TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Optimalizace struktury a dotazů podnikové databáze

Diplomová práce

Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie

Autor práce: **Bc. Pavel Dvorský**
Vedoucí práce: Ing. Roman Špánek, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání diplomové práce

Optimalizace struktury a dotazů podnikové databáze

Jméno a příjmení: **Bc. Pavel Dvorský**
Osobní číslo: M14000154
Studijní program: N2612 Elektrotechnika a informatika
Studijní obor: Informační technologie
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Proveďte analýzu současného stavu databáze zakázek na základě dostupných dat.
2. Vytvořte přehled dotazů, které je třeba vykonávat, jejich současnou časovou a prostorovou složitost a vyhodnoďte možné kroky optimalizace.
3. Navrhněte nové řešení, které bude schopné všechny požadované dotazy vykonávat v reálném čase, tedy bez před-počítání, případně navrhněte řešení pro dotazy, které nebude možné v reálném čase řešit.
4. Implementujte navržené řešení a ověřte jeho vlastnosti (rychlost odezvy, prostorovou náročnost) na reálných, případně modelových datech. Vyhodnoďte dosažené výsledky a navrhněte případné další kroky optimalizace.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	35 – 45 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] ZANIOLO, Carlo a Jim GRAY. Advanced database systems. San Francisco: Morgan Kaufmann Publishers, 1997. The Morgan Kaufmann series in data management systems. ISBN 1-55860-443-X.

[2] STANCZYK, Stefan K. Theory and practice of relational databases. London: UCL Press, 1990. ISBN 1-85728-232-9.

[3] STARKS, Joy L., Philip J. PRATT a Mary Z. LAST. Concepts of database management. Ninth edition. Australia: Cengage, [2019]. ISBN 978-1-337-09342-2.

Vedoucí práce:	Ing. Roman Špánek, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	25. listopadu 2019
Předpokládaný termín odevzdání:	18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 25. listopadu 2019

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

11. května 2020 Bc. Pavel Dvorský

Poděkování

Rád bych poděkoval kolegům za poskytnutí databáze pro vypracování této práce a jejich pomoc při seznámení s databází a její strukturou.

Abstrakt

Tato práce pojednává o analýze podnikové databáze a optimalizaci běžně používaných funkcí. Dále navrhuje úpravu databázové struktury a nahrazení univerzálního řešení správy vazeb mezi tabulkami, které zpomaluje výkon celého systému.

Dále se zabývá teoretickým řešením optimalizace statistiky a indexů, jejich údržbu a vhodnou metodikou pro jejich optimalizaci.

V neposlední řadě probírá možnosti využití In-Memory technologií. I když se jejich použití nehodí pro danou podnikovou databázi v této práci, nově připravovaný software postavený na základech této databáze může využít In-Memory databáze pro načítání 3D modelů.

Klíčová slova

MS SQL, optimalizace dotazů, DB, In-Memory

Abstract

This work deals with the analysis of the enterprise database and the optimization of commonly used functions. It also proposes modifying the database structure and replacing the universal solution for managing the relationships between tables, which slows down the performance of the entire system.

It also deals with the theoretical solution of optimization of statistics and indices, their maintenance and a suitable methodology for their optimization.

Last but not least, it discusses the possibilities of using In-Memory technologies. Although their use is not suitable for the given enterprise database in this work, newly prepared software built on the basis of this database can use the In-Memory database to load 3D models.

Keywords

MS SQL, query optimization, DB, In-Memory

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Databázové systémy jsou dnes součástí mnoha aplikací na mnoha zařízeních a slouží pro uchovávání dat různých druhů. Díky struktuře těchto systému lze s uloženými daty snadno pracovat i ve velkém množství. Nicméně špatně navržený a neudržovaný systém přináší problémy ve výkonu. Pomalé vyhledávání či ukládání dat v dnešním uspěchaném světě přináší ale nespokojenost uživatelů.

Cílem této práce je provést analýzu podnikové databáze zakázek a navrhnout optimalizacepro zvýšení jejího výkonu. Ne všechny kroky optimalizace ale je možné aplikovat na existující databázi. Omezení může být technologické, nebo rozvoji brání lidský faktor. I když není možné optimalizaci provést na stávajících datech, vždy je možnost v budoucnosti, kdy přijde nástupce a aktuální systém nahrazen novějšími technologiemi.

Tato práce se zaměřuje na optimalizaci kódu dotazů vybraných databázových funkcí, analýzu a úpravu databázové struktury, využití partition dělení a In-Memory OLTP systému.

2. Analýza podnikové databáze Aspe 10

Databáze programu Aspe 10 vznikla v první polovině roku 2016, kdy se v podstatě převzala databáze předchozí verze programu Aspe 9 a rozšířila se o tabulky a sloupce potřebné do nové aplikace. Struktura této databáze tedy byla navržena před více než 5 lety v databázovém systému MSSQL. Od té doby došlo k mnoha rozšířením a změnám, které v dnešní době negativně ovlivňují výkon databáze.

Vývoj aplikace Aspe 10 je v tuto chvíli ukončen. I když se připravuje nástupce této aplikace, dá se předpokládat, že po dobu následujících několika let se bude aplikace stále používat. Z toho důvodu padlo rozhodnutí analyzovat stávající strukturu a navrhnout optimalizace pro zvýšení výkonu.

2.1 Struktura databáze

Strukturu databáze lze rozdělit do následujících skupin:

- číselníky,
- cenové soustavy (datové základny),
- globální nastavení,
- rozpočty,
- zjišťovací protokoly,
- faktury,
- harmonogramy,
- technologický modely,
- ostatní.

Hlavní zaměření této práce je na optimalizaci nejčastěji používaných funkcí programu, což jsou operace s rozpočty, zjišťovacími protokoly a fakturami. ER diagram těchto tabulek je v Příloze A. Dále se tato práce zaměřuje na nejčastěji používané procedury a funkce a jejich optimalizaci.

