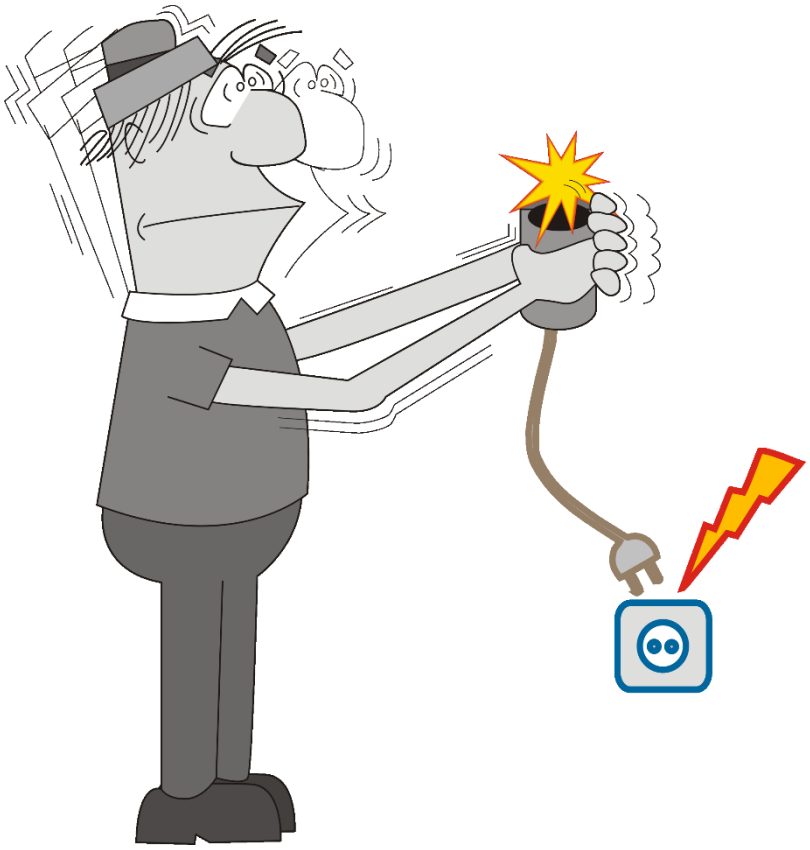
SPOTŘEBA

Databázové zpracování odečtů elektroměru a jiných potměšilých přístrojů



Rev.: 04 / 2024
Aut.: Hom50

Obsah

1 Anotace

Tento text je primárně zamýšlen pro procvičování ve výuce předmětu „Databáze“, část „Dotazovací jazyk SQL“. Předpokládá znalost logiky relačních databází a mírnou znalost konstrukcí SQL, zvláště příkazu „select“. Text podává výklad řešení step by step, proto nemá smysl některé části přeskakovat. Protože se má za to, že čtenář disponuje fakt jen základy SQL, jsou v jednotlivých krocích použity konstrukce a obraty nepříliš často používané nebo rozšiřující běžnou výuku. Proto jsou do textu vložena upozornění s event. náměty na promyšlení.

Upozornění jsou označena ikonou 

2 Popis situace

Jednou z největších částek ve výdajích domácností tvoří platby za energie a podobné komodity. Pro udržení alespoň nějaké vyrovnanosti rodinného rozpočtu je vhodné se vyvarovat jakéhokoliv jednorázového většího výdaje - alespoň tam, kde to lze předpokládat. Právě u energií lze náklady rozumně odhadnout a podle toho si nastavit měsíční zálohy. Pak při ročním vyúčtování nemusí dojít k nemilému šoku, ale k příjemnému doplnění peněženky.

To ovšem předpokládá povědomí o skutečné spotřebě té které komodity (a není k tomu zapotřebí ani umělá inteligence, stačí zbytky té vlastní). Pro účely výuky databázových aplikací ukažme řešení jedné úzce zaměřené a přitom praktické úlohy: přehled o skutečných (nejen) měsíčních spotřebách. Pro názornost zvolme jako komoditu např. elektřinu, pro jinou lze postup aplikovat analogicky.

Je zřejmé, že bez průběžných odečtů měřidla se neobejdeme. Čím více jich bude k disposici, tím přesnější a detailnější mohou být získané výstupy. Nelze však běhat třikrát denně k elektroměru a rovněž nelze stanovit přesnou denní dobu, kdy k odečtům bude docházet. Na druhé straně, mít pouze tři odečty za rok by bylo evidentně nedostačující. Z praktického hlediska bychom snad zvládli tak jeden, někdy tři odečty týdně, a to „kdy si vzpomenu“. Datumy resp. časy odečtů budou tedy stochastické (náhodné).

Cílíme ke stanovení měsíčních záloh a tedy k měsíčním spotřebám. Obecně je elementární přesně známá spotřeba rovna rozdílu dvou po sobě následujících odečtů. V průběhu roku jsou však různé měsíce energeticky různě náročné. Čím více měsíců bude odečty pokryto, tím lépe. Je zřejmé, že toto množství dat už závisí jen na nás, počítači nebo jinému IT prostředku je to při kapacitách dnešních medií celkem jedno. Stejně tak je do jisté míry jedno, jak složité operace s těmi daty budou provádět. V případě našeho úspěchu s měsíčními (kvartálními, ročními) odhady můžeme zkusit naopak detailnější informace (třeba jak je to s naší spotřebou v jednotlivých dnech týdne: jak spotřebováváme v pondělky, úterky atd).

Níže popsané řešení umožňuje získávání informací právě nejen k delším intervalům (měsíce, roky). Pokud se totiž podaří odhadnout spotřebu v jednotlivých dnech takových intervalů, potom s kumulací do týdnů, měsíc, roků už si databázový systém a hlavně jeho autor poradí hravě. Právě toto je jádrem zpracování.

3 Vstupní data

Důležitá poznámka: *Celý tento text předpokládá použití takového databázového systému, který v množině zpracovávaných datových typů obsahuje typ (např.* ***Date)*** *pro kalendářní datum resp. čas jakožto určení přesného okamžiku na časové ose. Navíc implementuje operaci odečítání* ***Datum-Datum*** *(výsledek desetinné číslo v jednotkách [den]), a dále operace* ***Datum+Číslo*** *a* ***Datum-Číslo*** *(****Číslo*** *jako desetinné číslo v jednotkách [den]). Výsledkem těchto dvou operací je hodnota typu* ***Date*** *o tolik dní po (nebo před) levým operandem* ***Datum****, kolik činí hodnota pravého operandu* ***Číslo****. Je vhodné mít stále na paměti, že jednotkou [den] desetinného čísla je dáno to, že např. hodnota 1/(24*60) = 1/1440 = 0.000694444 znamená pro datový typ* ***Date*** *jednu minutu.*

Databázové systémy tohoto typu mívají (ale nemusí mít) hodnoty typu ***Date*** *interně přímo uloženy jako desetinné číslo, jehož celou část lze chápat jako pořadové číslo daného dne a desetinnou část jako určení časového okamžiku v tomto dni (nula = 0 = půlnoc = začátek dne; 0,5 = 1/2 = poledne). Celkově tedy hodnota typu* ***Date*** *určuje polohu konkrétního okamžiku na časové ose, jejíž počátek (=0) zvolili autoři databázového systému nebo ho přejali z nadřazeného operačního systému.*

V operačních systémech Microsoftu jde tedy o stejný mechanismus, který má implementovaný např. program Excel a další z Microsoft Office. Pokud jiné databázové systémy používají rozdílný způsob práce s datumem a časem, většinou mívají k disposici funkci typu DateSerial pro převod datumu a času na desetinné číslo zpracovatelné ve shora zmíněné logice.

3.1 Odečty

Především tedy budou zpracovány jednotlivé odečty. Z logiky věci vyplývá, že minimálně bude zapotřebí zadat

- **Okamžik odečtu**. Rozhodně tedy kalendářní datum. Databázový typ **Date** (viz předchozí Poznámku) bývá ve většině databázových systémů zpracováván jako **Date and Time**. Jedna hodnota tohoto typu obsahuje tedy jak kalendářní datum, tak čas. Je-li při zadávání (ať „ručním“ nebo programem) zadáno jen kalendářní datum, je doplněna časová část jako 00:00:00, tedy jako „počáteční“ půlnoc tohoto dne.
- **Hodnota** zobrazená měřidlem. Měřidla určitě poskytují údaj o celých jednotkách měřené veličiny ([kWh], [m^3] apod. Mnohé typy měřidel jdou do větších přesností různým počtem desetinných míst, většinou tří ([Wh], [dm^3]=[l]), ale např. vodoměry i čtyř ([dl]). Je škoda, že právě novější elektroměry žádné desetinné místo nezobrazují (nemůžeme tedy s dostatečnou přesností zjistit, kolik elektrické energie je zapotřebí např. na jeden prací cyklus pračky). Na druhé straně pro naše účely při naší předpokládané frekvenci odečtů postačuje pouhá celá část hodnoty. I roční vyúčtování dodavatelů obsahují jen celočíselné hodnoty.
- Volitelně: Pokud budujeme komplexnější informační systém tohoto druhu, určitě budeme sledovat nejen elektrickou energii, ale např. i spotřebu plynu a další. K okamžiku odečtu a hodnotě měřidla připojíme jako třetí vstupní hodnotu typ komodity, např. E pro elektřinu, P pro plyn - to mě napadlo jako první, lze samozřejmě i jinak.

Upozornění: *Sloupec TYP se kromě poslední kapitoly nebude využívat.*

Když už ale budujeme informační systém, je lepší zadávat přesnější data, zvláště jsme-li databázově hraví a vymyslíme logiku řešení tak jak je popsána níže. Kalendářní datum zadáme včetně časové části (z praktického hlediska by byly zcela zbytečné vteřiny! Úplně stačí přesnost na pěti- nebo deseti-minuty), údaje o stavu měřidla na počet desetinných míst dle úvahy shora.

Tabulka (vtipně nazvaná ODEČTY) průběžně doplňovaná o další záznamy může mít např. následující strukturu:

ODEČTY		
DATUM	**TYP**	**STAV**
...	...	...
30.09.2023 12:20	E	17 830
03.10.2023 15:05	E	17 848
10.10.2023 16:10	E	17 890
...	...	...

3.2 Ceny

Kromě množství odebrané komodity je její cena druhým nutným vstupním údajem (pro rodinný rozpočet většinou tím hlavním) pro odhad měsíčních záloh.. Realitou dnešních dnů je její totální astabilita; nevíme dne ni hodiny, kdy se změní. Bohužel o její aktuální výši nebo změně se nelze dozvědět z tak pochybných informačních zdrojů, jako je rádio, televize - a už vůbec ne obecně internet. Nezbývá než kontaktovat našeho dodavatele ať už přímo, nebo udělat výjimku - nahlédnout do jeho webových stránek a doufat, že jsme skutečně na stránkách dodavatele a ne na stránkách šikovného podvodníka.

> ***Poznámka:*** *Zjištění aplikovatelné ceny většiny komodit od většiny dodavatelů vyžaduje předchozí důkladné, nejméně měsíční studium v oblasti ekonomické a hlavně matematické. Na tato studia zde čtenáře odkazujeme. Protože tento náš text se týká databází, budeme pracovat se zjednodušeným (ale pro nastavení záloh reálně celkem dobře použitelným) modelem:*
>
> *Celková platba = platba za odebrané množství + měsíční paušály*

Vodítkem může být poslední dodavatelem vystavená faktura a v ní detaily za poslední účtované období. Každé takové období začíná většinou dnem, kdy dodavatel změnil nějakou komponentu ceny. Pro reálný odhad výše záloh je nutno mít vstupní data členěná v předchozí poznámce zmíněným způsobem. Tuto tabulku dat nazvěme např. CENÍK, její část může být následující:

CENIK				
TYP	**Plati Od**	**Plati Do**	**Jednotkova Cena**	**Mesicni Pausal**
...	...	...	...	...
E	01.09.2023	30.09.2023 23:59:59	9,52	320
E	01.10.2023	31.10.2023 23:59:59	9,52	320
E	01.11.2023	30.11.2023 23:59:59	7,65	390
E	01.12.2023	31.12.2023 23:59:59	8,10	390
E	01.01.2024	31.01.2024 23:59:59	8,30	400
...	...	...	...	...

