndYear as

select      CoI.year as year, CoI.insDistrict as insDistrict, CoI.SumOfCosts as costs,
            CoC.year as prevYear, CoC.SumOfCosts as costsOfPrevYear,
            CoI.SumOfCosts / CoC.SumOfCosts as costIncrease

from        (select     year, insDistrict, SUM(costs) as SumOfCosts
             from       DrugPrescription natural join Time natural join Insurant
             group by   year, insDistrict) CoI

inner join  (select     year, insDistrict, SUM(costs) as SumOfCosts
             from       DrugPrescription natural join Time natural join Insurant
             group by   year, insDistrict) CoC

on          CoI.year = CoC.year + 1
```

Figure 3.4: Operator `useAsCube` (comparative analysis situation)

Chapter 4

Extension to Schema Level

4.1 Analysis Situation Schema

see Figure 4.1, 4.2

4.2 Navigation Step Schema

see Figure 4.3, 4.4, 4.5

4.3 Navigation guards

see Table 4.1, 4.2, 4.4, 4.5

see Figure 4.6, 4.7

Table 4.1: Navigation guard expressions concerning granularity levels (as denotes a non-comparative and cas a comparative analysis situation)

Operator definition
as.`granLevel(`D`)`: If $D \in \mathit{DimSchemas}_{as}$, then as.`granLevel(`D`)` $=$ $\mathit{GranLvl}_{as}(D)$.
cas.`granLevelOfCoI(`D`)`: If $D \in \mathit{DimSchemas}_{\mathit{CoI}_{cas}}$, then cas.`granLevelOfCoI(`D`)` $=$ $\mathit{GranLvl}_{\mathit{CoI}_{cas}}(D)$.
cas.`granLevelOfCoC(`D`)`: If $D \in \mathit{DimSchemas}_{\mathit{CoC}_{cas}}$, then cas.`granLevelOfCoC(`D`)` $=$ $\mathit{GranLvl}_{\mathit{CoC}_{cas}}(D)$.
as.`hasSuperGranLevel(`D`)`: If $D \in \mathit{DimSchemas}_{as}$, then as.`hasSuperGranLevel(`D`)` is true iff $(\exists G \in \mathit{Lvls}_D)(\mathit{GranLvl}_{as}(D) \to G)$
cas.`hasSuperGranLevelInCoI(`D`)`: If $D \in \mathit{DimSchemas}_{\mathit{CoI}_{cas}}$, then cas.`hasSuperGranLevelInCoI(`D`)` is true iff $(\exists G \in \mathit{Lvls}_D)(\mathit{GranLvl}_{\mathit{CoI}_{cas}}(D) \to G)$
cas.`hasSuperGranLevelInCoC(`D`)`: If $D \in \mathit{DimSchemas}_{\mathit{CoC}_{cas}}$, then cas.`hasSuperGranLevelInCoC(`D`)` is true iff $(\exists G \in \mathit{Lvls}_D)(\mathit{GranLvl}_{\mathit{CoC}_{cas}}(D) \to G)$
as.`hasSubGranLevel(`D`)`: If $D \in \mathit{DimSchemas}_{as}$, then as.`hasSubGranLevel(`D`)` is true iff $(\exists G \in \mathit{Lvls}_D)(G \to \mathit{GranLvl}_{as}(D))$
cas.`hasSubGranLevelInCoI(`D`)`: If $D \in \mathit{DimSchemas}_{\mathit{CoI}_{cas}}$, then cas.`hasSubGranLevelInCoI(`D`)` is true iff $(\exists G \in \mathit{Lvls}_D)(G \to \mathit{GranLvl}_{\mathit{CoI}_{cas}}(D))$
cas.`hasSubGranLevelInCoC(`D`)`: If $D \in \mathit{DimSchemas}_{\mathit{CoC}_{cas}}$, then cas.`hasSubGranLevelInCoC(`D`)` is true iff $(\exists G \in \mathit{Lvls}_D)(G \to \mathit{GranLvl}_{\mathit{CoC}_{cas}}(D))$

Table 4.2: Navigation guard expressions concerning dice levels (as denotes a non-comparative and cas a comparative analysis situation)

Operator definition
$\underline{as\texttt{.diceLevel(}D\texttt{):}}$ If $D \in DimSchemas_{as}$, then $as\texttt{.diceLevel(}D\texttt{)} = DiceLvl_{as}(D)$.
$\underline{cas\texttt{.diceLevelOfCoI(}D\texttt{):}}$ If $D \in DimSchemas_{CoI_{cas}}$, then $cas\texttt{.diceLevelOfCoI(}D\texttt{)} = DiceLvl_{CoI_{cas}}(D)$.
$\underline{cas\texttt{.diceLevelOfCoC(}D\texttt{):}}$ If $D \in DimSchemas_{CoC_{cas}}$, then $cas\texttt{.diceLevelOfCoC(}D\texttt{)} = DiceLvl_{CoC_{cas}}(D)$.
$\underline{as\texttt{.hasSuperDiceLevel(}D\texttt{):}}$ If $D \in DimSchemas_{as}$, then $as\texttt{.hasSuperDiceLevel(}D\texttt{)}$ is true iff $(\exists L \in Lvls_D)(DiceLvl_{as}(D) \rightarrow L)$
$\underline{cas\texttt{.hasSuperDiceLevelInCoI(}D\texttt{):}}$ If $D \in DimSchemas_{CoI_{cas}}$, then $cas\texttt{.hasSuperDiceLevelInCoI(}D\texttt{)}$ is true iff $(\exists L \in Lvls_D)(DiceLvl_{CoI_{cas}}(D) \rightarrow L)$
$\underline{cas\texttt{.hasSuperDiceLevelInCoC(}D\texttt{):}}$ If $D \in DimSchemas_{CoC_{cas}}$, then $cas\texttt{.hasSuperDiceLevelInCoC(}D\texttt{)}$ is true iff $(\exists L \in Lvls_D)(DiceLvl_{CoC_{cas}}(D) \rightarrow L)$
$\underline{as\texttt{.hasSubDiceLevel(}D\texttt{):}}$ If $D \in DimSchemas_{as}$, then $as\texttt{.hasSubDiceLevel(}D\texttt{)}$ is true iff $(\exists L \in Lvls_D)(L \rightarrow DiceLvl_{as}(D))$
$\underline{cas\texttt{.hasSubDiceLevelInCoI(}D\texttt{):}}$ If $D \in DimSchemas_{CoI_{cas}}$, then $cas\texttt{.hasSubDiceLevelInCoI(}D\texttt{)}$ is true iff $(\exists L \in Lvls_D)(L \rightarrow DiceLvl_{CoI_{cas}}(D))$
$\underline{cas\texttt{.hasSubDiceLevelInCoC(}D\texttt{):}}$ If $D \in DimSchemas_{CoC_{cas}}$, then $cas\texttt{.hasSubDiceLevelInCoC(}D\texttt{)}$ is true iff $(\exists L \in Lvls_D)(L \rightarrow DiceLvl_{CoC_{cas}}(D))$

Table 4.3: Navigation guard expressions concerning dice nodes (as denotes a non-comparative and cas a comparative analysis situation)

Operator definition
as`.diceNode(`D`)`: If $D \in DimSchemas_{as}$, then as`.diceNode(`D`)` $= DiceNode_{as}(D)$.