2.2 Struktura rozpočtu

Rozpočet neboli stavba v programu Aspe 10 je dělen na 5 základních prvků. Následující paragrafy stručně popíší jednotlivé prvky a jejich vlastnosti. Výpis všech

sloupců daných tabulek je dlouhý a obsahuje spoustu zbytečných informací, které nejsou podstatné pro tuto práci a tebu budou uvedené pouze základní nebo důležité sloupce.

První prvek tvoří samostatná stavba a je uložena v tabulce *r_zakazky*. Tato tabulka obsahuje obecné informace stavby, kterými jsou její kód, značka, název, popis, investor, aktivní varianta stavby, parametry, výchozí hodnoty doplňujících údajů a další.

Druhým prvkem je varianta stavby, která je uložena v tabulce *r_zakazky_var*. Každá stavba může mít několik variant, kdy pouze jedna varianta je považována za aktivní. V této tabulce jsou uložené údaje dané varianty, tedy její kód, značka, název, parametry atd.

Třetí prvek tvoří objekt uložen v tabulce *r_etapy_var*. Objekt navazuje na variantu stavby stejně jako v předchozím případě jedna varianta může obsahovat několik objektů. Tabulka obsahuje základní informace o objektu, kterými jsou kód, značka, název, investor, projektant, datum zahájení a dokončení, parametry, doplňující údaje a další. Objekt je speciální prvek v tom, že na něj navazují nejen stavební díly, ale také dodatky (ZBV), zjišťovací protokoly a objekty další úrovně.

Čtvrtým prvkem jsou stavební díly uložené v tabulce *R_STAV_DILY*. Stejně jako u předchozích prvků tato tabulka obsahuje informace typu kód, značka, název, popis atd. Stavební díly lze považovat jako rozdělení položek dle určitých kritérií (všeobecné konstrukce a práce, zemní práce, vodorovné konstrukce, svislé konstrukce atd.) pro lepší přehlednost.

Pátým a prvkem jsou položky rozpočtu uložené v tabulce *r_polozky*. Položky lze považovat za nejdůležitější prvek rozpočtu, neboť určují samostatnou stavbu, vykonané práce a cenu. Tabulka obsahuje základní informace položky, jako jsou kód, značka, název, množství, cena a měrná jednotka, vazbu na variantu a další doplňující údaje.

Kromě těchto 5 základních prvků rozpočet obsahuje dodatky, které dodatečně upravují množství položky, zjišťovací protokoly pro čerpání položek a faktury.

Dodatky jsou uloženy v tabulce *r_dodatky* a obsahují opět základní informace jako kód, značka, název, jméno autora, datum vystavení apod. Každý dodatek obsahuje změny na položkách uložené v tabulce *r_polozky_dod* se sloupci kód, vazby na položku a dodatek, změněné množství a další.

Zjišťovací protokoly jsou uloženy v tabulce *z_zjist_prot* a obsahují kód, značku, vazby na objekt, dodatek a fakturu a další informace jim příslušné. Samostatné čerpání položek je uloženo v tabulce *z_polozky*, obsahující vazby na položku rozpočtu a zjišťovací protokol, čerpané množství a další doplňující informace.

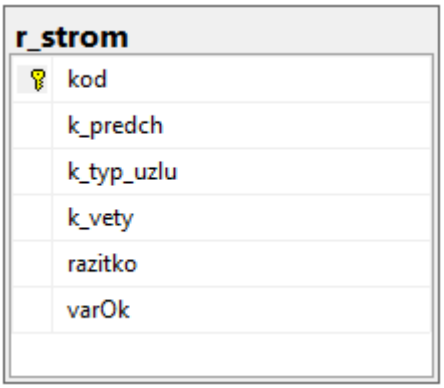
Faktury jsou dále uložené v tabulce *f_faktury* obsahující kód, značku, investora, období, cenu, vazbu na variantu a další doplňující údaje.

2.3 Vazební tabulka

V předchozí kapitole 2.2 je možné si všimnout 2 věcí, konkrétně:

a. pro databázi byl zvolen postup použití unikátního číselného kódu (sloupec *kod)* jakožto primárního klíče,

b. některé tabulky neobsahují vazby na další prvky. Tyto vazby jsou uložené v tabulce *r_strom*, která se tak dá považovat za páteřní tabulku celé databáze.



Obrázek 1 - tabulka r_strom

Na obrázku 1 jsou vidět sloupce vazební tabulky *r_strom*. Důležitými sloupečky jsou primární klíč *kod*, sloupec *k_predch* odkazující na *kod* předchozího prvku, *k_typ_uzlu* reprezentující prvek (položka rozpočtu, objekt, …) a *k_vety* odkazující na *kod* v konkrétní tabulce.

Tato tabulka je pozůstatek z doby, kdy se ještě nevědělo, jak bude samotný program fungovat a co bude obsahovat. V takovém případě je tato tabulka univerzálním řešením, neboť lze kdykoli přidat jakékoli nové vazby. Ovšem nyní je to krok navíc, který je nezbytné provádět při jakékoli operaci.

3. Optimalizace kódu

Tato kapitola se zabývá optimalizací nejčastěji používaných procedur a funkcí databáze.

Funkce uf_Nasledniky

Cílem této funkce je vypsat kód r_strom, kód předchůdce a kód věty všech konkrétních prvků, které náleží vybranému prvku. Jako příklad byla náhodně zvolena stavba s kódem 103656 a hledaly se všechny její objekty.