Pozorný čtenář zřejmě usoudí, že sloupec *Platí Do* je zbytečný: hodnota v něm je rovna okamžiku těsně před následující hodnotou *Platí Od*. V tomto popisovaném tématu je použit pouze z výukových důvodů (aby těch ryze databázových problémů v jistém okamžiku pro čtenáře „nebylo trochu moc“).

3.3 Datumy

Protože tyto texty jsou určeny pro výuku základů databázového zpracování, bylo snahou jednak se maximálně vyhnout programování jako takovému, jednak poskytnout co nevětší názornost při studiu *postupu* řešení. Jak bylo řečeno v úvodu, jádrem řešení jsou odhady spotřeby až na úroveň jednotlivých dnů. Proto jako další tabulka „vstupních“ dat figuruje tabulka všech kalendářních datumů, pro které bude prováděn odhad spotřeby. Pro datumy se požaduje, aby

- byly bez časové části (přesněji aby čas byl 00:00:00, aby to byla „počáteční“ půlnoc), a
- aby byly uvedeny *všechny* dny v požadovaném intervalu, a to právě jednou.

Takovou tabulku sice lze vytvořit celkem primitivním prográmkem, ale začínající databázisti určitě raději tabulku vytvoří např. dvěma kliky v Excelu a pak ji do databáze dalšími čtyřmi kliky naimportují.

DATUMY
DATUM
1.1.2023
2.1.2023
...
31.12.2023

Tabulka může obsahovat libovolný interval dnů (nemusí začínat Novým rokem a končit Silvestrem), který je překryt odečty - třeba od deseti dnů do deseti let. Blíže také viz odst. „Odhad spotřeby za období“. Příklad celého jednoho roku je vpravo.

Hlavním úkolem pro databázi (a pro jejího autora) pak bude na základě tabulky odečtů (nepravidelné datumy) rozšířit tabulku datumů (pravidelné datumy) o další sloupce až ke sloupci s odhadem spotřeby v daném dni (tedy od jeho počáteční půlnoci do počáteční půlnoci dne následujícího).

3.4 Data a soubory ke stažení

Ke stažení a procvičování jsou připraveny databázové soubory formátu Microsoft Access Database (přípona ACCDB). Jde o formát používaný metodou *CreateDatabase* instance objektové třídy *DbEngine* knihovny *DAO* (ACEDAO.DLL, odkazované jako *Microsoft Office ii.j Access database engine Object Library*, ii.j je verze Office instalovaná na počítači čtenáře).

Jako první ke stažení uvádíme databázový soubor odkazovaný zde. Obsahuje pouze tabulky ODEČTY, CENÍK a DATUMY ve struktuře popsané výše.

Tak jak bude v jednotlivých kapitolách postupovat výklad, budou uvedeny další odkazy na soubory s texty dotazů a tabulkami popsaných vždy v dané kapitole.

4 Odhad stavu měřidla

4.1 Odvození vzorce

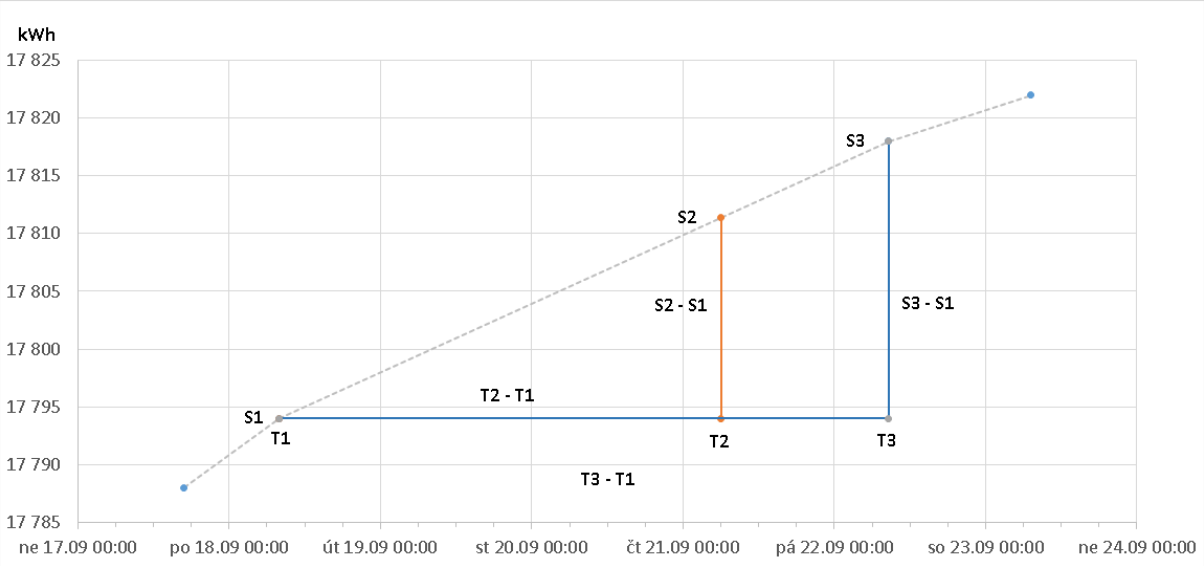
Formulace této základní úlohy je jednoduchá: Čas plyne a pro náhodné body na časové ose jsou známy hodnoty veličiny, které ona veličina v daných okamžicích nabyla (veličinou je zde stav měřidla). Odhadněte na základě těchto známých hodnot hodnotu v libovolném okamžiku, pro který hodnota známa není. Záměrně nepoukazujeme na ty partie matematiky, které se takto definovanou úlohou zabývají. Je totiž zřejmé, že žádné „matematické“ chování naší spotřeby elektřiny zhola nelze předpokládat. Kdy se nám zachce, tak rožneme. Kdy to nutně potřebujeme, zapneme pračku. Rozhodně nesvítíme podle pravidel funkce např. hyperbolický kosinus.

Jediné, co ze zaznamenaných odečtů odvodíme, je toto: mezi dvěma sousedními odečty (resp. okamžiky jejich odečtů) spotřeba rostla, klesala nebo stagnovala. Ovšem jak to bylo mezi nimi, to už nikde není podchyceno (to by pak nebyly sousední odečty v tomto smyslu). Není proto sebemenšího důvodu proč nepředpokládat, že to bylo rovnoměrně. Pak ale odhad stavu není ani tak úloha matematická, jako spíše počtářská.

Vyjděme z příkladu: v okamžicích T1 a T3 (to jsou ty okamžiky sousedních odečtů) byly zjištěny stavy měřidla S1 a S3. Odhadujeme stav S2 v okamžiku T2:

ODEČTY		
DATUM	**TYP**	**STAV**
...	...	...
T1 = 18.09.2023 7:55	E	S1 = 17 794
T3 = 22.09.2023 8:35	E	S3 = 17 818
...	...	...
T2 = **21.09.2023 6:00**	E	S2 = **???**
...	...	...

Datům odpovídá jejich grafické znázornění:



Obr. 1: K výpočtu odhadu stavu měřidla

Z podobnosti trojúhelníků plyne:

$$(S2 - S1) : (T2 - T1) = (S3 - S1) : (T3 - T1) \qquad (1)$$

tedy

$$(S2 - S1) = (T2 - T1) \times (S3 - S1) : (T3 - T1) \qquad (2)$$

z toho

$$S2 = S1 + (T2 - T1) \times (S3 - S1) : (T3 - T1) \qquad (3)$$

Pro názornost zobrazme rozdíly okamžiků formátem hh:mm. Pro vlastní ověřovací výpočet je však pohodlnější převod třebas na minuty (což lze, jeden časový rozdíl je v čitateli, jeden ve jmenovateli, takže je jedno, v jakých jednotkách budeme tento ověřovací výpočet provádět - hlavně aby byly jednotky stejné). Je tedy

$$(T3 - T1) = 96^h\ 40^m = 5800\ [min]$$
$$(T2 - T1) = 70^h\ 05^m = 4205\ [min]$$
$$(S3 - S1) = 24$$
$$S1 = 17\ 794$$

Dosazením do (3) je pak odhad stavu měřidla

$$S2 = 17\ 794 + 4\ 205 * 24 / 5\ 800 = 17\ 811.4\ [kWh]$$

což je v dobré shodě s grafickým vyjádřením. Hodnota je ve stejných jednotkách, v jakých je poskytuje měřidlo.

Protože nulou dělit nelze, je třeba ošetřit případ, kdy T3 = T1. V takovém případě by se zdálo, že není třeba nic počítat, protože evidentně S2 = S1. Pokud ale jsou data v pořádku (a to rozhodně předpokládáme), takový případ nenastane. Znamenalo by to, že pro jediný okamžik

jsou v tabulce ODEČTY dva záznamy. Jsou-li se stejnou hodnotou stavu, je jeden z nich zbytečný, jsou-li s různou hodnotou stavu, je to chyba odečitatele.

Poznámka pro matematické labužníky: *Jde o nejjednodušší případ lineární interpolace v rovině funkcí* $Y = A \times X + B$.

4.2 Odhad stavu měřidla k půlnoci daného dne

Jde o doslovnou aplikaci předchozího odstavce. Dokonce jde o okamžiky vyznačené na grafu v obr. 1 jako průsečíky svislých čar příslušejících popiskům vodorovné osy (to jsou právě půlnoci daných dnů). Každému takovému okamžiku se přiřadí hodnota odhadu stavu měřidla daná vztahem (3).

4.3 Odhad spotřeby za den

Je-li S0 odhad stavu měřidla k *počáteční* půlnoci dne D0 a S1 odhad stavu měřidla ke *koncové* půlnoci dne D0 (t. j. k počáteční půlnoci následujícího dne D1 = D0+1), je odhad spotřeby dne D0 evidentně

```
Spotřeba (D0) = S1 - S0
```

4.4 Odhad spotřeby za období

Logicky je pak odhad spotřeby za období roven součtu odhadů spotřeby za jednotlivé dny období. K určení dnů období slouží „vstupní“ tabulka DATUMY (viz výše). To pro účely výuky jednak zjednodušuje vlastní databázové řešení, jednak poskytuje smysluplné téma pro ukázku některých aspektů při konstrukci dotazů. Tabulka DATUMY bude pro jednotlivé (proto po sobě jdoucí) dny jednotlivými kroky doplněna o další („pomocné“) sloupce, v posledním kroku o kýžený sloupec SPOTŘEBA.

To klade podmínku na vztah mezi tabulkou ODEČTY a tabulkou DATUMY. Interval datumů v tabulce ODEČTY musí přesahovat interval datumů v tabulce DATUMY. Pro studium: pohledem na obr. 1 rozvažte!

5 Úloha SPOTŘEBA - verze s pod-dotazy

Závěr předchozí kapitoly: bude se rozšiřovat tabulka DATUMY o další sloupce až ke sloupci SPOTŘEBA. V každém novém sloupci se ke každému datumu přiřadí vypočtená konkrétní hodnota (přesněji odhad hodnoty). Následující odstavce popisují genezi těchto sloupců, především tedy obsah jednotlivých buněk, který bude určen konkrétním algoritmem za konkrétním účelem.

Poznámka 1: *Všude, kde bude dále uveden pojem Datum, se na tom místě rozumí určení okamžiku na časové ose jakožto kalendářního datumu plus času v tomto dni. Bude-li uvedena datumová konstanta bez časové části, rozumí se čas 00:00:00 (tj. počáteční půlnoc daného dne).*

Poznámka 2: *Příkazy SQL níže uvedené jsou ve tvaru, který je jen jedním z možných. Čtenář je může libovolně používat jako vzor pro své vlastní tvůrčí variace. V některých situacích je dokonce musí mírně upravit, a to dle syntaktických pravidel čtenářem používaného databázového systému.*

5.1 Datumy předchozích a následujících odečtů

Pro výpočet odhadu stavu měřidla k jistému okamžiku (zde k půlnoci daného dne z tabulky DATUMY) je třeba znát skutečný stav měřidla v předchozím (P) a následujícím (N) známém odečtu vzhledem k danému okamžiku. Předchozích a následujících odečtů je však mnoho, proto přesněji: „nejbližšího“ nižšího a „nejbližšího“ vyššího odečtu.