cas`.diceNodeOfCoI(`D`)`: If $D \in DimSchemas_{CoI_{cas}}$, then cas`.diceNodeOfCoI(`D`)` $= DiceNode_{CoI_{cas}}(D)$.
cas`.diceNodeOfCoC(`D`)`: If $D \in DimSchemas_{CoC_{cas}}$, then cas`.diceNodeOfCoC(`D`)` $= DiceNode_{CoC_{cas}}(D)$.
as`.hasSuperDiceNode(`D`)`: If $D \in DimSchemas_{as}$, then as`.hasSuperDiceNode(`D`)` is true iff $(\exists N \in Nodes_{as}(D))(DiceNode_{as}(D) \rightarrow N)$
cas`.hasSuperDiceNodeInCoI(`D`)`: If $D \in DimSchemas_{CoI_{cas}}$, then cas`.hasSuperDiceNodeInCoI(`D`)` is true iff $(\exists N \in Nodes_{CoI_{as}}(D))(DiceNode_{CoI_{cas}}(D) \rightarrow N)$
cas`.hasSuperDiceNodeInCoC(`D`)`: If $D \in DimSchemas_{CoC_{cas}}$, then cas`.hasSuperDiceNodeInCoC(`D`)` is true iff $(\exists N \in Nodes_{CoC_{as}}(D))(DiceNode_{CoC_{cas}}(D) \rightarrow N)$
as`.hasSubDiceNode(`D`)`: If $D \in DimSchemas_{as}$, then as`.hasSubDiceNode(`D`)` is true iff $(\exists N \in Nodes_{as}(D))(N \rightarrow DiceNode_{as}(D))$
cas`.hasSubDiceNodeInCoI(`D`)`: If $D \in DimSchemas_{CoI_{cas}}$, then cas`.hasSubDiceNodeInCoI(`D`)` is true iff $(\exists N \in Nodes_{CoI_{as}}(D))(N \rightarrow DiceNode_{CoI_{cas}}(D))$