Tato funkce vytvoří tabulku @SoucTab, která je postupně naplněna všemi následníky do hledaného uzlu (stavební díl). Z této funkce se nakonec vyberou pouze následníci hledaného uzlu do tabulky @tree, jenž je výstupem.

```
ALTER FUNCTION [dbo].[uf_Nasledniky]
(
  @iKodStrom int,
  @uzel int
)
RETURNS @tree TABLE (kod int, k_predch int, k_vety int)
AS
BEGIN
----
declare @SoucTab table(
      kod int,
      k_predch int,
      k_vety int,
      k_typ_uzlu int)

insert into @SoucTab
select kod, k_predch, k_vety, k_typ_uzlu from r_strom where k_predch
= @iKodStrom
```

Naplnění tabulky @SoucTab je vidět v následujícím kódu. Zde si lze všimnout, že dochází k několika selectům při vyhledávání dat. V tomto kroku se tabulka plní následníky všech řádnů, které tato tabulka obsahuje. Následníci se neukládají duplicitně a vybírají se pouze ze seznamu existujících následníků, jenž je uložen v tabulce @tb_uzel. Hlavní problém této tabulky činí vnořené selecty u podmínky WHERE, které zpomalují tuto funkci.

```
while (1=1)
begin
      insert into @SoucTab
      select kod, k_predch, k_vety, k_typ_uzlu from r_strom
      where k_predch in (select kod from @SoucTab) and
                  k_typ_uzlu in (select kod from @tb_uzel) and
                  kod not in (select kod from @SoucTab)

      if @@rowcount = 0 break
end
```

Nově navržené řešení rozšiřuje tabulku @SoucTab o pomocný sloupec úroveň, který slouží k následnému spojení tabulek za účelem odstranění vnořených cyklů. V cyklu WHILE následujícího kódu probíhá plnění tabulky @SoucTab, kdy se stejně jako v předchozím kroku tabulka plní následníky, ale díky sloupci úroveň není nutné použít vnořeného selectu, který brání vkládání duplicitních hodnot.

```
declare @iuroven int

declare @SoucTab table(
      k_vety int,
      kod int,
      k_predch int,
      k_typ_uzlu int,
      uroven int default 0)

…

select @iuroven = 0

while (1=1)
begin
      select @iuroven = @iuroven + 1
      insert into @SoucTab
            (k_vety, kod, k_predch, k_typ_uzlu,  uroven)
      select r.k_vety, r.kod, r.k_predch, r.k_typ_uzlu,
            @iuroven as uroven
      from r_strom r
      join c_typy_uzlu ctu
            on r.k_typ_uzlu = ctu.kod
      join @SoucTab st
            on st.kod = r.k_predch
      where st.uroven = @iuroven - 1
            and r.k_typ_uzlu in (select kod from @tb_uzel)

      if @@ROWCOUNT = 0 break
end
```

Na obrázku 2 je znázorněn časový rozdíl pro vykonání funkce uf_Nasledniky a optimalizované funkce uf_NaslednikyN. Optimalizací kódu došlo ke zrychlení operace o 0,016 vteřiny, a to z původních 0,02 vteřiny na 0,004 vteřiny. Zrychlení se může zdát zanedbatelně malé, ale pokud se tato funkce použije opakovaně, může dojít až k několikavteřinovému zpomalení.

EventClass	TextData	StartTime	EndTime
SQL:BatchStarting	SELECT @@SPID;	2020-05-17 21:39:52.923	
SQL:BatchCompleted	SELECT @@SPID;	2020-05-17 21:39:52.923	2020-05-17 21:39:52.923
SQL:BatchStarting	select * from uf_Nasledniky(103656,14)	2020-05-17 21:39:52.923	
SQL:BatchCompleted	select * from uf_Nasledniky(103656,14)	2020-05-17 21:39:52.923	2020-05-17 21:39:52.943
SQL:BatchStarting	SELECT @@SPID;	2020-05-17 21:39:55.803	
SQL:BatchCompleted	SELECT @@SPID;	2020-05-17 21:39:55.803	2020-05-17 21:39:55.803
SQL:BatchStarting	select * from uf_NaslednikyN(103656,14)	2020-05-17 21:39:55.803	
SQL:BatchCompleted	select * from uf_NaslednikyN(103656,14)	2020-05-17 21:39:55.803	2020-05-17 21:39:55.807

Obrázek 2 - optimalizace funkce uf_Nasledniky

Funkce uf_SeznObjektuStejneUrovneFiltr

Tato funkce se používá pro získání seznamu všech objektů stejné úrovně jako zvolený objekt dané stavby. Využívá se při generování stromové struktury v programu a při filtrování seznamu objektů. Vstupními daty jsou kód zvoleného objektu z tabulky r_strom, kód jeho předchůdce a kód uživatele z důvodu oprávnění k objektům. Výstupem je seznam objektů stejné úrovně se základními informacemi objektu.

```
ALTER  FUNCTION [dbo].[uf_SeznObjektuStejneUrovneFiltr]
(
      @k_predch INT,
      @k_etapy_akt INT,
      @k_uzivatel int
)
returns @sezngrid table
(
      [kod] [int] Primary Key,
      [k_predch] [int],
      [znacka] [varchar] (15) COLLATE database_default NOT NULL ,
      [nazev] [varchar] (100) COLLATE database_default NULL ,
      [vypracoval] [varchar] (60)   COLLATE database_default,
      [zahajeni] [smalldatetime] NULL,
      [dokonceni] [smalldatetime] NULL,
      [predani] [smalldatetime] NULL
)

…

DECLARE @tree table(kod int,  k_predch int , k_vety int, levelNo
int)
DECLARE @pravaObj table(kod int, kod_etapy_Strom int, jsouPrava bit,
NadrZP bit, pristup bit )
```