Směřujeme ke zjištění stavu odečtu (říkejme mu předchozí) s *největším nižším* datumem a stavu odečtu (říkejme mu následující) s *nejmenším vyšším* datumem vzhledem k půlnoci konkrétního dne.

Důležitá poznámka: *Skutečně se musíme opírat o datumy (nezapomenout, že datum zde užívané je včetně časové části), nikoliv o stavy měřidla. Těch může být s různými datumy několik stejných, např. když nám na tři dny překopnou kabel.*

Postupně: nejprve největší datum:

```
select max(ODECTY.DATUM) from ODECTY
```

To by však dodalo maximum datumové hodnoty ze všech odečtů. My to potřebujeme jen pro datum menší než konkrétní datum, např. 13.6.2023 v půl desáté:

```
select max(ODECTY.DATUM)
  from ODECTY
  where ODECTY.DATUM <= #06/13/2023 9:30#
```

A je nutno to udělat ne pro jediné konkrétní datum, ale pro všechny datumy v tabulce DATUMY. Pro jediné konkrétní datum z tabulky DATUMY:

```
Datum nejbližšího nižšího odečtu
select max(ODECTY.DATUM)
  from ODECTY
  where ODECTY.DATUM <= DATUMY.DATUM
```

Celkově ale nepožadujeme jedinou hodnotu datumu. Nemůže to však být jen prostá množina datumů, protože by se následně nezjistilo, jaký byl stav měřidla pro každé z těchto datumů. Minimálně to musí být množina dvojic typu

```
[ Datum aktuálního dne z tabulky DATUMY;
  Datum nejbližšího nižšího odečtu z tabulky ODEČTY ]
```

Datum přitom bude získáno z každého záznamu tabulky DATUMY dotazem typu „select DATUM from DATUMY ...“, Datum nejbližšího nižšího odečtu pak jako výsledek předchozího dotazu. Tedy celkem (pro lepší čitelnost použijme pro zdrojovou tabulku DATUMY alias D, pro zdrojovou tabulku ODECTY alias O):

```
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
select
  D.DATUM,
  (select max(O.DATUM)
   from ODECTY O
   where O.DATUM <= D.DATUM) as pDatum
from DATUMY D
order by DATUM
```

Zcela analogicky lze získat následující datumy (nDatum).

Databázové systémy se však nebrání oba kroky spojit dohromady:

```
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
select
  D.DATUM,
  (select max(O.DATUM)
   from ODECTY O
   where O.DATUM <= D.DATUM) as pDatum,
  (select min(O.DATUM)
   from ODECTY O
   where O.DATUM > D.DATUM) as nDatum
from DATUMY D
order by DATUM
```

Dotaz pojmenujme např. **DA_PredNaslDatumy**. Protože D.DATUM je bez časové části, rozumí se „počáteční půlnoc“ daného dne. Část jeho výsledku:

DA_PredNaslDatumy		
DATUM	**pDatum**	**nDatum**
...	...	...
02.10.2023	30.9.2023 12:20	3.10.2023 15:05
03.10.2023	30.9.2023 12:20	3.10.2023 15:05
04.10.2023	3.10.2023 15:05	10.10.2023 16:10
05.10.2023	3.10.2023 15:05	10.10.2023 16:10
06.10.2023	3.10.2023 15:05	10.10.2023 16:10

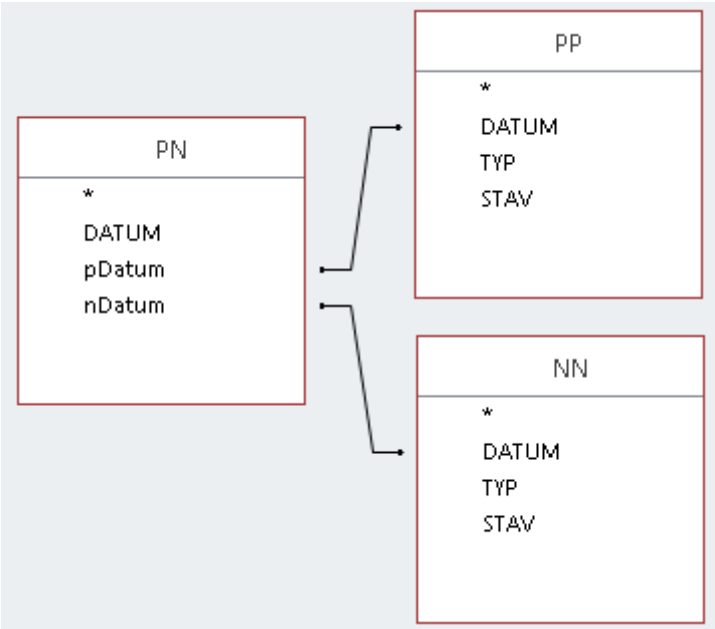
DA_PredNaslDatumy		
DATUM	**pDatum**	**nDatum**
...	...	...


Pro výuku základů databází: Výrazem v seznamu výrazů SQL příkazu *select* může tedy být i pod-dotaz odevzdávající jednu hodnotu.

5.2 Předchozí a následující odečty

Pro řešení problému spotřeby dále: Je sice výborné, že známe hodnoty datumů nejbližších odečtů, ale pro odhad stavu měřidla k (počáteční) půlnoci každého dne podle vztahů uvedených shora je třeba dále znát stav měřidla v okamžik odečtu těchto nejbližších odečtů. Jinak řečeno, je dále zapotřebí rozšířit výsledek dotazu DA_PredNaslDatumy o další dva sloupce, označené např. pStav (předchozí stav) a nStav (následující stav).

Tedy: z výsledku dotazu DA_PredNaslDatumy (alias PN) potřebujeme dvakrát sáhnout do Odečtů (jednou alias PP, jednou alias NN) pro řádek se stejným datumem (PP.DATUM resp. NN.DATUM), jako má aktuálně zpracovávaný řádek PředchozíchNásledujícíchDatumů (tj. PN.pDatum resp. PN.nDatum):



Obr. 2: Propojení datových zdrojů

K tomuto „sáhnutí“ lze využít možnosti připojení resp. propojení (*join*) jednoho datového zdroje s jiným. Při procházení výsledku dotazu PN se propojí jeho aktuální řádek na ten řádek zdroje PP, který má hodnotu PP.DATUM rovnající se hodnotě PN.pDatum. Analogicky pro následující odečty a pole PN.nDatum a NN.DATUM. Toto připojení resp. propojení zajišťuje v příkazu *select* klauzule (*left* resp. *right* resp. *inner*) *join*:

```
AktuálníDatum (aDatum) = Aktuálně zpracovávané datum zdroje D1
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
select
  PN.DATUM AS aDatum,
  PP.DATUM AS pDatum,
  PP.STAV AS pStav,
  NN.DATUM AS nDatum,
  NN.STAV AS nStav
from
 (
  DA_PredNaslDatumy PN inner join ODECTY PP
    on PN.pDatum = PP.DATUM
 )
  inner join ODECTY NN
    on PN.nDatum = NN.DATUM
```

 Pro výuku základů databází: Klíčové slovo *from* uvozuje popis - v tomto případě jediného - datového zdroje. V kulatých závorkách (a) je nejprve uvedeno propojení výsledku dotazu PN s tabulkou ODEČTY (poprvé; alias PP, pro autora zdroj předchozích odečtů). Tím vznikne „nový“ datový zdroj, ze kterého je požadováno propojení s tabulkou ODEČTY (podruhé; alias NN, pro autora zdroj následujících odečtů). Tím vznikne „finální“ datový zdroj, ze kterého se čerpají hodnoty do seznamu výrazů příkazu *select*. Také jinak: celý obr. 2 tvoří jediný datový zdroj. V příkazu *select*, který ho klauzulí *from* určí za datový zdroj, je možno používat kterékoliv jeho datové pole. V případě nejednoznačnosti při jejich určení je nutno použít obvyklý selektor, tedy např. PP.STAV nebo NN.STAV.

Pro řešení problému spotřeby dále: Dotaz pojmenujme např. **DB_PredNaslOdecty**. Část jeho výsledku:

DB_PredNaslOdecty				
aDatum	**pDatum**	**pStav**	**nDatum**	**nStav**
...	...	...	...	...
02.10.2023	30.9.2023 12:20	17 830	3.10.2023 15:05	17 848
03.10.2023	30.9.2023 12:20	17 830	3.10.2023 15:05	17 848
04.10.2023	3.10.2023 15:05	17 848	10.10.2023 16:10	17 890
05.10.2023	3.10.2023 15:05	17 848	10.10.2023 16:10	17 890
06.10.2023	3.10.2023 15:05	17 848	10.10.2023 16:10	17 890

DB_PredNaslOdecty				
aDatum	**pDatum**	**pStav**	**nDatum**	**nStav**
...	...	...	...	...

5.3 Odhady stavů k půlnoci

Nyní odhady stavu k půlnoci každého dne podle vztahu (3):

```
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
PůlnočníStav (mStav) = Odhad stavu k půlnoci (midnight) dne aDatum
select
  aDatum, pDatum, pStav, nDatum, nStav,
  pStav+(aDatum-pDatum)*(nStav-pStav)/(nDatum-pDatum) AS mStav
from DB_PredNaslOdecty
```

Dotaz pojmenujme např. **DC_PulnocniStavy**. Část jeho výsledku:

DC_PulnocniStavy					
aDatum	**pDatum**	**pStav**	**nDatum**	**nStav**	**mStav**
...	...	...	...	...	...
02.10.2023	30.9.2023 12:20	17 830	3.10.2023 15:05	17 848	17 838.589
03.10.2023	30.9.2023 12:20	17 830	3.10.2023 15:05	17 848	17 844.368
04.10.2023	3.10.2023 15:05	17 848	10.10.2023 16:10	17 890	17 850.215
05.10.2023	3.10.2023 15:05	17 848	10.10.2023 16:10	17 890	17 856.176
06.10.2023	3.10.2023 15:05	17 848	10.10.2023 16:10	17 890	17 862.138
...	...	...	...	...	...

 Pro výuku základů databází - náměty k rozvaze a případnou realizaci:

1. Příkaz *select* realizující dotaz DC_PulnocniStavy pracuje pouze s datovými poli určenými (jedním) předchozím dotazem DB_PredNaslOdecty. Nešlo by půlnoční stavy nechat vypočítat už tam?

2. Předchozí a následující odečty jsou použity naposledy v dotazu DB_PredNaslOdecty. Musí vůbec výsledek dotazu DC_PulnocniStavy poskytovat i předchozí a následující odečty? Nestačí jen aktuální datumy z tabulky DATUMY a k nim vypočtený odhad půlnočních stavů?
3. Počítá nám to vůbec dobře? Zkontrolujte alespoň na první pohled. Nápověda: Ta první hrubá kontrola by mohla spočívat v ověření, že půlnoční stav není menší než předchozí a zároveň není větší než následující - prostě leží mezi nimi. Upravte mírně dotaz DC_PulnocniStavy.

5.4 Spotřeba za celý den

To už je jednoduché. Bude vytvořen dotaz, který bude procházet výsledky předchozího dotazu DC. V každém řádku pak odečte stav k (počáteční) půlnoci aktuálního dne (mStav) od stavu k (počáteční) půlnoci následujícího dne. A to je kýžená spotřeba aktuálního dne.


Pro výuku základů databází: Škodolibý vyučující by v tomto okamžiku řekl: Teď už jen realizujte!

Pro řešení problému spotřeby: Škodolibost takového vyučujícího by spočívala v tom, že při provádění jediného příkazu *select* (který si interně posouvá „ukazovátko“ = pointer na aktuální řádek) mu nelze zevně posunout toto ukazovátko o jeden záznam, uchopit žádanou hodnotu a vrátit ukazovátko zpět. Přesněji řečeno: nelze to jinak než opět prostředky dotazovacího jazyka SQL. Tam však konstrukce typu *go forward* - zjisti data - *go back* není. A protože tento text striktně drží čtenáře co nejdál od programování, zbývají několikrát zmíněné pod-dotazy.

Ve výuce byla studentům prezentována úloha „Spotřeba benzinu“, kde se vyskytl stejný problém: získat z tabulky tankování k údaji o [km] aktuálního tankování údaj o [km] předchozího tankování - a tyto [km] jsou v předchozím řádku téže tabulky tankování. Logiku toho, co jsme dovodili ve výuce o získání předchozích [km], jsme mimochodem použili i tého úloze o spotřebě elektřiny, viz odstavec „Datum předchozího odečtu“ a v něm „největší z menších“. V něm to zdánlivé couvnutí ukazovátka zajistil poddotaz.