cas`.hasSubDiceNodeInCoC(`D`)`: If $D \in DimSchemas_{CoC_{cas}}$, then cas`.hasSubDiceNodeInCoC(`D`)` is true iff $(\exists N \in Nodes_{CoC_{as}}(D))(N \rightarrow DiceNode_{CoC_{cas}}(D))$

Table 4.4: Navigation guard expressions examining next and previous dice node (as denotes a non-comparative and cas a comparative analysis situation)

Operator definition
$\underline{as\texttt{.hasNextDiceNode(}D\texttt{):}}$ If $D \in DimSchemas_{as}$, then as`.hasNextDiceNode(`D`)` is true iff $(\exists \hat{N}, N' \in Nodes_{as}(D))(\hat{N} \neq N' \wedge DiceNode_{as}(D) \rightarrow \hat{N} \wedge N' \rightarrow \hat{N} \wedge$ $next(DiceNode_{as}(D)) = N')$
$\underline{cas\texttt{.hasNextDiceNodeInCoI(}D\texttt{):}}$ If $D \in DimSchemas_{CoI_{cas}}$, then cas`.hasNextDiceNodeInCoI(`D`)` is true iff $(\exists \hat{N}, N' \in Nodes_{CoI_{cas}}(D))(\hat{N} \neq N' \wedge DiceNode_{CoI_{cas}}(D) \rightarrow \hat{N} \wedge N' \rightarrow \hat{N} \wedge$ $next(DiceNode_{CoI_{cas}}(D)) = N')$