Prvním krokem této funkce se vytvoření a naplnění tabulky @pravaObj, která obsahuje informace o oprávnění daného uživatele ke všem objektům stavby. Dále se vytvoří tabulka @tree, která obsahuje seznam všech objektů stavby a číselnou hodnotu levelNo, jenž reprezentuje úroveň objektu. Na základě těchto hodnot se pak do výstupní tabulky @sezngrid vloží požadovaný seznam objektů.

```
insert into @sezngrid
      SELECT     r_etapy_var.kod, T1.k_predch,
                 r_etapy_var.znacka,
                 r_etapy_var.nazev,
                 r_etapy_var.vypracoval,
                 r_etapy_var.zahajeni,
                 r_etapy_var.dokonceni,
                 r_etapy_var.predani
        from @tree T1
         inner join r_etapy_var on T1.k_vety = r_etapy_var.kod
           inner join @pravaObj P on T1.kod = P.kod_etapy_Strom and
      (P.jsouPrava=1 or pristup=1)
           where levelNo = (select TOP 1 levelNo
                 from @tree where kod = @k_etapy_akt)
```

Problém u této funkce je její stáří a změna funkcionality aplikace. Práva na objekty již dávno neexistují, takže celá tabulka @pravaObj strácí význam. Stejně tak tabulka @tree nemá smysl, neboť zbytečně vyhledává data navíc. V rámci zachování funkcionality v programu jsou zachovány vstupní parametry funkce, i když parametr uživatele i kód zvoleného objektu jsou v tuto chvíli zbytečné informace. Celou tuto funkci lze nahradit následujícím kódem, který přímo z tabulky r_strom na základě kódu předchůdce vybere všechny objekty stejné úrovně.

```
insert into @sezngrid
      select E.kod, S.k_predch, E.znacka, E.nazev, E.vypracoval,
E.zahajeni, E.dokonceni, E.predani
           from r_strom S
      inner join r_etapy_var E
           on E.kod = S.k_vety
      where S.k_predch = @k_predch and S.k_typ_uzlu = 12
      order by E.kod
```

Optimalizací této funkce se dosáhlo zrychlení z původních 0,043 vteřiny na prakticky instantní výstup 0,000 vteřiny.

TextData	StartTime	EndTime
SELECT @@SPID;	2020-05-17 23:52:51.400	
SELECT @@SPID;	2020-05-17 23:52:51.400	2020-05-17 23:52:51.400
select * from uf_SeznObjektuStejneUrovneFiltr(1607513, 1607514, 20)	2020-05-17 23:52:51.400	
select * from uf_SeznObjektuStejneUrovneFiltr(1607513, 1607514, 20)	2020-05-17 23:52:51.400	2020-05-17 23:52:51.443
SELECT @@SPID;	2020-05-17 23:52:53.130	
SELECT @@SPID;	2020-05-17 23:52:53.130	2020-05-17 23:52:53.130
select * from uf_SeznObjektuStejneUrovneFiltrN(1607513, 1607514, 20)	2020-05-17 23:52:53.133	
select * from uf_SeznObjektuStejneUrovneFiltrN(1607513, 1607514, 20)	2020-05-17 23:52:53.133	2020-05-17 23:52:53.133

Obrázek 3 - optimalizace funkce uf_SeznObjektuStejneUrovneFiltr

Funkce uf_Vrat_SeznamZakazekXGrid

Tato funkce vrací informace o zakázkách za daný útvar. Data pro XGrid se načítají neustále, takže jakákoli změna u této funkce se projeví při práci v programu. Na první pohled je zřejmé, kde je časové spoždění u této funkce. Podmínka WHERE pro vložení dat do tabulky obsahuje 7 vnořených selectů na různé úrovni.

```
insert @zakazky(kod,znacka,nazev,zahajeni,dokonceni,grid_oc,
  …
  where ((k_var_zakl in (select A.k_vety from
      (select * from  s__uzivtab_prava where (bez_omezeni = 1) or
          ((bez_omezeni = 0) and (kod in
          (select k_uzivtab_prava from s__uzivtab_prava_uziv
          where k_uzivatel = @uzivatel and cteni =1))))A
          inner join (select top 1 kod from s__uzivtab
          where tabulka = 'r_zakazky_var') B on A.k_uzivtab = B.kod
          )) or (k_var_Zakl not in (select A.k_vety
          from (select * from  s__uzivtab_prava) A
          inner join (select top 1 kod from s__uzivtab
          where tabulka = 'r_zakazky_var') B on A.k_uzivtab = B.kod
  )))
```

Nahrazením většiny selectů v této podmínce se snížil čas vykonání funkce na polovinu.

```
…
where ((k_var_zakl in
  (select P.k_vety from s__uzivtab_prava P
  right outer join s__uzivtab_prava_uziv PU
    on PU.k_uzivtab_prava = P.kod
  inner join (select kod from s__uzivtab
      where tabulka = 'r_zakazky_var') UT
    on UT.kod = P.k_uzivtab
  where ((PU.k_uzivatel = @uzivatel and cteni = 1)) or
    (P.bez_omezeni = 1))) or (k_var_zakl not in
      (select P.k_vety from s__uzivtab_prava P
      right outer join s__uzivtab_prava_uziv PU
        on PU.k_uzivtab_prava = P.kod
      inner join (select kod from s__uzivtab where tabulka =
          'r_zakazky_var') UT
        on UT.kod = P.k_uzivtab
```

TextData	StartTime	EndTime
SELECT @@SPID;	2020-05-18 02:29:09.333	
SELECT @@SPID;	2020-05-18 02:29:09.333	2020-05-18 02:29:09.333
select * from uf_Vrat_SeznamZakazekXGrid(20, 84)	2020-05-18 02:29:09.333	
select * from uf_Vrat_SeznamZakazekXGrid(20, 84)	2020-05-18 02:29:09.333	2020-05-18 02:29:09.347
SELECT @@SPID;	2020-05-18 02:29:11.180	
SELECT @@SPID;	2020-05-18 02:29:11.180	2020-05-18 02:29:11.180
select * from uf_Vrat_SeznamZakazekXGridN(20, 84)	2020-05-18 02:29:11.183	
select * from uf_Vrat_SeznamZakazekXGridN(20, 84)	2020-05-18 02:29:11.183	2020-05-18 02:29:11.190