V popisovaném problému spotřeby elektřiny bychom nechali zpracovat nejmenší z větších datumů. Z hlediska výuky by to nepřineslo nic nového, zkusme něco na zcela jiném principu. Využijí se skutečnosti platné v tomto řešeném problému a tomto databázovém systému (v jiných problémech a jiných databázových systémech je jejich platnost vhodné ověřit):

- V tomto problému: Pracuje se časovými elementy rovnými jednomu dni, a tyto elementy tvoří rostoucí řadu s diferencí jeden den.
- V tomto databázovém systému: Jednotkou času, která je použita pro datový typ *date and time*, je jeden den = 24 hodin. Existuje operace + vyhodnocující výraz datum+1 jako okamžik o 24 hodin dále v toku času oproti hodnotě datum.
- V tomto databázovém systému: V klauzuli *from* příkazu *select* lze použít zápis A *inner join* B *on* <Výraz> - viz „Poznámka k příkazu *select*“ níže v Přílohách.

Výsledný dotaz je pak skutečně triviální, označme ho např. **DD_Spotreby**:

```
Alias S = Současné datum a stav
Alias N = Následující datum a stav
select
  S.aDatum as aDatum,
  N.mStav-S.mStav as Spotreba
from
  DC_PulnocniStavy S inner join DC_PulnocniStavy N
    on N.aDatum = S.aDatum+1
order by S.aDatum
```

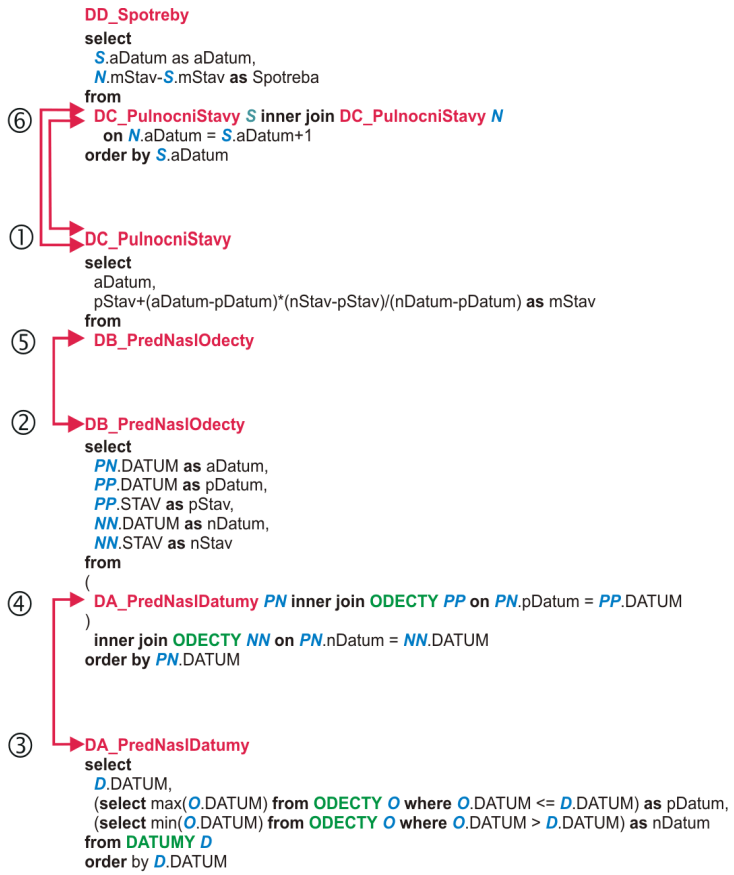
s částí výsledku

DD_Spotreby	
aDatum	**Spotreba**
...	...
02.10.2023	5,779
03.10.2023	5,847
04.10.2023	5,962
05.10.2023	5,962
06.10.2023	5,962
...	...


Pro výuku základů databází: Ve dnech 4/10, 5/10 a 6/10 je spotřeba stejná. Není to chyba?

5.5 Řetězení volání

V této fázi řešení zajistí odhad spotřeby v jednotlivých dnech připraveného intervalu datumů následující posloupnost volání pod-dotazů (spouštěných „vrcholovým“ dotazem DD_Spotřeby):



Logicky sice správně, ale češtinářsky dost nesrozumitelně:

- Na začátku provádění (Database.Execute) potřebuje vrcholově spouštěný dotaz DD čerpat z datového zdroje, který je výsledkem dotazu DC. Proto jako krok označený ① DD volá DC jako poddotaz a čeká, až mu připraví svůj výsledek = recordset DC. K tomu ovšem DC potřebuje provést krok ② - čerpá totiž z výsledku DB. DC tedy přeruší svou činnost a čeká, až mu DB připraví recordset DB. DB ale k tomu potřebuje nejprve získat výsledek DA. Proto do třetice dojde k přerušení činnosti pod-dotazu, tentokrát pro krok ③ = vykonání DA. Poté bude k disposici recordset DA, který v kroku ④ bude předán do DB. DB bude schopen pokračovat ve své činnosti, tj. přípravě datového zdroje pro DC; jeho zpřístupnění je obsahem kroku ⑤. Načež se krokem ⑥ konečně dokončí DD a je možno dále zpracovávat denní spotřeby. Uff !

Nebo ještě jinak (co je srozumitelnější, nechť čtenář posoudí sám):

- Dotaz DD volá 2x pod-dotazy DC.
- Pod-Dotaz DC volá pod-pod-dotaz DB.
- Pod-Pod-Dotaz DB volá pod-pod-pod-dotaz DA a navíc 2x otevírá pro samostatné zpracování tabulku dat ODEČTY.
- Pod-Pod-Pod-Dotaz DA volá dva pod-pod-pod-pod dotazy čerpající každý samostatně z tabulky dat ODEČTY, sám čerpá z tabulky dat DATUMY.

Už při výukové databázi mající cca 100 datumů a 40 odečtů lze počet nutně zpracovatelných kombinací odečtů a datumů odhadnout na miliony. Sice je možno předpokládat vtipné využití nejrůznějších nástrojů typu indexové a jiné pomocné tabulky, ale přesto se vnucuje otázka:

Jak to databázový systém zvládne?

5.6 Ojj! Problém ...

Nezvládne!

To je odpověď na otázku předchozího odstavce. Ovšem upřesnění: Jak který databázový systém. Pro účely tohoto výukového textu - a pro účely primární výuky databází v prostředí MS OFFICE, tj. MS Jet Database a MS Access - byly texty, příklady i databáze vytvářeny v něm. Finální vrcholový dotaz (tj. DD) v něm tento systém nezvládá jako celek z časových důvodů. Krokováním bylo odhadnuto trvání při shora použitém objemu dat na cca 17 hodin. Rozhodně by bylo zajímavé předložit toto konkrétní řešení se stejnými daty jinému databázovému systému.


Pro výuku základů databází: Má-li tento text být studijním materiálem, musí mít studenti možnost vlastní realizace popisovaných konstrukcí s (téměř) okamžitou reakcí databázového systému. Pokud to byť jen v jediném databázovém systému je nereálné, je třeba buď řešení upravit, nebo řešit principiálně jinak.

5.7 Ke stažení

Databázi obsahující zdrojová data a dotazy popsané v této kapitole až do tohoto místa lze stáhnout zde.

6 Úloha SPOTŘEBA - verze s tabulkami

Pro vyřešení výše naznačeného problému zkusme nejprve úpravu už vytvořeného. V předchozím popisovaném řešení je celkem zřejmé, že časovou náročnost způsobuje násobné řetězení volání pod-dotazů. Pokud by některé (případně všechny) čerpaly nikoliv z pod-dotazů, ale z fyzických tabulek, mohlo by to znamenat podstatnou časovou úsporu. Úprava na takovou logiku řešení je více méně mechanická:

- Dřívější Pod-Pod-Pod-Dotaz DA bude samostatným dotazem DA, který uloží svůj výsledek do tabulky; označme-ji např. TA.
- Dřívější Pod-Pod-Dotaz DB bude samostatným dotazem DB, který čerpá nikoliv přímo z dotazu DA, ale z tabulky TA dotazem DA vytvořené. Tento samostatný dotaz DB analogicky uloží svůj výsledek do tabulky; označme-ji např. TB.

- Dřívější Pod-Dotaz DC bude samostatným dotazem DC, který čerpá z tabulky TB, a který uloží svůj výsledek do tabulky; označme-ji např. TC.
- Finální dotaz DD zůstane samostatným dotazem DD, ale bude čerpat z tabulky TC. Ať to zůstane konzistentní, může i on uložit svůj výsledek do tabulky, označme ji např. TD.


Pro výuku základů databází: Připomínáme tvar příkazu *select* pro vytvoření tabulky ze zdroje:

```
select SEZNAM into TABULKA from ZDROJ
```

přičemž ZDROJ může být i v kulatých závorkách uzavřený text příkazu *select* (tedy pod-dotaz).

Půjde jen o doslovnou realizaci závěru předchozí kapitoly. Pro snazší orientaci jsou ponechány názvy tabulek stejné jako názvy pod-dotazů, pouze „předpona“ D (dotaz) byla nahrazena „předponou“ T (tabulka). U příkazů *select* s klauzulí *into* generujících tabulky byla použita „předpona“ G.

6.1 Datumy předchozích a následujících odečtů

Původní Pod-Pod-Pod-Dotaz DA:

```
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
select
  D.DATUM,
  (select max(O.DATUM)
   from ODECTY O
   where O.DATUM <= D.DATUM) as pDatum,
  (select min(O.DATUM)
   from ODECTY O
   where O.DATUM > D.DATUM) as nDatum
from DATUMY D
order by DATUM
```

Příkaz GA_PredNaslDatumy vytvářející tabulku TA:

```
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
select * into TA_PredNaslDatumy
 from
```

```
(
  select
    D.DATUM,
    (select max(O.DATUM)
     from ODECTY O
     where O.DATUM <= D.DATUM) as pDatum,
    (select min(O.DATUM)
     from ODECTY O
     where O.DATUM > D.DATUM) as nDatum
  from DATUMY D
  order by DATUM
)
```

6.2 Předchozí a následující odečty

Původní Pod-Pod-Dotaz DB:

```
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
select
  PN.DATUM AS aDatum,
  PP.DATUM AS pDatum,
  PP.STAV AS pStav,
  NN.DATUM AS nDatum,
  NN.STAV AS nStav
from
 (
  DA_PredNaslDatumy PN inner join ODECTY PP
    on PN.pDatum = PP.DATUM
 )
  inner join ODECTY NN
    on PN.nDatum = NN.DATUM
```

Příkaz GB_PredNaslOdecty vytvářející tabulku TB:

```
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
```

```
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
select * into TB_PredNaslOdecty
 from
  (
    select
      PN.DATUM AS aDatum,
      PP.DATUM AS pDatum,
      PP.STAV AS pStav,
      NN.DATUM AS nDatum,
      NN.STAV AS nStav
    from
     (
      TA_PredNaslDatumy PN inner join ODECTY PP
        on PN.pDatum = PP.DATUM
     )
      inner join ODECTY NN
        on PN.nDatum = NN.DATUM
  )
```

6.3 Odhady stavů k půlnoci

Původní Pod-dotaz DC:

```
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
PůlnočníStav (mStav) = Odhad stavu k půlnoci (midnight) dne aDatum
select
  aDatum, pDatum, pStav, nDatum, nStav,
  pStav+(aDatum-pDatum)*(nStav-pStav)/(nDatum-pDatum) AS mStav
from DB_PredNaslOdecty
```

Příkaz GC_PulnocniStavy vytvářející tabulku TC:

```
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
```

```
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
PůlnočníStav (mStav) = Odhad stavu k půlnoci (midnight) dne aDatum
select * into TC_PulnocniStavy
 from
 (
  select
    aDatum, pDatum, pStav, nDatum, nStav,
    pStav+(aDatum-pDatum)*(nStav-pStav)/(nDatum-pDatum) AS mStav
  from TB_PredNaslOdecty
 )
```