$\underline{cas\texttt{.hasNextDiceNodeInCoC(}D\texttt{):}}$ If $D \in DimSchemas_{CoC_{cas}}$, then cas`.hasNextDiceNodeInCoC(`D`)` is true iff $(\exists \hat{N}, N' \in Nodes_{CoC_{cas}}(D))(\hat{N} \neq N' \wedge DiceNode_{CoC_{cas}}(D) \rightarrow \hat{N} \wedge N' \rightarrow \hat{N} \wedge$ $next(DiceNode_{CoC_{cas}}(D)) = N')$
$\underline{as\texttt{.hasPrevDiceNode(}D\texttt{):}}$ If $D \in DimSchemas_{as}$, then as`.hasPrevDiceNode(`D`)` is true iff $(\exists \hat{N}, N' \in Nodes_{as}(D))(\hat{N} \neq N' \wedge DiceNode_{as}(D) \rightarrow \hat{N} \wedge N' \rightarrow \hat{N} \wedge$ $prev(DiceNode_{as}(D)) = N')$
$\underline{cas\texttt{.hasPrevDiceNodeInCoI(}D\texttt{):}}$ If $D \in DimSchemas_{CoI_{cas}}$, then cas`.hasPrevDiceNodeInCoI(`D`)` is true iff $(\exists \hat{N}, N' \in Nodes_{CoI_{cas}}(D))(\hat{N} \neq N' \wedge DiceNode_{CoI_{cas}}(D) \rightarrow \hat{N} \wedge N' \rightarrow \hat{N} \wedge$ $prev(DiceNode_{CoI_{cas}}(D)) = N')$
$\underline{cas\texttt{.hasPrevDiceNodeInCoC(}D\texttt{):}}$ If $D \in DimSchemas_{CoC_{cas}}$, then cas`.hasPrevDiceNodeInCoC(`D`)` is true iff $(\exists \hat{N}, N' \in