Obrázek 4 - optimalizace funkce uf_Vrat_SeznamZakazekXGrid

Funkce uf_Nalezni_k_predch_predch

Tato funkce vrací kód předchozího prvku z tabulky r_strom a jejím vstupem je kód z tabulky r_strom libovolného prvku. Pokud je vstupem varianta stavby, vrátí funkce hodnotu NULL. Přestože se jedná o jednoduchou funkci a často používanou, je až komplikovaně napsaná. Autor zde postupně kontroluje typ vstupního prvku od položky po objekt.

```
DECLARE @ret int
Declare @typ int
select @typ = dbo.TypUzlu('r_polozky')
select top 1 @ret=k_predch from r_strom where kod = @kod AND
       k_typ_uzlu = @typ
if @ret is null
begin
     select @typ = dbo.TypUzlu('r_stav_dily')
     select top 1 @ret=k_predch from r_strom where kod = @kod AND
       k_typ_uzlu = @typ
end
if @ret is null
begin
     select @typ = dbo.TypUzlu('r_etapy_var')
     select top 1 @ret=k_predch from r_strom where kod = @kod AND
       k_typ_uzlu = @typ
    end
RETURN @ret
```

Celá tato funkce se dá zkrátit na 4 řádky se stejným výstupem, kdy se z tabulky r_strom rovnou vezme pole k_predch a poté se akorát nahradí hodnotou NULL, pokud bude kód předchůdce 0 (tedy vstupní prvek je varianta stavby). Tato změna má na rychlost téměř nulový dopad, jedná se čistě o údržbu dat.

```
DECLARE @ret int
select @ret = k_predch from r_strom where kod = @kod
     if (@ret = 0) select @ret = NULL
RETURN @ret
```

TextData	StartTime	EndTime
SELECT @@SPID;	2020-05-18 07:23:57.730	
SELECT @@SPID;	2020-05-18 07:23:57.730	2020-05-18 07:23:57.730
select dbo.uf_Nalezni_k_predch_predch(1334342)	2020-05-18 07:23:57.733	
select dbo.uf_Nalezni_k_predch_predch(1334342)	2020-05-18 07:23:57.733	2020-05-18 07:23:57.733
SELECT @@SPID;	2020-05-18 07:24:01.033	
SELECT @@SPID;	2020-05-18 07:24:01.033	2020-05-18 07:24:01.033
select dbo.uf_Nalezni_k_predch_predchN(1334342)	2020-05-18 07:24:01.037	
select dbo.uf_Nalezni_k_predch_predchN(1334342)	2020-05-18 07:24:01.037	2020-05-18 07:24:01.037

Obrázek 5 - optimalizace funkce uf_Nalezni_k_predch_predch

4. Úprava databázové struktury

Jednou z možných optimalizací databáze Aspe 10 je úprava databázové struktury. Vzhledem k firemní politice ale tento krok naráží na problém. Jak bylo zmíněno v kapitole 2, vývoj tohoto produktu byl ukončen. To má za následek, že velké zásahy do struktury jsou nežádoucí. Nicméně návrh této optimalizace lze aplikovat u nového produktu, který má v budoucnu nahradit stávající Aspe 10. Toto je možné, neboť nový produkt vychází ze stávající databáze Aspe 10. Je jednodušší provést změnu struktury databáze na začátku vývoje než u hotového produktu.

4.1 Odstranění tabulky r_strom

Prvním krokem při optimalizaci struktury databáze Aspe 10 je odstranění vazební tabulky *r_strom*, která snižuje výkon databáze. Jedná se o problém, který je tématem na úpravu již řadu let. Ovšem úprava této tabulky by ale znamenala úpravu celého programu, což není časově výhodné řešení pro stávající produkt.

Uveďme si následující příklad: z dostupných dat vypišme značku a název položky s kódem 3037503 (náhodně zvolená položka) a dále značku a název stavebního dílu, pod který patří.

Aktuální řešení vyžaduje celkem spojení 4 tabulek, pokud se chceme vyhnout vnořeným dotazům. Výsledný kód vypadá následovně:

```
SELECT     C.znacka 'znacka_pol',
           C.nazev 'nazev_pol',
           D.znacka 'znacka_dil',
           D.nazev 'nazev_dil'
      FROM r_strom A
      INNER JOIN r_strom B
           ON A.k_predch = B.kod
      INNER JOIN r_polozky C
           ON C.kod = A.k_vety
      INNER JOIN R_STAV_DILY D
           ON D.kod = B.k_vety
      WHERE A.k_typ_uzlu = 15 AND A.k_vety = 3037503
```

Nově navrhované řešení je vazbu vložit přímo do tabulky, ke které patří. V tomto případě se tabulka *r_polozky* rozšíří o sloupec *k_sd* a výsledný kód bude zjednodušen na spojení pouze 2 tabulek:

```
SELECT     A.znacka 'znacka_pol',
           A.nazev 'nazev_pol',
           B.znacka 'znacka_dil',
           B.nazev 'nazev_dil'
      FROM r_polozky A
      INNER JOIN R_STAV_DILY B
           ON A.k_sd = B.kod
      WHERE A.kod = 3037503
```

Rozšíření je vyžadováno u všech tabulek, které obsahují vazby v tabulce *r_strom*. Seznam vazeb je uložen v tabulce *c_typy_uzlu* (viz tabulka 1). Zde je uveden kód daného uzlu, jeho význam a odkaz na příslušnou tabulku.

I když tabulka *c_typy_uzlu* obsahuje všechny typy vazeb, ne všechny jsou použité. Například vazba s kódem 24 je zjišťovací protokol, ale v databázi není použitá. Jak bylo zmíněno v kapitole 2.2, tabulka *z_zjist_prot* již obsahuje sloupec *k_etapa*, jenž je vazba na objekt.