6.4 Spotřeba za celý den

Původní Dotaz DD:

```
Alias S = Současné datum a stav
Alias N = Následující datum a stav
select
  S.aDatum as aDatum,
  N.mStav-S.mStav as Spotreba
from
  DC_PulnocniStavy S inner join DC_PulnocniStavy N
    on N.aDatum = S.aDatum+1
order by S.aDatum
```

Příkaz GD_Spotreby vytvářející tabulku TD:

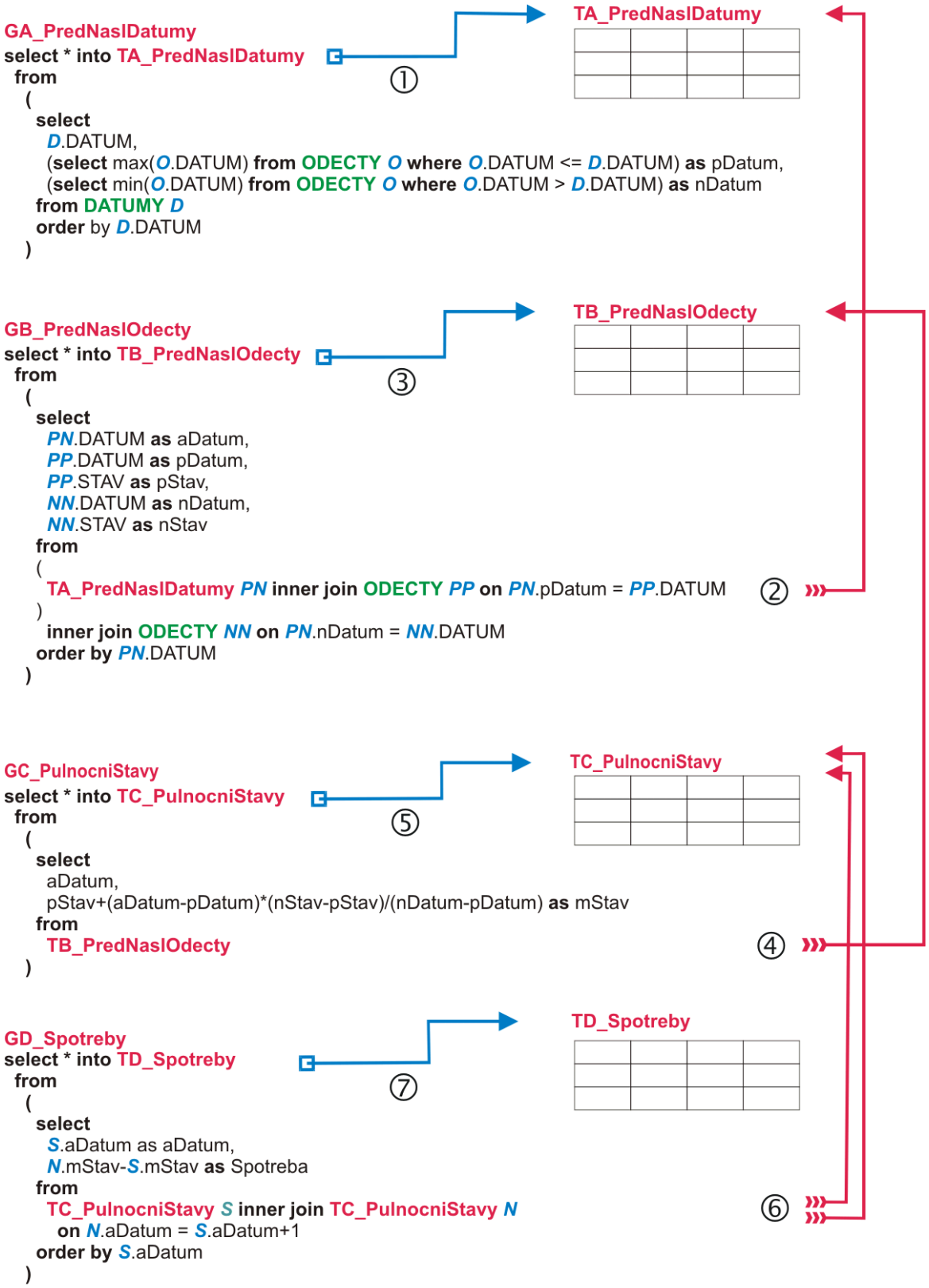
```
Alias S = Současné datum a stav
Alias N = Následující datum a stav
select * into TD_Spotreby
 from
  (
    select
      S.aDatum as aDatum,
      N.mStav-S.mStav as Spotreba
    from
      TC_PulnocniStavy S inner join TC_PulnocniStavy N
```

```
        on N.aDatum = S.aDatum+1
    order by S.aDatum
)
```

6.5 Posloupnost volání

Ve verzi s pod-dotazy byly jednotlivé kroky volány jaksi automaticky každým dalším vnořeným pod-dotazem. Uživatel v prostředí použitého databázového programu (nebo aplikace či jiného nástroje - říkejme mu také Uživatel) spouštěl jediný objekt, a tím byl dotaz na oné „vrcholové“ úrovni; pak už to „běželo samo“. Na rozdíl od verze s pod-dotazy jde ve verzi s tabulkami sice o řetězení, ale činností (jednotlivých kroků) uživatele. Tu posloupnost jednotlivých kroků je pak nutno dodržet.


Pro výuku základů databází: To může být pro uživatele, který je fyzickou osobou, nejen intelektuálně náročné, ale pro budování komplexnějších informačních systémů bez programování resp. analogických mechanismů až nepoužitelné. Z výukových důvodů je však nutno tento postup jednak uvést, jednak pochopit. Modelový příklad s odečty měřidel je k tomu celkem vhodný.



Symbolika v tomto schématu je analogická jako v případě řešení s pod-dotazy výše. Protože jde o jednodušší a hlavně přímočařejší logiku, neuvádíme slovní popis (jako shora), ale pro studium doporučujeme jeho vlastní vytvoření.

6.6 Ke stažení

Databázi obsahující zdrojová data a dotazy popsané v této kapitole až do tohoto místa lze stáhnout zde.

7 K řešení s tabulkami

7.1 Diskuse

Zatímco doba vykonávání úlohy s pod-dotazy se počítá odhadem na hodiny, celková doba řešení s tabulkami trvá méně než 0.5 [sec]. Je to ale za cenu jistého dis-komfortu. Jde o čtyři „vytvářecí dotazy“, které neodevzdávají na rozdíl od příkazu *select* bez části *into* množinu záznamů. Vyžadují samostatně spouštění, a to v přesně určené posloupnosti. Jak bylo naznačeno výše, je tento postup určen pro občasné jednorázové získání výsledků (fyzickým uživatelem), ale zejména pro začlenění do komplexního programového systému (čemuž se pro účely výuky zatím důsledně vyhýbáme).

Zatímní výsledky - s pod-dotazy a čtyřmi tabulkami - nabízí prostředí připravené k pokusům.

Intuitivně jsme usoudili, že časová náročnost v případě pod-dotazů je dána jejich řetězením. Tak jsme to čtyř-úrovňové řetězení odstranili přístupem do čtyř fyzických tabulek - tj. nahradili jsme *všechny* pod-dotazy.

Protože jsme ale pečliví (přesněji nejsme hrr), tak jsme při řešení pomocí pod-dotazů postupovali přesně dle kapitoly shora (viz). Vytvořili jsme text dotazu DA, uložili a hned vyzkoušeli, jak a jestli vůbec funguje, a jestli jsou výsledky správné. Postupně s dalšími dotazy.

Určitě jsme si všimli, že dotaz DA byl vykonaný téměř okamžitě. Dotaz DB téměř taky - přesněji nepozorovali jsme znatelné prodloužení (DB volá DA!). Pokud není náš počítač superpočítačové dělo, pak jsme si možná - minimálně při prvním spuštění DC - všimli velmi malé časové prodlevy, která u DA a DB patrná nebyla: DC volá DB a ten zas DA. Při spuštění DD už jsme se v reálném čase výsledku nedočkali. Tak jsme nahradili dotazy tabulkami a tím se to vyřešilo.

Nyní úvaha a pokus číslo 1: Když DA, DB i DC pracují při řetězení uspokojivě, nestačilo by do fyzické tabulky jakožto zdroje přesměrovat jen ten poslední DD? Tedy DA i DB nechat tak jak jsou, DC upravit na „vytvářecí“ dotaz“ = příkaz GC pro tabulku TC, a dotazu DD určit jako zdroj tuto tabulku TC?

Příkaz GC_PulnocniStavy vytvářející tabulku TC:

```
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
PůlnočníStav (mStav) = Odhad k půlnoci (midnight) dne aDatum
```

```
select * into TC_PulnocniStavy
 from
  (
   select
     aDatum, pDatum, pStav, nDatum, nStav,
     pStav+(aDatum-pDatum)*(nStav-pStav)/(nDatum-pDatum) AS mStav
   from DB_PredNaslOdecty
  )
```

Upravený dotaz DD_Spotreba čerpající z tabulky TC:

```
Alias S = Současné datum a stav
Alias N = Následující datum a stav
select
  S.aDatum as aDatum,
  N.mStav-S.mStav as Spotreba
from
  TC_PulnocniStavy S inner join TC_PulnocniStavy N
    on N.aDatum = S.aDatum+1
order by S.aDatum
```

Nyní průzkum výsledku:

- Spuštěním „vytvářecího dotazu“ = příkazu GC by měla být vytvořena tabulka TC. Ověření: Je vytvořena!
- Lze tedy spustit dotaz DD, který z TC čerpá. Ověření: Nejen že vidíme výsledky, ale vidíme je téměř okamžitě!!
- Závěr: Úspěch - alespoň částečný. Místo čtyř samostatných kroků v případě čtyř tabulek stačí jen dva.

7.2 Ke stažení

Databázi obsahující zdrojová data a dotazy popsané v této kapitole až do tohoto místa lze stáhnout zde.

7.3 Námět k procvičování

Pokud trváme na řešení s fyzickými tabulkami a s propojením (join) pomocní DATUM a DATUM+1, víc kroků asi neušetříme. Pro procvičování však námět k pokusu č. 2:

- V předchozí části jsme do fyzické tabulky nasměrovali poslední „finální“ dotaz DD. Ten čerpá z (jediné) tabulky vytvořené předchozím DC.
- Předchozí posloupnost včetně řetězení DA, DB a DC zůstala víceméně zachována.
- Co kdybychom nechali vytvořit tu (jedinou) tabulku hned prvním (DA), a ponechali naopak původní posloupnost včetně řetězení DB, DC a DD - s tím, že jen nasměrujeme DB do tabulky vytvořené DA?

Ten, kdo se námětu pilně chopí a skutečně úpravu provede, uvidí to sám. Pro ostatní: I v tomto případě celkem překvapivě vše funguje poměrně svižně jako v předchozí verzi s jednou tabulkou „na konci“. Stejně jako v předchozí verzi jde o řešení se dvěmi samostatnými kroky.

Poznámka: Trpělivý studující vyzkouší i poslední variantu s jednou tabulkou (vytvoří ji druhý dotaz pro dotaz třetí), opět se stejným uspokojivým výsledkem.

Protože jde o námět k samostatnému procvičování, databázi ke stažení neuvádíme.

8 Navazující výpočty

Pokud nebude řečeno jinak, opírají se navazující výpočty o výsledky dotazu DD_Spotreby resp. o data tabulky TD_Spotreby, vše popsáno podrobně výše.

V některých dotazech je použito volání funkcí, které jsou k disposici v použitém databázovém systému. Pro konkrétní systém je vždy zapotřebí se s nimi seznámit. Pro databázový systém používaný např. v Microsoft Office je dokumentace k funkcím na odkazu zde. Pro použití ve výrazech SQL příkazů *select* je samozřejmě možno používat jen ty funkce, které vyhovují kontextu příkazu. O trochu obecnější (a tedy zúžený) výčet se pokouší článek zde autora © Jaromíra Skřivana, na jehož autorská práva odkazujeme.