Nodes_{CoC_{cas}}(D))(\hat{N} \neq N' \wedge DiceNode_{CoC_{cas}}(D) \rightarrow \hat{N} \wedge N' \rightarrow \hat{N} \wedge$ $prev(DiceNode_{CoC_{cas}}(D)) = N')$

Table 4.5: Navigation guard expressions examining slice conditions, filter conditions, score filter conditions, and whether a result set is empty (*as* denotes a non-comparative and *cas* a comparative analysis situation)

Operator definition
as`.containsSliceCond(`D`,`P`)`: If $D \in \mathit{DimSchemas}_{as}$ and $P \in \mathit{DimPredicates}_D$, then as`.containsSliceCond(`D`,`P`)` is true iff $(\exists P' \in \mathit{SliceConds}_{as})(P \Rightarrow P')$
cas`.containsSliceCondInCoI(`D`,`P`)`: If $D \in \mathit{DimSchemas}_{CoI_{cas}}$ and $P \in \mathit{DimPredicates}_D$, then cas`.contains-SliceCondInCoI(`D`,`P`)` is true iff $(\exists P' \in \mathit{SliceConds}_{CoI_{cas}})(P \Rightarrow P')$
cas`.containsSliceCondInCoC(`D`,`P`)`: If $D \in \mathit{DimSchemas}_{CoC_{cas}}$ and $P \in \mathit{DimPredicates}_D$, then cas`.contains-SliceCondInCoC(`D`,`P`)` is true iff $(\exists P' \in \mathit{SliceConds}_{CoC_{cas}})(P \Rightarrow P')$