Tabulka 1 - seznam typů uzlů tabulky r_strom

kod	nazev	tabulka	popis_uzlu
1	Uzel	m_uzly	popis
2	HW klíč	m_hw_klice	******
3	Licence	m_licence_moduly	******
4	Kontakt	m_kontakty	******
5	Faktura	f_faktury	******
6	Položka faktury	f_faktury_polozky	******
7	Smlouva	m_smlouvy	******
8	E-mail	m_email	e_mail
9	Dokument	m_dokumenty	******
11	Varianta	r_zakazky_var	znacka+" - "+nazev
12	Objekt	r_etapy_var	znacka+" - "+nazev
14	Stavební díl	r_stav_dily	znacka+" - "+nazev
15	Položka rozpočtu	r_polozky	znacka+" - "+nazev
16	Potřeba	r_kalk_pol	******
17	Dodatek	r_dodatky	znacka+" - "+nazev
18	Útvar	c_utvary	******
21	Stavba	r_zakazky	******
23	NULL	r_polozky	@.znacka+"/"+@.nazev+" - "
24	Zjištovací protokol	z_zjist_prot	******
25	Položka ZP	z_polozky	******
26	Milník	r_milniky	milníky
41	Valorizační zjišťovací protokol	z_zjist_prot	******
43	Činnost harmonogramu	hgm_cinnosti	znacka+" - "+nazev

Tabulka 2 obsahuje seznam aktuálně používaných uzlů získaný z dostupných dat pomocí následujícího kódu:

```
SELECT     DISTINCT(k_typ_uzlu) 'kod',
           B.nazev,
           B.tabulka,
           B.popis_uzlu
      FROM r_strom A
      LEFT JOIN c_typy_uzlu B
           ON A.k_typ_uzlu = B.kod
```

Tabulka 2 - seznam aktuálních uzlů tabulky r_strom

kod	nazev	tabulka	popis_uzlu
11	Varianta	r_zakazky_var	znacka+" - "+nazev
12	Objekt	r_etapy_var	znacka+" - "+nazev
14	Stavební díl	r_stav_dily	znacka+" - "+nazev
15	Položka rozpočtu	r_polozky	znacka+" - "+nazev
17	Dodatek	r_dodatky	znacka+" - "+nazev
26	Milník	r_milniky	milníky
43	Činnost harmonogramu	hgm_cinnosti	znacka+" - "+nazev

Zde je vidět, jak se postupem času změnilo využití vazební tabulky, neboť se nyní používá pouze třetina původních uzlů.

Jak bylo zmíněno v kapitole 2.1, činnost harmonogramu a milník nebudou v této práci řešeny, zaměření je tedy na ostatních 5 uzlů.

Uzel Varianta

Tabulka *r_zakazky_var* již obsahuje sloupec *k_zakazka*, což je přímá vazba na stavbu. Zde není potřeba provádět žádné změny.

Uzel Objekt

Tabulka *r_etapy_var* neobsahuje žádné vazby na své předchůdce, tedy je potřeba tuto tabulku upravit. Jak bylo zmíněno v kapitole 2.2, objekt navazuje přímo na variantu, ale také může navazovat na jiný objekt (stává se podobjektem). Z toho důvodu je nezbytné tabulku rozšířit o 2 sloupce.

První nově přidaný sloupec je *k_predch,* reprezentující vazbu na předchozí objekt. Jelikož objekt nemusí mít předchůdce a je tedy vázán přímo pod variantu, může být toto

pole prázdné. Druhý nově přidaný sloupec je *k_varianta*, reprezentující vazbu na variantu stavby. Technicky toto pole může být prázdné a za pomoci prvního nového sloupce by se dalo dohledat variantu, do které objekt patří. Jelikož se ale často hledají všechny objekty stavby, mít toto pole vyplněné ušetří výsledný čas. Z toho důvodu nesmí mít nový sloupec *k_varianta* hodnotu NULL. Změna tabulky se provede pomocí následujícího kódu:

```
ALTER TABLE r_etapy_var
     ADD   k_predch INT NULL,
           k_varianta INT NOT NULL
```

Uzel Stavební díl

Tabulka *R_STAV_DILY*, stejně jako v případě objektu, neobsahuje přímou vazbu na svého předchůdce. Jelikož není potřeba vyhledávání stavebních dílů konkrétní varianty, rozšíří se tabulka pouze o sloupec *k_objekt* následujícím kódem:

```
ALTER TABLE R_STAV_DILY
     ADD   k_objekt INT NOT NULL
```

Uzel Položka rozpočtu

Tabulka *r_etapy_var* obsahuje pouze vazbu na variantu stavby, jelikož se často vyhledávají za celou stavbu (např. výpočet ceny stavby je součet ceny za všechny položky). Tuto tabulku je tedy potřeba rozšířit o sloupec *k_sd*, který bude odkazovat na stavební díl, pod který položka patří. Jelikož položka musí spadat pod stavební díl, nesmí mít pole hodnotu NULL.

Dále stejně jako v případě objektu i položka může mít navazující položku, tedy je potřeba přidat druhý sloupec *k_predch*, který reprezentuje vazbu na předchozí položku. Pokud se nejedná o navazující položku, bude mít pole hodnotu NULL. Jelikož často dochází k vyhledávání položek konkrétného objektu, doporučuji tabulku rozšířit o třetí sloupec *k_objekt*. Tento sloupec reprezentuje objekt, pod který položka přímo spadá (tedy objekt poslední úrovně) a nesmí obsahovat hodnotu NULL. Tabulka se upraví následujícím kódem:

```
ALTER TABLE r_polozky
     ADD   k_sd INT NOT NULL,
           k_predch INT NULL,
           k_objekt INT NOT NULL
```

Uzel Dodatek

Tabulka *r_dodatky* neobsahuje vazbu na objekt, ke kterému je dodatek vázaný, a tedy je potřeba tabulku rozšířit o sloupec *k_objekt*, který jako u předchozích případů nesmí být NULL. Jelikož se s dodatky tako takovými nepracuje příliž často, není potřeba přidávat vazbu na stavbu.

```
ALTER TABLE r_dodatky
     ADD   k_objekt INT NOT NULL
```

Všechny nově přidané sloupce jsou cizí klíče na kód příslušných tabulek, a tedy jsou typu INT.