8.1 Měsíční spotřeba

Jeden z nejjednodušších dotazů pro začátečníky. Pro pokročilé čtenáře *těchto* textů snad netřeba dalšího výkladu. I případná úprava typu např. omezení na interval měsíců bude pro ně hračkou.

```
select
  month(S.aDatum) as Mesic, year(S.Datum) as Rok,
  Sum(S.Spotreba) as SpotrebaCelkem
from TD_Spotreby S
group by year(S.Datum), month(S.aDatum)
order by year(S.Datum), month(S.aDatum)
```

Dotaz pojmenujme např. **DE_SpotrebyMesicne**. Část jeho výsledku:

DE_SpotrebyMesicne		
Mesic	**Rok**	**SpotrebaCelkem**
...	...	...
10	2023	183,939
11	2023	258,492
12	2023	253,961
...	...	...

8.2 Průměrná spotřeba v průběhu týdne

V domácnostech, kde týden je rozdělen na pracovní a víkendové dny, může být inspirující informace o rozdělení průměrné spotřeby v jednotlivých dnech týdne. Dobrou vypovídací schopnost má veličina [Spotřeba / Den].

```
select
  DatePart("w",aDatum,2) as Den,
  WeekDayName(DatePart("w",aDatum,2), False, 2) as  JmDne,
  Avg(Spotreba) AS Prumerne
from DD_Spotreby
group by DatePart("w",aDatum,2)
order by DatePart("w",aDatum,2)
```

Dotaz pojmenujme např. **DF_SpotrebaVtydnu**. Část jeho výsledku:

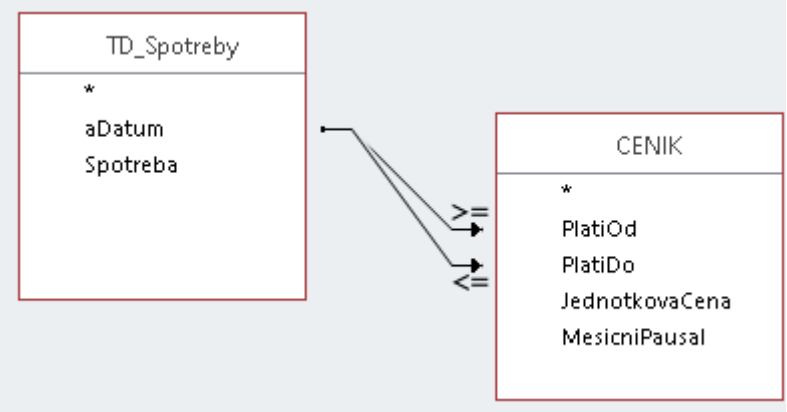
DF_SpotrebaVtydnu [kWh/Den]		
Den	**JmDne**	**Prumerne**
...	...	...
4	čtvrtek	5,108
5	pátek	6,225
6	sobota	9,883
...	...	...

8.3 Cena v jednotlivých dnech

Výhodou zvolené logiky (primární tabulka DATUMY jednotlivých po sobě jdoucích dnů) je značné zjednodušení navazujících výpočtů pro vcelku libovolná období a libovolná členění.

Výše byl podán postup pro získání odhadu spotřeby pro jednotlivé dny období pokrytého odečty (dotaz DD_Spotreby resp. tabulka TD_Spotreby). Ve spojení s tabulkou CENÍK je pak odhad nákladů na zaplacení spotřeby pro každý den skutečně jednoduchý.

Schematicky:



Obr. 4: Přístup k položkám ceníku

Dotazem:

```
select
  S.aDatum, S.Spotreba,
  C.JednotkovaCena, C.MesicniPausal,
  S.Spotreba * C.JednotkovaCena as CenaZaProud,
  C.MesicniPausal * 12 / 365.24 as Pausal,
  CenaZaProud+Pausal as CenaCelkem
from
  TD_Spotreby S left join CENIK C
    on (C.PlatiOd<=S.aDatum) and (S.aDatum<=C.PlatiDo)
```

Dotaz pojmenujme např. **DG_DenniPlatby**. Část jeho výsledku (30.10. byl vypnutý proud):

DG_DenniPlatby						
aDatum	Spotreba	Jednotkova Cena	Mesicni Pausal	Cena Za Proud	Pausal	Cena Celkem
...	...	...	...	...	...	...
30.10.2023	0,000	9,52	320	0	10,51	10,51
31.10.2023	14,749	9,52	320	140,41	10,51	150,92
01.11.2023	18,414	7,65	390	140,86	12,81	153,67
02.11.2023	9,639	7,65	390	73,74	12,81	86,55
...	...	...	...	...	...	...

8.4 Cena v jednotlivých měsících

Triviální dotaz do výsledku předchozího DG_DenniPlatby:

```
select
  Year(aDatum) AS Rok,
  Month(aDatum) AS Mesic,
  Sum(CenaCelkem) AS Zaplatit
from DG_DenniPlatby
group by Year(aDatum), Month(aDatum)
order by Year(aDatum), Month(aDatum)
```

Dotaz nazveme např. **DH_ZaplatitMěsíčně** s částí výsledku:

DH_ZaplatitMesicne		
Mesic	Rok	Zaplatit
...	...	...
10	2023	2 077,02
11	2023	2 361,87
12	2023	2 441,49
...	...	...

8.5 Návrh měsíčních záloh

Toto už je záležitost diskuse v každé domácnosti. Principiálně se lze obejít bez nějakých dalších databázových nástrojů. Pokud se však pilný čtenář chce dále procvičovat, pak několik námětů:

- Upravte vybrané dotazy tak, aby zobrazovaly výsledky pro období (interval dnů, měsíců). Pro jednoduchost zadávejte počáteční a koncový den jako konstanty přímo v dotazu SQL.
- Nechte zobrazit průměrnou částku k měsíčnímu zaplacení. A pro jistotu zvýšenou např. o 5%, ať je rezerva.
- Zvolte nějakou výši zálohy (třeba jako konstantu v dotazu SQL) a nechte zobrazit, kolik vám budou za zvolené období vracet. Vyjde-li záporná hodnota, je buď chyba v dotazu - ale častěji v chybném odhadu zálohy :-)

Přílohy

9 Poznámka k příkazu *select* s klauzulí *inner join*

Poznámka se týká příkazu select dle syntaxe tvaru

```
select
  SEZNAM
from
  ZDROJ1 inner join ZDROJ2 on PODMINKA
  ... atd
```

PODMINKA ve všech implementacích má tvar

```
ZDROJ1.POLE1 = ZDROJ2.POLE2
```

přičemž mnohé implementace připouští nejen relační operátor =, ale i všechny ostatní:

```
= | <> | < | <= | > | >=
```

kde každý operand uvedených relačních operací má pouze tvar podle definice

```
ZDROJi.POLEj
```

V některých databázových systémech lze použít také komplexnější podmínku tvaru

```
PODMINKA1 { AND | OR } PODMINKA2 { AND | OR } ...
```

kde každí PODMÍNKAi má shora uvedený tvar.

Ačkoliv to není nikde výslovně zmíněno, lze v databázovém systému Microsoftu (možná i jinde?) jako PODMINKA uvést obecně logický výraz (tj. odevzdávající hodnotu True nebo False). Musí však být splněno:

- Výraz musí obsahovat alespoň jedno použití datového pole ze ZDROJ1 a alespoň jedno použití datového pole ze ZDROJ2.
- Může obsahovat volání funkcí, ale jen z množiny interpretovaných funkcí (nejen matematických), kde to má vzhledem ke kontextu smysl - viz např. zde

tedy např.

```
... ON sin(ZDROJ1.ALFA) - cos(ZDROJ2.BETA-3.14159/4) < 0.01
```

Takový příkaz bude vykonán takto:

- Nejprve se vytvoří dočasný kartézský součin ZDROJ1 × ZDROJ2.
- Výstup příkazu bude generován z těch prvků kartézského součinu, pro které se vyhodnotí PODMINKA jako **true**.

Výstup bude respektovat event. další klauzule příkazu select (řazení - order by apod).

V tomto kontextu tedy vypadá PODMINKA v části ON jako další filtr ve vztahu logického součinu k výrazu v klauzuli *where*. Skutečně lze pozorovat v řešeních mnoha studentů konstrukce např. typu

```
...join...on ZDROJ1.POLE1 = ZDROJ2.POLE2 and ZDROJ1.POLE3 > 100
```

(bez části *where* v příkazu *select*), protože jim připadne tato část nadbytečná.

Důležitá poznámka: *Velkou opatrnost je třeba zachovávat při současném použití klauzule „where“ a podmínek v části „on“ v příkazu s „join“. Vždy je třeba se v konkrétním databázovém systému detailně seznámit s tím, jak se a) skutečně provádí jednotlivé typy „join“ (left, inner ...) a b) v jakém vztahu jsou podmínky v klauzuli „where“ a části „on“ (pořadí vyhodnocení, dopad každé z nich na výsledek dotazu ...).*

10 (Nedůležitá) poznámka k logickým hodnotám

Jde o hodnoty datového typu Logical, Boolean, Yes/No apod. (podle použitého databázového systému).

Databázové systémy obecně vykonávají operace s logickými hodnotami po matematické stránce samozřejmě v pořádku. Různé databázové systémy však mohou používat různý způsob ukládání těchto hodnot ve svých souborech. Od toho se pak může odvíjet zobrazení logických hodnot, pokud databázový systém nabízí i nějaký databázový program pro uživatele (např. Microsoft ke svému databázovému systému nabízí databázový program Access).

Teoreticky pro uchování logické hodnoty stačí jediný bit, z hlediska instrukční sítě procesorů je však praktičtější použít jeden byte. Opět: lze z něj použít jediný, autory databáze definovaný bit (např. první zleva nebo první zprava). Procesory mají optimálně navržené instrukce pro práci s celými čísly se znaménkem i bez znaménka, na jednom i více bytech. Toho využívají autoři databází: ukládají logickou hodnotu většinou na jednom bytu (někteří plýtvají a třeba i na více) jako celé číslo. Tam však podobnost napříč databázovými systémy končí. Nejčastěji se lze setkat s těmito způsoby:

- **Celé číslo se znaménkem, jediný určený bit.** Hodnota bitu 1 bývá stanovena jako PRAVDA, 0 jako NEPRAVDA. Je-li určeným bitem první zprava (bit 0), pak liché číslo znamená PRAVDA , sudé NEPRAVDA. Je-li určeným bitem první zleva (bit 7), pak záporné číslo znamená PRAVDA, nula a kladné NEPRAVDA. Je-li určeným bitem jakýkoliv jiný, pak se testuje pouze tento bit.
- **Celé číslo se znaménkem, hodnota.** Tady je to pestré. Např. je-li alespoň jeden bit nenulový, chápe se jako PRAVDA, jsou-li všechny bity nulové, je to numericky nula a chápe se jako NEPRAVDA. Může to být i takto: PRAVDA je to tehdy, jsou-li všechny bity 1 (tj. u procesorů Intel hodnota -1), jinak je to NEPRAVDA. Může to být ale i naopak.
- **Celé číslo bez znaménka, jediný bit.** Testuje se tento bit.
- **Celé číslo bez znaménka, hodnota.** Chápe se jako PRAVDA pouze tehdy, jsou-li všechny bity rovny 1 (pro jeden byte je to hodnota $FF_{16} = 255_{10}$), jinak je to NEPRAVDA. Někteří autoři databází to definují naopak: Pouze všechny nulové bity (tj. $00_{16} = 0_{10}$)jsou NEPRAVDA, jinak je to PRAVDA. Opět to klidně může být i naopak.
- **Poznámka:** Jak je vidět, je v tom pěkný zmatek. Naštěstí tyto podrobnosti začínající databázista nemusí vědět s výjimkou jediné věci, a to je interpretace tabulky uvedené níže v odstavci „K bodu 3: Kontrola relevantnosti“. Byl použit databázový systém Microsoftu a jeho program Access, kde PRAVDA je zobrazena jako -1, NEPRAVDA jako 0.

11 Možná řešení námětů z odstavce *Odhady stavů k půlnoci*

11.1 K bodu 1: Sloučení dotazů DB a DC

Cílíme k následujícímu výsledku:

DB_PredNaslOdectySmStavem					
aDatum	**pDatum**	**pStav**	**nDatum**	**nStav**	**mStav**
...	...	...	...	...	...
02.10.2023	30.09.2023	17 830	03.10.2023	17 848	17 838,589
03.10.2023	30.09.2023	17 830	03.10.2023	17 848	17 844,368
04.10.2023	03.10.2023	17 848	10.10.2023	17 890	17 850,215
05.10.2023	03.10.2023	17 848	10.10.2023	17 890	17 856,176
06.10.2023	03.10.2023	17 848	10.10.2023	17 890	17 862,138
...	...	...	...	...	...