as`.containsBMsrCond(`F`)`: If $B \in \mathit{BMsrPredicates}_{as}$, then as`.containsBMsrCond(`B`)` is true iff $(\exists B' \in \mathit{BMsrConds}_{as})(B \Rightarrow B')$
cas`.containsBMsrCondInCoI(`B`)`: If $B \in \mathit{BMsrPredicates}_{CoI_{cas}}$, then cas`.containsBMsrCondInCoI(`F`)` is true iff $(\exists B' \in \mathit{BMsrConds}_{CoI_{cas}})(B \Rightarrow B')$
cas`.containsBMsrCondInCoC(`B`)`: If $B \in \mathit{BMsrPredicates}_{CoC_{cas}}$, then cas`.containsBMsrCondInCoC(`F`)` is true iff $(\exists B' \in \mathit{BMsrConds}_{CoC_{cas}})(B \Rightarrow B')$
as`.containsFilterCond(`F`)`: If $F \in \mathit{MsrPredicates}_{as}$, then as`.containsFilterCond(`F`)` is true iff $(\exists F' \in \mathit{FilterConds}_{as})(F \Rightarrow F')$
cas`.containsFilterCondInCoI(`F`)`: If $F \in \mathit{MsrPredicates}_{CoI_{cas}}$, then cas`.containsFilterCondInCoI(`F`)` is true iff $(\exists F' \in \mathit{FilterConds}_{CoI_{cas}})(F \Rightarrow F')$
cas`.containsFilterCondInCoC(`F`)`: If $F \in \mathit{MsrPredicates}_{CoC_{cas}}$, then cas`.containsFilterCondInCoC(`F`)` is true iff $(\exists F' \in \mathit{FilterConds}_{CoC_{cas}})(F \Rightarrow F')$