5. Partitioning

5.1 Definice a využití

Partitioning[4] slouží k rozdělení databázové tabulky na základě určitých kritérií do menších oddílů, které mohou být uloženy na různé disky. Díku tomu je možné pracovat pouze s částí tabulky, která obsahuje relevantní data, a také umožňuje práci více uživatelů paralelně nad danou tabulkou.

Partitioning může být rozdělen na 2 metody: vertikální a horizontální.

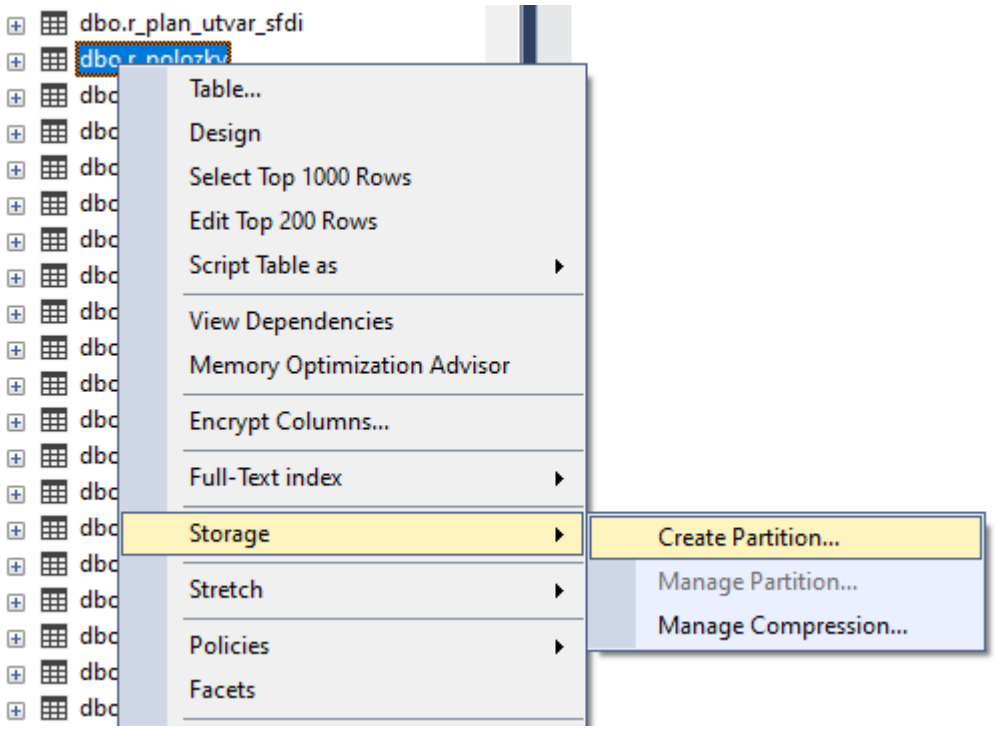
Vertikální partitioning rozděluje sloupce do různých tabulek. Běžně se tato metoda používá u tabulek oddělení statických dat (rychle vyhledávané) a dynamických dat (pomalu vyhledávané).

Horizontální partitioning rozděluje řádky tabulky podle konkrétního sloupce. Tabulka zákazníků tak lze rozdělit na více oddílů, kde každý oddíl bude obsahovat například zákazníky z konkrétního kraje.