Možný tvar dotazu:

```
PředchozíStav (pStav) = Stav nejbližšího nižšího odečtu
PředchozíDatum (pDatum) = Datum nejbližšího nižšího odečtu
NásledujícíStav (nStav) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (nDatum) = Datum nejbližšího vyššího odečtu
PůlnočníStav (mStav) = Odhad k půlnoci (midnight) dne aDatum
select
  PN.DATUM as aDatum,
  PP.DATUM as pDatum,
  PP.STAV as pStav,
  NN.DATUM as nDatum,
  NN.STAV as nStav,
  pStav + (aDatum-pDatum)*(nStav-pStav)/(nDatum-pDatum) as mStav
from
  (DA_PredNaslDatumy PN inner join ODECTY PP
      on PN.pDatum = PP.DATUM)
    inner join ODECTY NN on PN.nDatum = NN.DATUM
```

11.2 K bodu 2: Zjednodušení výsledku dotazu DC

Výsledek je stejný jako výše, ale bez použití dotazu DB - přímo z dotazu DA:

DC_PulnocniStavyZjednodusene	
aDatum	**mStav**
...	...
02.10.2023	17 838,589
03.10.2023	17 844,368
04.10.2023	17 850,215
05.10.2023	17 856,176
06.10.2023	17 862,138
...	...

Možný tvar dotazu:

```
PředchozíStav (PP.STAV) = Stav nejbližšího nižšího odečtu
PředchozíDatum (PP.DATUM) = Datum nejbližšího nižšího odečtu
NásledujícíStav (NN.STAV) = Stav nejbližšího vyššího odečtu
NásledujícíDatum (NN.DATUM) = Datum nejbližšího vyššího odečtu
PůlnočníStav (mStav) = Odhad k půlnoci (midnight) dne aDatum
select
  PN.DATUM,
  PP.STAV
  +(PN.DATUM-PP.DATUM)
  *(NN.STAV-PP.STAV)
  /(NN.DATUM-PP.DATUM) as mStav
from
  (DA_PredNaslDatumy PN inner join ODECTY PP
       on PN.pDatum = PP.DATUM)
    inner join ODECTY NN on PN.nDatum = NN.DATUM
```

11.3 K bodu 3: Kontrola relevantnosti odhadu stavu

Zřejmě nejjednodušší je doplnit seznam počítaných výrazů v dotazu DC:

```
select
  aDatum, pDatum, pStav, nDatum, nStav,
  pStav+(aDatum-pDatum)*(nStav-pStav)/(nDatum-pDatum) AS mStav,
  pStav <= mStav and mStav <= nStav as OK
from DB_PredNaslOdecty
```

s výsledkem

DC_S_kontrolou						
aDatum	**pDatum**	**pStav**	**nDatum**	**nStav**	**mStav**	**OK**
...	...	...	...	...	...	...
02.10.2023	30.09.2023	17 830	03.10.2023	17 848	17 838,589	-1
03.10.2023	30.09.2023	17 830	03.10.2023	17 848	17 844,368	-1
04.10.2023	03.10.2023	17 848	10.10.2023	17 890	17 850,215	-1
05.10.2023	03.10.2023	17 848	10.10.2023	17 890	17 856,176	-1
06.10.2023	03.10.2023	17 848	10.10.2023	17 890	17 862,138	-1
...	...	...	...	...	...	...

12 Možné řešení situace s výměnou měřidel

Na rozdíl od předchozích se tato kapitola týká odběru (studené) vody. Pro tuto komoditu bylo totiž pro testování k disposici dostatečné množství dat jak vlastních odečtů, tak dat o výměnách měřidel. Protože jde o reálná data, jistě bude kapitola působit věrohodněji.

12.1 Nezbytný úvod

Ve stanovených intervalech mají dodavatelé komodit zákonem stanovenou povinnost provádět u svých odběratelů výměnu měřidel za měřidla prokazatelně nově kalibrovaná k tomu schválenou firmou. Při výměně potvrzuje odběratel datum a čas výměny, a rovněž osobně zkontrolované stavy původního a nově zabudovaného měřidla. Toto nově zabudované měřidlo

nebývá zcela nové; prošlo však údržbou, jeho bezchybná funkce je nově zkontrolována a měřidlo nakalibrováno. Proto ve většině případů není stav na něm zobrazovaný nulový.

Při fakturaci odběrateli rozdíly stavů měněných a nově instalovaných měřidel bere dodavatel samozřejmě v úvahu. Okamžiky výměny měřidel jsou pak (vedle změny ceny) bodem, který dělí tok času na pod-intervaly. Každý takový pod-interval je tedy „od výměny nebo změny ceny k následující výměně nebo změně ceny".

Případ změny ceny je zahrnut ve výše popsaném řešení. Rozšířit toto řešení o případ výměn měřidel (může jich být ve sledovaném časovém období více!) se jeví pro velkou část čtenářů s ohledem na jejich předpokládanou úroveň dosavadních znalostí jako problematické. Zkusme tedy upravit logiku řešení tak, aby se maximálně využily doposud popsané postupy - nejlépe zcela.

Pokud se tedy na už hotovém SQL řešení nemá změnit nic a přesto cílový odhad spotřeby a plateb má zůstat stejný, musí se evidentně změnit vstupní data. Určitě nic nenaděláme (i když bychom velmi rádi) s cenami. Mít místo jedné tabulky jednotlivých dní pokrývající požadované období tabulek několik je kontraproduktivní (= hovorově blbost): už proto, že většinou sledujeme více komodit a u nich se určitě nekryjí intervaly výměny měřidel.

Zbývá tedy tabulka odečtů. Pokud někdy během období daného tabulkou DATUMY dojde k výměně měřidla, pak v tabulce ODEČTY se budou hodnoty stavu nejblíže před okamžikem výměny a nejblíže po okamžiku výměny skokově lišit (ať nahoru nebo dolu). Tento „skok“ bude roven rozdílu údajů na starém měřidle a na měřidle nově instalovaném:

ODEČTY		
DATUM (odečtu)	**TYP** (komodity)	**STAV** (skutečně zobrazený)
...	...	...
23.03.2017 12:19:00	V	2282,40
24.03.2017 12:30:00	V	2282,79
25.03.2017 16:10:00	V	2283,35
26.03.2017 13:54:00	V	**2283,68**
27.03.2017 10:00:00	V	**5061,56**
27.03.2017 13:20:00	V	5061,95
28.03.2017 18:39:00	V	5062,32
29.03.2017 14:26:00	V	5062,44
...	...	...

Ovšem tento rozdíl nám zcela rozbije odhady cen podle doposud vytvořeného algoritmu. Aby k tomu nedošlo, musí se v tabulce ODEČTY hodnoty zaznamenaných stavů nějak přepočíst na plynulé pokračování stavů z období před výměnou do období po výměně (stejně pak v případě další výměny).

Pro vyrovnání zmíněného skokového rozdílu odečtů je zřejmé, že bude nutno znát hodnoty údajů starého a nového měřidla v okamžiku každé výměny. To je ovšem pro nás, pečlivé hospodáře vlastní domácnosti, to nejmenší. Prostě sáhneme (i o půlnoci poslepu ☺) pro příslušné, před mnoha lety založené protokoly o výměně. Vybereme jednak ty, které se týkají dané komodity, jednak spadají do intervalu datumů, který nás momentálně zajímá. Z nalezených podkladů vznikne analogie následující tabulky (označme ji např. MĚŘIDLA):

MERIDLA			
DATUM (výměny)	**TYP** (komodity)	**STAV_P** (původního měřidla)	**STAV_N** (nového měřidla)
...	...	...	...
T1 = 27.03.2017 8:30:00	V	**2283,92**	**5060,99**
T2 = 18.10.2023 12:00:00	V	5974,03	2720,98
...	...	...	...

Poznámka: *Pro vysvětlení navrhovaného úpravy řešení odhadů to jistě stačí. Pro případnou komplexní realizaci samozřejmě tabulku už dávno máme, abychom nemuseli hledat prastaré protokoly. Tabulka je naplněna daty výměny měřičů jednak pro všechny komodity sledované v naší domácnosti, jednak - terminologií strejdy gůůgla - od věky věků. Že se interval datumů této tabulky vůbec nemusí krýt s intervalem datumů jak námi požadovaného k odhadu, tak námi požadované komodity, vůbec nesmí vadit.*

12.2 Jedna výměna

Zmíněný „skok“ (= rozdíl údajů na novém a starém měřidle, označme ho třeba R1) můžeme nyní kvantifikovat použitím dat z tabulky MĚŘIDLA:

```
R1 = MĚŘIDLA.STAV_N - MĚŘIDLA.STAV_P          (4)
```

Pro názornost doplňme naši tabulku odečtů o ony dva, provedené v okamžiku výměny:

ODEČTY			
DATUM (odečtu)	**TYP** (komodity)	**STAV** (skutečně zobrazený)	
...	...	...	
23.03.2017 12:19:00	V	2282,40	
24.03.2017 12:30:00	V	2282,79	
25.03.2017 16:10:00	V	2283,35	
26.03.2017 13:54:00	V	2283,68	
T1 = 27.03.2017 8:30:00	V	**2283,92**	

ODEČTY			
DATUM (odečtu)	**TYP** (komodity)	**STAV** (skutečně zobrazený)	
T1 = 27.03.2017 8:30:00	V	**5060,99**	R1 = 5060,99-2283,92 = **2777,07**
27.03.2017 10:00:00	V	5061,56	
27.03.2017 13:20:00	V	5061,95	
28.03.2017 18:39:00	V	5062,32	
29.03.2017 14:26:00	V	5062,44	
...	...	...	

Máme tedy tabulku fyzických (námi provedených) odečtů, nyní však reflektující stavy před a po výměnách měřidel. Pro uvažovaný způsob odhadů však potřebujeme ekvivalentní tabulku, ovšem bez skoků způsobených výměnou měřidel. A teprve tuto tabulku použijeme jako „ostrou“ pro vytvoření námi akceptovatelných odhadů.

Prakticky to znamená:

- Na začátku sledování naší spotřeby odečítáme z nějakého nainstalovaného měřidla. Hodnoty odečtů ODEČTY.STAV budou postupně růst až do okamžiku první výměny měřidla (okamžik T1).
- Stavy měněného = původního měřidla (STAV_P) a nově instalovaného (STAV_N) v okamžiku T1 jsou v jednom ze záznamů tabulky MĚŘIDLA. Pro jednoznačné označení použijme databázovou symboliku MĚŘIDLA.STAV_P a MĚŘIDLA.STAV_N .
- Následující námi provedené fyzické odečty už budou dle ukazatele nového měřidla. Ponechme jejich označení spolu s předchozími (opět včetně datového zdroje) ODEČTY.STAV.
- Kdybychom tento následující odečet provedli přesně v okamžiku T1, bude se od údaje dle původního měřidla lišit o hodnotu R1 = MĚŘIDLA.STAV_N - MĚŘIDLA.STAV_P. Bude-li MĚŘIDLA.STAV_N ≥ MĚŘIDLA.STAV_P, bude R1 ≥ 0, bude-li MĚŘIDLA.STAV_N ≤ MĚŘIDLA.STAV_P, bude R1 ≤ 0.
- Hodnoty všech následujících (fyzických) odečtů ODEČTY.STAV pak budou o R1 **větší** (operace **přičítání**; pozor, R1 může být i záporné!) než by byly ty, kdyby měřidlo nebylo měněno.
- Ovšem právě tyto následující (fyzické) odečty potřebujeme přepočítat. Při každém odečtu po okamžiku T1 potřebujeme tedy od každého takového fyzického odečtu ODEČTY.STAV **odečíst** hodnotu R1. Tím získáme hodnotu ODEČTY.STAV_P = hodnotu přepočtenou na pokračování předchozích odečtů a tím se také odstraní onen nepříjemný skok v (časově) plynulé posloupnosti odečtů.

Je zřejmé, že odečítat R1 se bude jen pro ty záznamy tabulky ODEČTY, pro něž je ODEČTY.DATUM > MĚŘIDLA.DATUM = T1.