cas`.containsScoreFilterCond(`F`)`: If F is a score filter over CoI_{cas} and CoC_{cas}, then cas`.containsScoreFilterCond(`F`)` is true iff $(\exists F' \in \mathit{ScoreFilters}_{cas})(F \Rightarrow F')$
as`.hasEmptyResult()` or cas`.hasEmptyResult()`: as`.hasEmptyResult()` (cas`.hasEmptyResult()`) is true iff as (cas) has an empty result set.

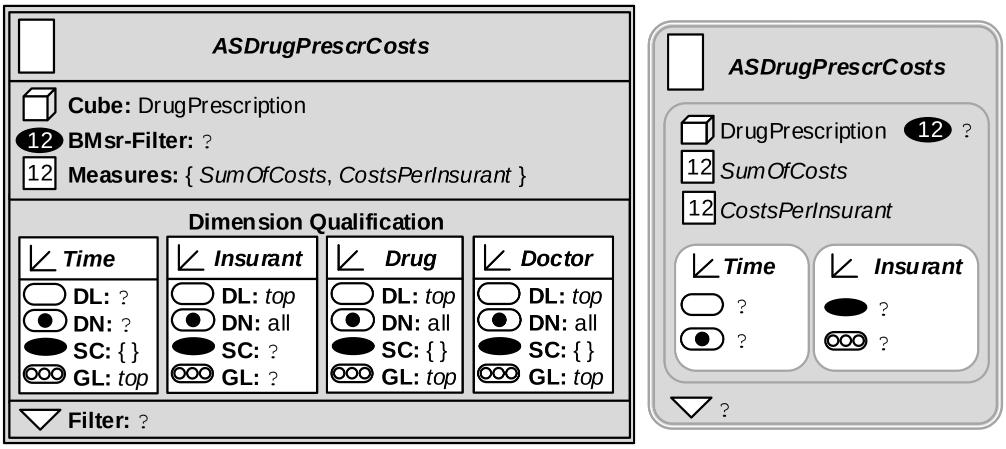


Figure 4.1: Non-comparative analysis situation with unbound variables

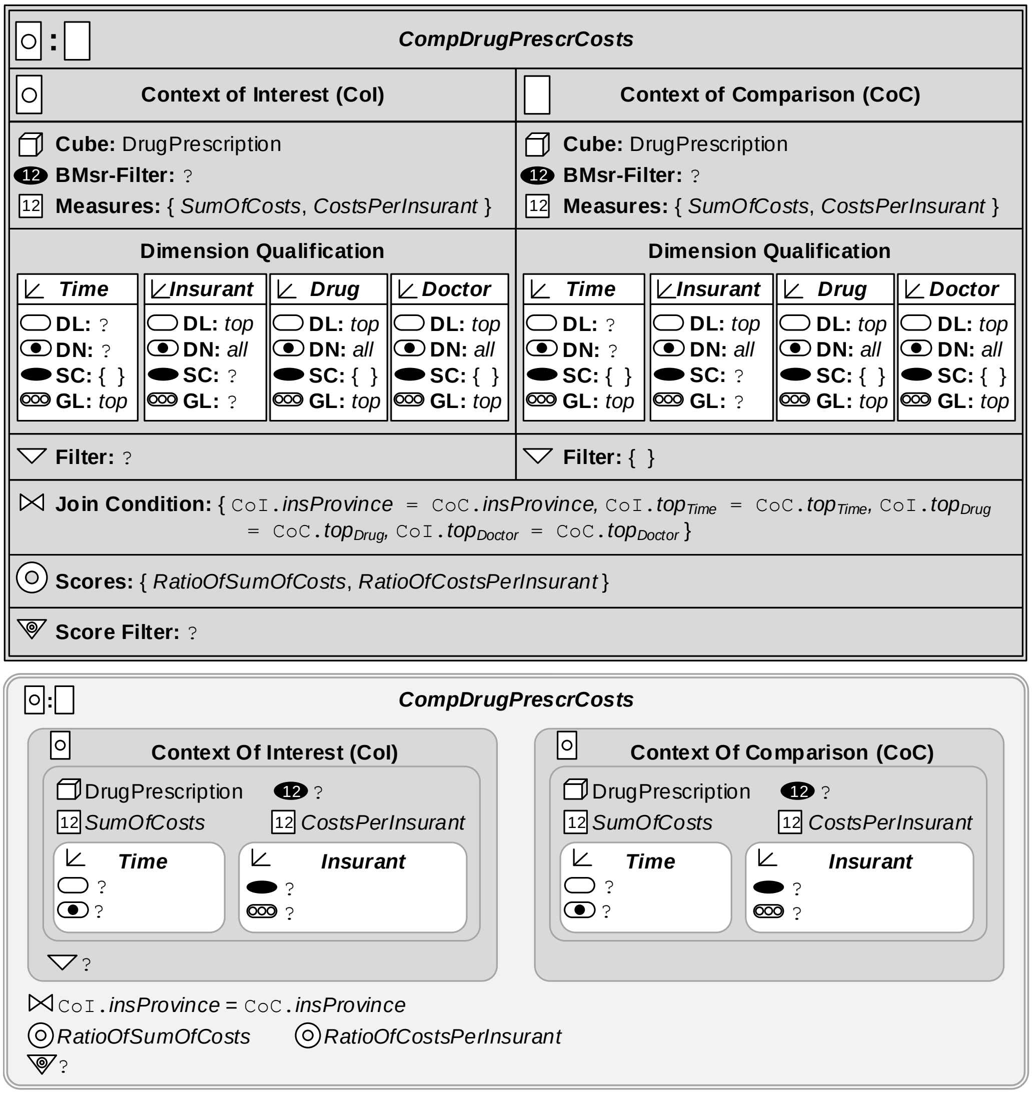


Figure 4.2: Comparative analysis situation with unbound variables

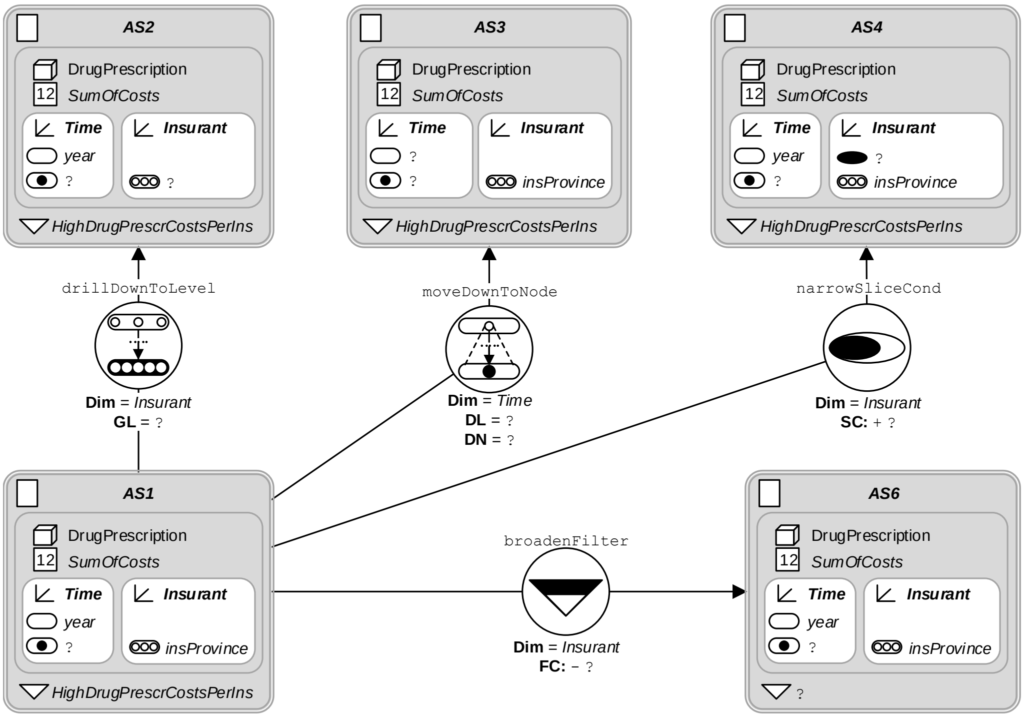


Figure 4.3: Navigation steps with unbound variables

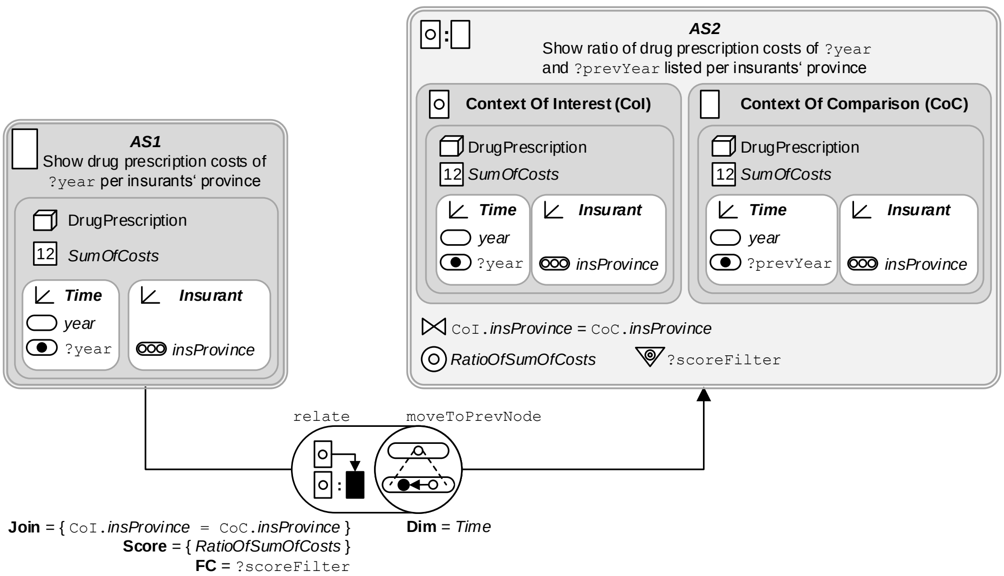


Figure 4.4: Navigation steps with unbound variables involving a comparative analysis situation

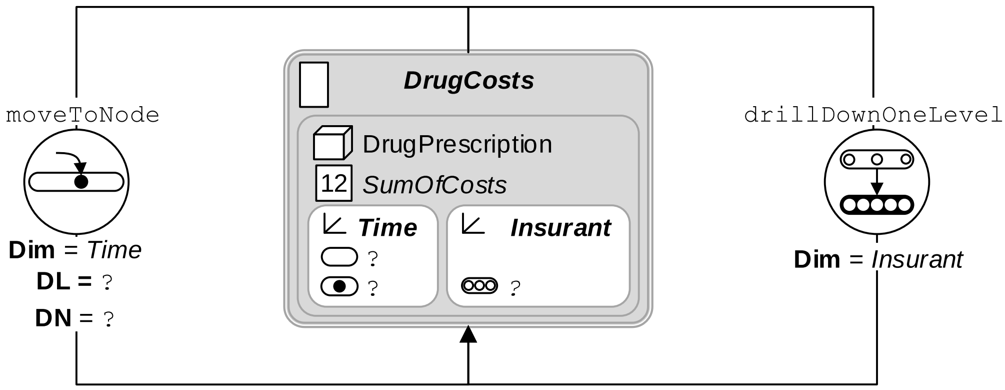


Figure 4.5: Navigation step schemas with equal source and targets

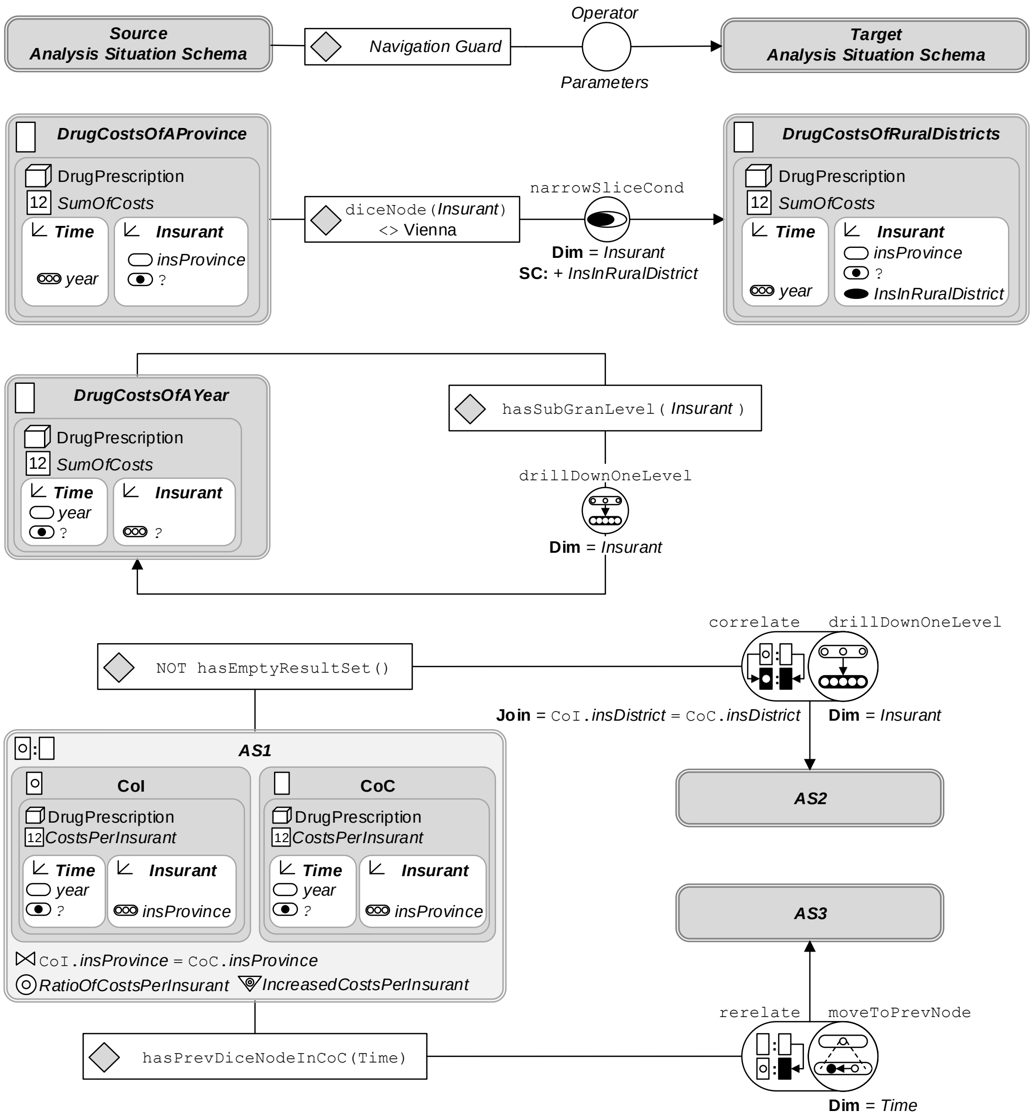


Figure 4.6: Navigation guards

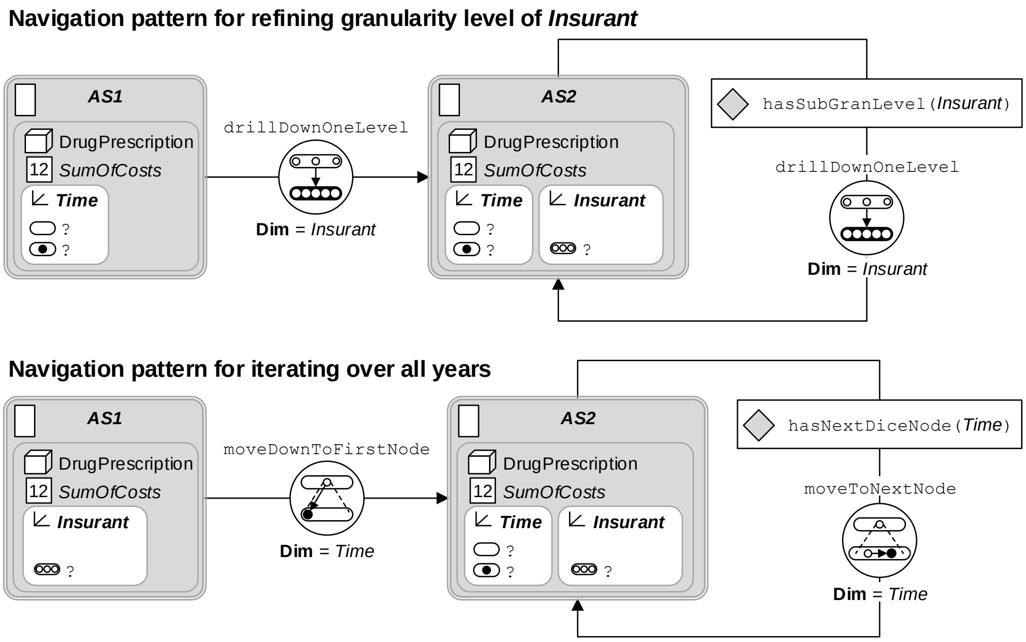


Figure 4.7: Navigation patterns

Chapter 5

Business Intelligence (BI) Analysis Graphs

5.1 Definition of BI Analysis Graphs

see Figure 5.1

5.2 BI Analysis Graph Schemas

see Figure 5.2

5.3 Instances of Analysis Graph Schemas

see Figure 5.3

5.4 Composite analysis situation

see Figure 5.4, 5.5, 5.6

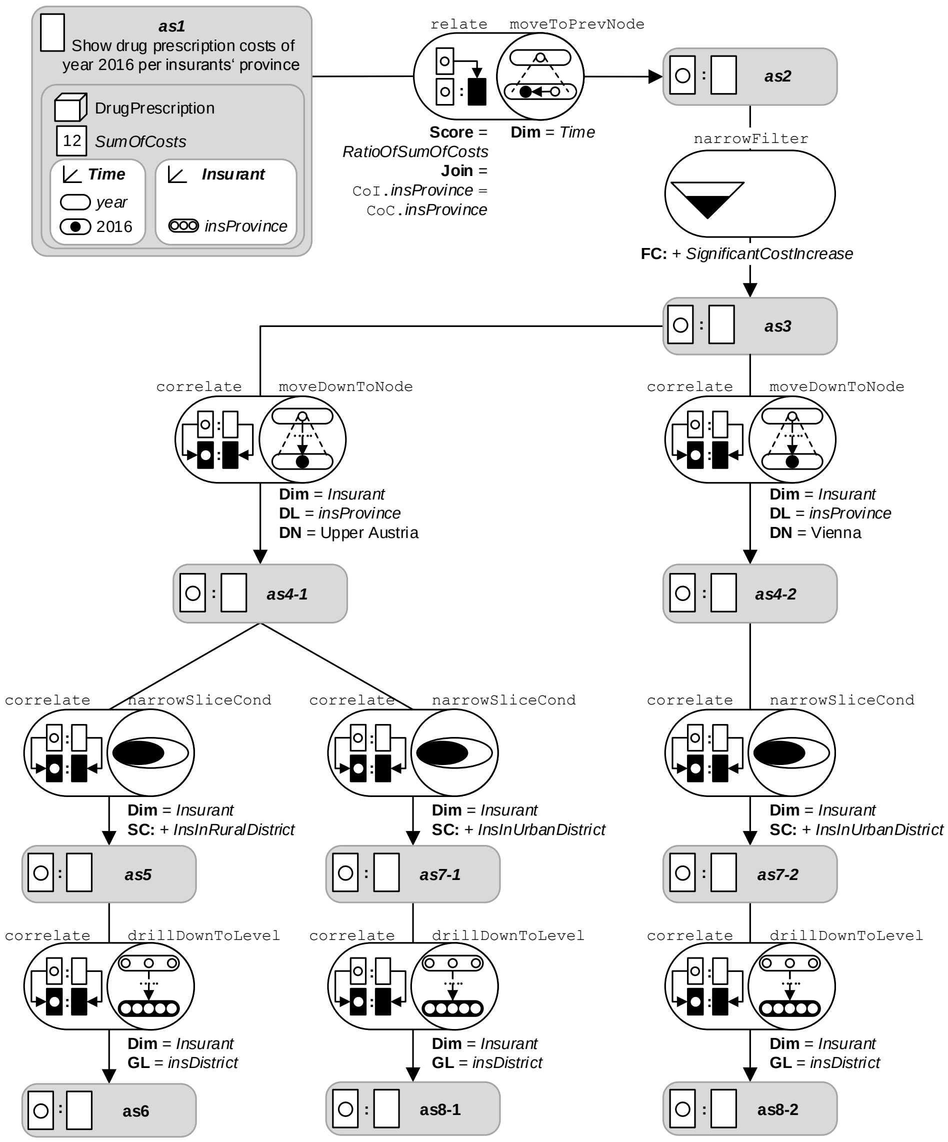


Figure 5.1: Example of BI analysis graph extracted from the running example

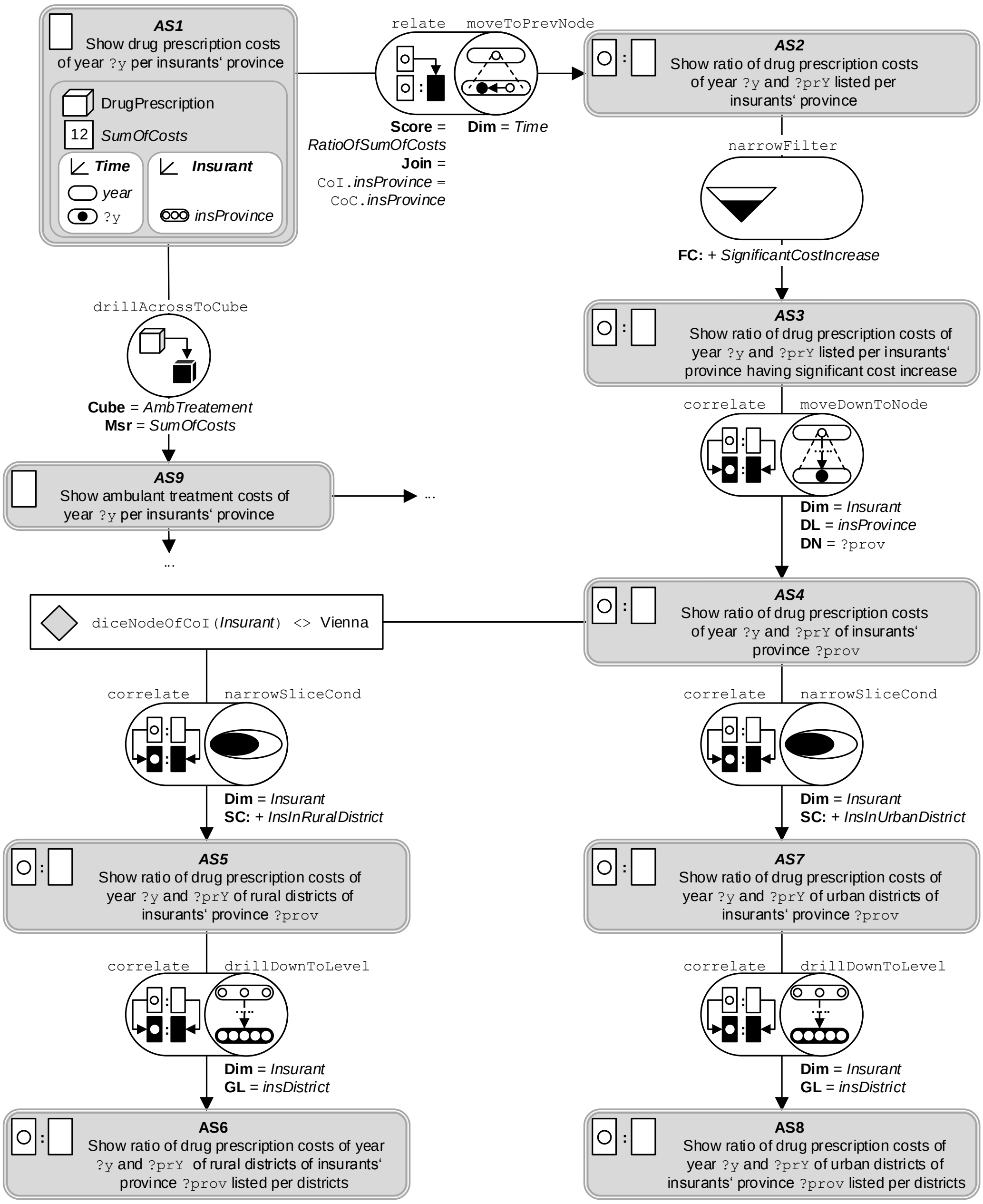


Figure 5.2: Example of BI analysis graph schema extracted from the running example

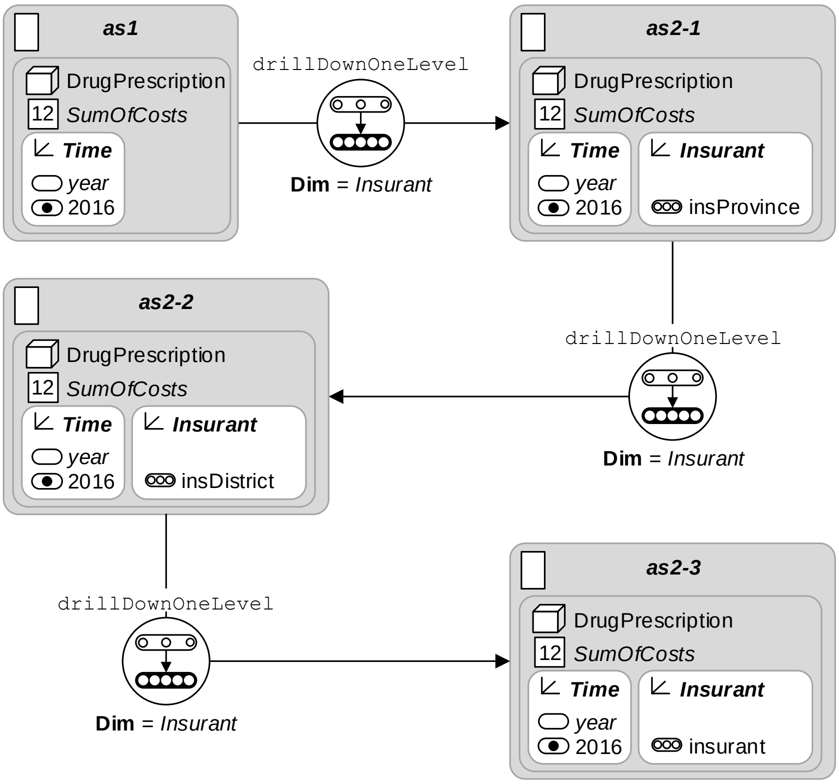


Figure 5.3: Analysis graph instance of analysis graph schema of the first navigation pattern of Figure 4.7

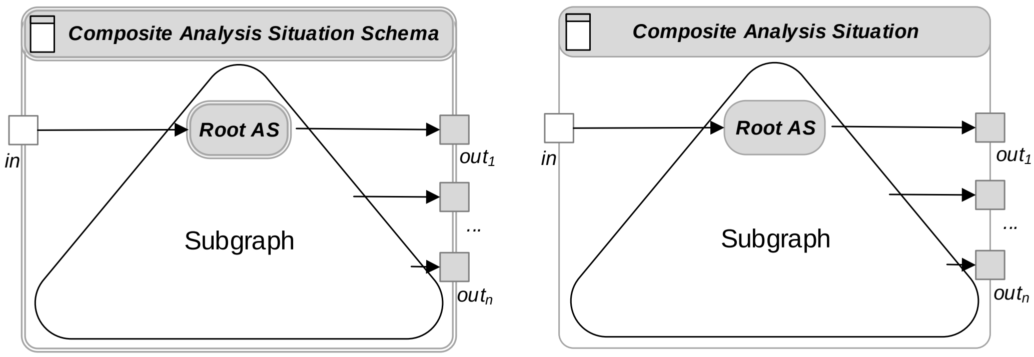


Figure 5.4: Composite analysis situation (schema)

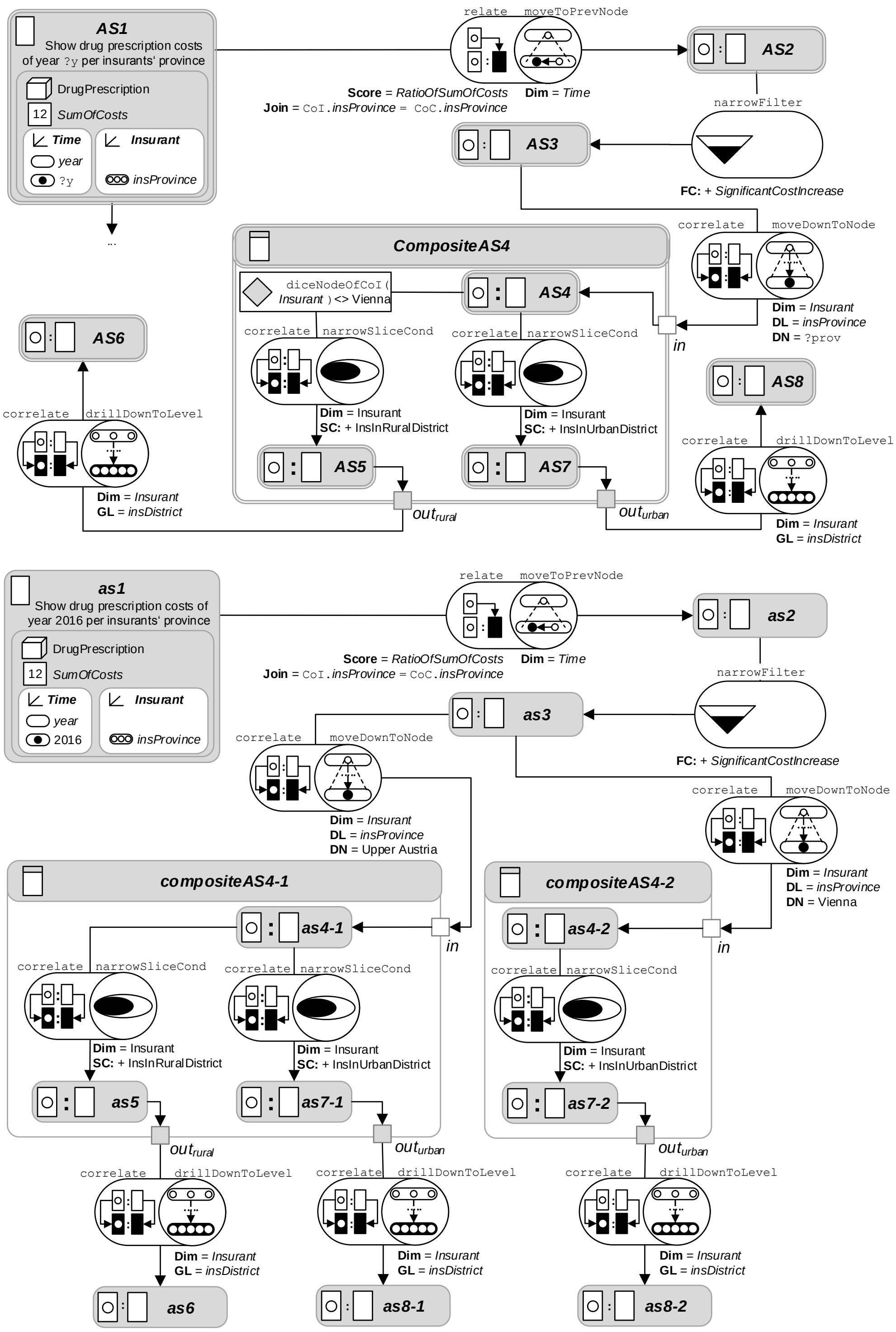


Figure 5.5: Composite analysis situation (schema) — running example

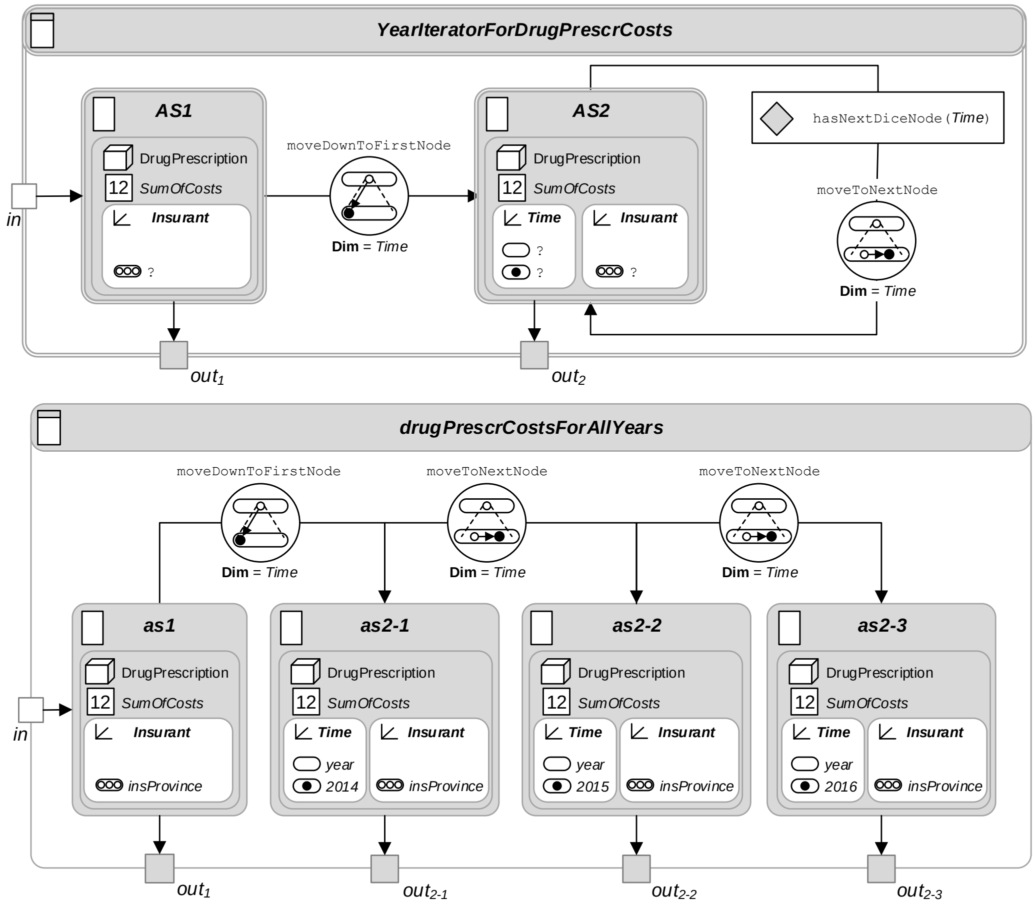


Figure 5.6: Composite analysis situation (schema) example (navigation pattern)

List of Figures

List of Tables