Odečty fyzické (skutečné) a přepočtené				
DATUM (odečtu)	**TYP** (komodity)	**STAV** (odečtený)	**STAV_P** (přepočtený)	**Přepočet** (podrobněji)
...	...	...	...	...
23.03.2017 12:19:00	V	2282,40	2282,40	
24.03.2017 12:30:00	V	2282,79	2282,79	
25.03.2017 16:10:00	V	2283,35	2283,35	
26.03.2017 13:54:00	V	2283,68	2283,68	
27.03.2017 8:30:00	V	**2283,92**	**2283,92**	
				R1 = MĚŘIDLA.STAV_N - MĚŘIDLA.STAV_P R1 = 5060,99 - 2283,92 = **2777.07**
27.03.2017 8:30:00	V	**5060,99**	**2283,92**	ODEČTY.STAV_P = = ODEČTY.STAV - R1 5060,99 - **2777,07** = 2283,92
27.03.2017 10:00:00	V	5061,56	2284,49	ODEČTY.STAV_P = = ODEČTY.STAV - R1 5061,56 - **2777,07** = 2284,50
27.03.2017 13:20:00	V	5061,95	2284,88	dtto = 2284,89
28.03.2017 18:39:00	V	5062,32	2285,25	dtto = 2285,26
29.03.2017 14:26:00	V	5062,44	2285,37	dtto = 2285,38
...	...	...	...	...

12.3 Více výměn

Do časového intervalu námi odhadovaných stavů může padnout více výměn měřidel. Ve výše uvedené tabulce MĚŘIDLA je uvedena další výměna v okamžiku T2. Ta způsobí další nespojitost v řadě hodnot odečtů tabulky ODEČTY.

Pokud bychom aplikovali algoritmus odstranění nespojitosti jen pro okamžik T1, zůstane nespojitost v okamžiku T2 event. v dalších. Zřejmě bude nutno modifikovat úvahy vyjádřené shora pro jedinou výměnu a pokusit se je zobecnit.

Poznámka: *Ve zbytku této kapitoly bude pro rozdíl stavů R(Ti) v okamžiku Ti použito rovněž označení Ri.*

Výchozí tabulka odečtů skutečně zobrazovaných postupně na různých měřidlech:

ODEČTY		
DATUM (odečtu)	**TYP** (komodity)	**STAV** (skutečně zobrazený)
...	...	...
23.03.2017 12:19:00	V	2282,40
24.03.2017 12:30:00	V	2282,79
25.03.2017 16:10:00	V	2283,35
26.03.2017 13:54:00	V	2283,68
T1 = 27.03.2017 8:30:00	V	**2283,92**
T1 = 27.03.2017 8:30:00	V	**5060,99**
27.03.2017 10:00:00	V	5061,56
27.03.2017 13:20:00	V	5061,95
28.03.2017 18:39:00	V	5062,32
29.03.2017 14:26:00	V	5062,44
...	...	...
16.10.2023 17:19:00	V	5973,05
17.10.2023 7:00:00	V	5973,24
17.10.2023 15:15:00	V	5973,65
T2 = 18.10.2023 12::00:00	V	**5973,96**
T2 = 18.10.2023 12:00:00	V	**2720,98**
19.10.2023 13:00:00	V	2721,44
19.10.2023 16:20:00	V	2721,52
19.10.2023 17:00:00	V	2721,54
19.10.2023 18:10:00	V	2721,57
...	...	...

- Po první výměně je skutečně nutno zmenšit stavy fyzických odečtů o R1 - ale nikoliv všech, pouze odečtů od okamžiku T1 do okamžiku T2, tj. pro ODEČTY.DATUM ∈ <T1;T2> .

- Od okamžiku T2 do okamžiku T3 další výměny (nebo pro T3 = konec intervalu odhadovaných datumů) se musí posunout stavy fyzických odečtů nejen o R1, ale také o R2 = MĚŘIDLA.STAV_N (T2) - MĚŘIDLA.STAV_P (T2) pro ODEČTY.DATUM ∈ <T2;T3> a MĚŘIDLA.DATUM=T2 . Stavy fyzických odečtů se musí tedy zmenšit o R1+R2 pro odečty mající ODEČTY.DATUM ≥ MĚŘIDLA.DATUM=T2.
- Analogicky pro event. další výměny v okamžicích T4, T5 atd. Stavy fyzických odečtů se musí v jednotlivých časových intervalech (od jedné výměny k výměně následující) zmenšit o R1+R2+R3 pro odečty mající ODEČTY.DATUM ≥ MĚŘIDLA.DATUM=T3, o R1+R2+R3+R4 pro odečty mající ODEČTY.DATUM ≥ MĚŘIDLA.DATUM=T4 atd.

Obecně je hodnota zmenšení toho odečtu, který má konkrétní ODEČTY.DATUM, rovna

$$\sum R(Ti) \qquad (5)$$

kde se sčítá $\forall$ Ti ≤ ODEČTY.DATUM.

Následující tabulka dokumentuje popsaný postup získání přepočtených stavů. Pro kontrolu je uveden výpočet celkové spotřeby v daném intervalu dnů (zde od 23.03.2017 do 19.10.2023). Jednou je celková spotřeba zjištěna jako součet spotřeb z každého ze tří (nespojitých) intervalů odečtených stavů, jednak z jediné spojité řady přepočtených stavů. Úžasné: jsou stejné!

Odečty fyzické (skutečné) a přepočtené						
DATUM (odečtu)	**TYP** (komodity)	**STAV** (skutečně zobrazený)	**Rozdíly** Ri	**Přepočet** STAV-∑Ri	**STAV_P** (přepočtený)	**SPOTŘEBA** (ze zobrazeného)
...	...	...	R0 = 0		...	
23.03.2017 12:19:00	V	2282,40		STAV - 0 =	2282,40	2283,92-2282,40 = 1,52
24.03.2017 12:30:00	V	2282,79			2282,79	
25.03.2017 16:10:00	V	2283,35			2283,35	
26.03.2017 13:54:00	V	2283,68			2283,68	
27.03.2017 8:30:00	V	**2283,92**	R1 = 2777,07		**2283,92**	
27.03.2017 8:30:00	V	**5060,99**		STAV - 2777,07 =	**2283,92**	5973,96-5060,99 = 912,97
27.03.2017 10:00:00	V	5061,56			2284,50	
27.03.2017 13:20:00	V	5061,95			2284,89	
28.03.2017 18:39:00	V	5062,32			2285,26	
29.03.2017 14:26:00	V	5062,44			2285,38	
...	...	...			...	
16.10.2023 17:19:00	V	5973,05			3195,98	
17.10.2023 7:00:00	V	5973,24			3196,17	
17.10.2023 15:15:00	V	5973,65			3196,58	
18.10.2023 12:00:00	V	**5973,96**			**3196,89**	

Odečty fyzické (skutečné) a přepočtené						
DATUM (odečtu)	**TYP** (komodity)	**STAV** (skutečně zobrazený)	**Rozdíly** Ri	**Přepočet** STAV-∑Ri	**STAV_P** (přepočtený)	**SPOTŘEBA** (ze zobrazeného)
18.10.2023 12:00:00	V	**2720,98**	R2 = -3252,98 R1+R2 = = -475,91	STAV - (-475,91) =	**3196,89**	2721,57-2720,98 = 0,59
19.10.2023 13:00:00	V	2721,44			3197,35	
19.10.2023 16:20:00	V	2721,52			3197,43	
19.10.2023 17:00:00	V	2721,54			3197,45	
19.10.2023 18:10:00	V	2721,57			3197,48	
Kontrola celkové spotřeby:					3197,48-2282,40 = **915,08**	**915,08**

12.4 SQL realizace

Cílem je vytvoření dotazu, který z tabulky ODEČTY [DATUM, TYP, STAV] se stavy z různých měřidel dané komodity za pomoci tabulky MĚŘIDLA s údaji měněných a nových měřidel vytvoří množinu záznamů o stejném rozsahu jako ODEČTY, avšak rozšířenou o datové pole (označme ho např. STAV_P) s obsahem rovným přepočtené hodnotě STAV. Kromě pole STAV_P může samozřejmě výsledek dotazu obsahovat další „pracovní“ datová pole.

Protože jsme v předchozí kapitole označili jako R(Ti) rozdíl

```
R(Ti) = MĚŘIDLA.STAV_N(Ti) - MĚŘIDLA.STAV_P(Ti)      (6)
```

lze vztah (5) zapsat

```
∑ ( MĚŘIDLA.STAV_N(Ti) - MĚŘIDLA.STAV_P(Ti) )      (7)
```

a upravit

```
∑ MĚŘIDLA.STAV_N(Ti) - ∑ MĚŘIDLA.STAV_P(Ti)      (8)
```

Cílíme tedy zjednodušeně k dotazu tvaru

```
select
  O.DATUM, O.TYP, O.STAV,
  O.STAV - Rx as STAV_P
from ODECTY O
order by O.TYP, O.DATUM
```

V tomto dotazu je třeba definovat výraz Rx. Použijeme k tomu vztah (8) - důvod viz níže. Vztah (8) je rozdílem dvou výrazů; každý z nich je v podstatě sám o sobě návodem k zápisu v SQL. Každý z nich je součtem stavů všech postupně použitých měřidel dané komodity (rovnost typu

komodity), pro které je datum výměny nanejvýš rovno datumu odečtu. Jeden z těchto výrazů se týká stavu starého měřidla, jeden stavu nového měřidla.

Použijeme-li pro MĚŘIDLA alias M a pro ODEČTY alias O, jde o dva výrazy; jeden pro součet stavů starých měřidel (SUMSTSS), jeden pro součet stavů nově instalovaných měřidel (SUMSTNN):

```
(select sum(STAV_S) from MERIDLA M
   where M.DATUM<=O.DATUM and M.TYP=O.TYP) as SUMSTSS
(select sum(STAV_N) from MERIDLA M
   where M.DATUM<=O.DATUM and M.TYP=O.TYP) as SUMSTNN
```

Použito v konstruovaném dotazu:

```
select
  O.DATUM, O.TYP, O.STAV,
  O.STAV - (SUMSTNN - SUMSTSS) as STAV_P,
  (select sum(STAV_S) from MERIDLA M
     where M.DATUM<=O.DATUM and M.TYP=O.TYP) as SUMSTSS,
  (select sum(STAV_N) from MERIDLA M
     where M.DATUM<=O.DATUM and M.TYP=O.TYP) as SUMSTNN
from ODECTY O
order by O.TYP, O.DATUM
```

Nyní vysvětlivka použití vztahu (8) (dva výrazy) místo např. (7) (jen jeden výraz). Některé (starší?) databázové systémy neinterpretují hodnotu null v aritmetických operacích jako nulu, ale výsledek takové operace odevzdávají jako null (nebo třeba přímo havarují). Tato situace může nastat v popisované úloze např. úplně na začátku měření, úplně na konci nebo v obou případech, když se při zpracování více komodit najednou nekryjí jejich intervaly datumů.

V případě diskutovaného dotazu jde o operaci SUMSTNN - SUMSTSS, kde jeden, druhý nebo oba operandy mohou být null. Je proto vhodné zajistit explicitní převod hodnoty null na hodnotu nula a je logičtější to udělat pro každý z operandů zvlášť.

Dotaz s doplněným testem na hodnoty null:

```
select
  O.DATUM, O.TYP, O.STAV,
  STAV+SUMSTS-SUMSTN as STAV_P,
  (select sum(STAV_S) from MERIDLA M where M.DATUM<=O.DATUM and
M.TYP=O.TYP) as SUMSTSS,
  (select sum(STAV_N) from MERIDLA M where M.DATUM<=O.DATUM and
M.TYP=O.TYP) as SUMSTNN,
  IIf(IsNull(SUMSTSS),0,SUMSTSS) as SUMSTS,
  IIf(IsNull(SUMSTNN),0,SUMSTNN) as SUMSTN
from ODECTY O
order by O.TYP, O.DATUM
```

Závěrem: Pokud bude databáze obsahovat tabulku MĚŘIDLA, pak zřejmým a celkem triviálním postupem vytvoříme z výsledku předchozího dotazu tabulku ODEČTY s přepočtenými stavy, a odhadneme spotřebu včetně respektování event. výměny měřidel.