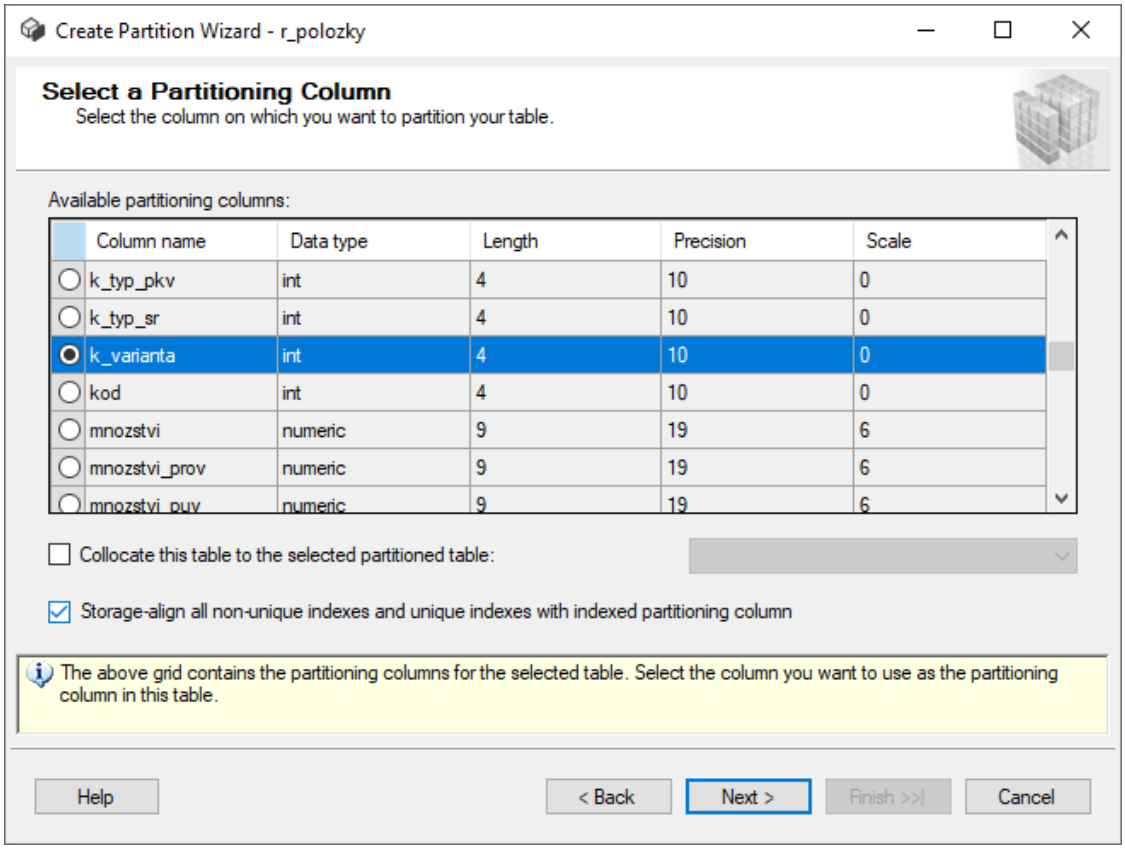
5.2 Aplikace pro databázi Aspe 10

Pro databázi Aspe 10 je vhodné použít horizontální partitioning. Aplikace pracuje vždy nad konkrétní stavbou či prvkem, což znamená rozdělení tabulek *r_polozky*, *r_etapy_var* a *f_faktury* podle sloupce *k_varianta* umožní práci vždy s relevantními daty. Tabulky dodatků a zjišťovacích protokolů není potřeba rozdělit, ovšem tabulky *z_polozky* a *r_polozky_dod* je vhodné rozdělit podle sloupců odkazující je na jejich nadřazený prvek, ke kterému patří, tedy *z_zjist_protokol* a *k_dodatek*.

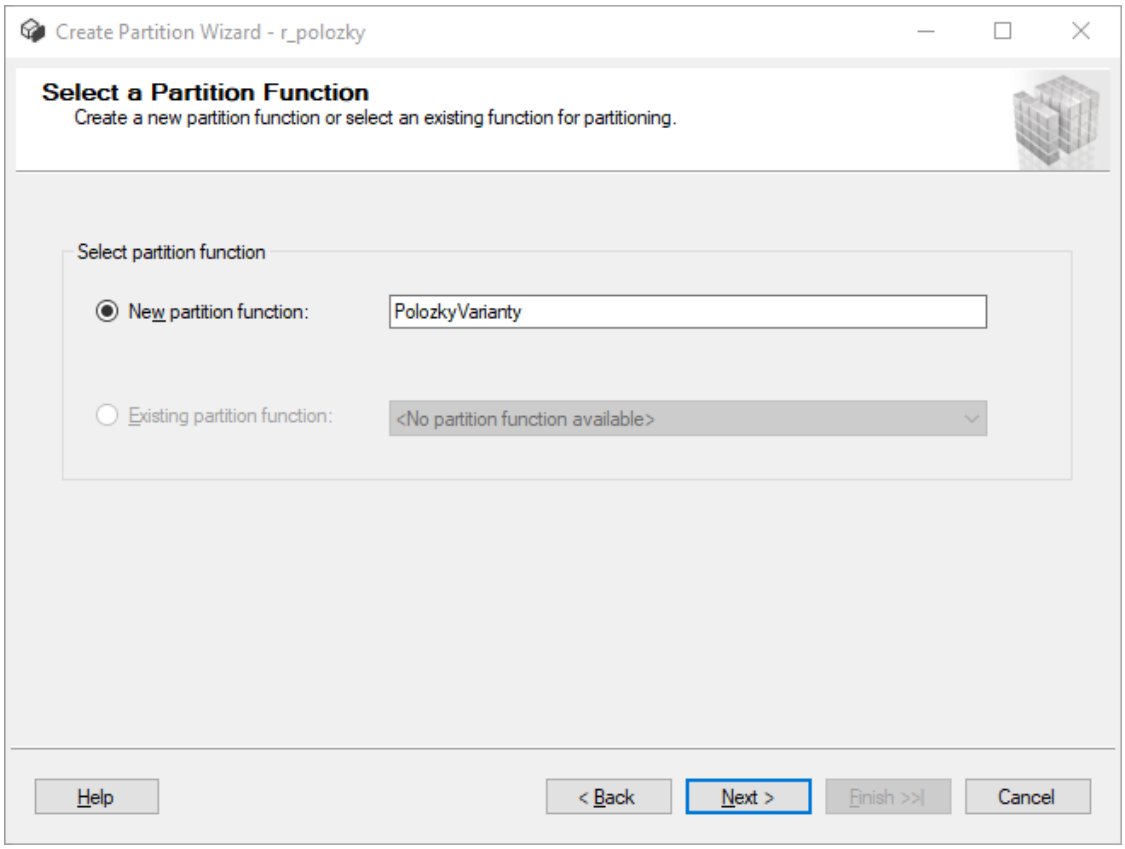
Nejvhodnějším způsobem vytvoření partition tabulek u již existujících dat je využít funkce programu MS SQL Server Management Studio. Wizard pro tuto funkci byl představen v SQL Server 2008. Wizard se otevře po kliknutí pravým tlačítkem myši na konkrétní databázi a výběrem Storage > Create Partition… v kontextovém menu (viz obrázek 6). Podle obrázku 7 se vybere sloupec *k_varianta*. Poté se vytvoří nová partition funkce a schéma (obrázky 8 a 9). Poslední krok nabízí možnost spuštění operace ihned nebo vytvořit script pro pozdější spuštění. V tomto případě se vybere možnost okamžitého spuštění.



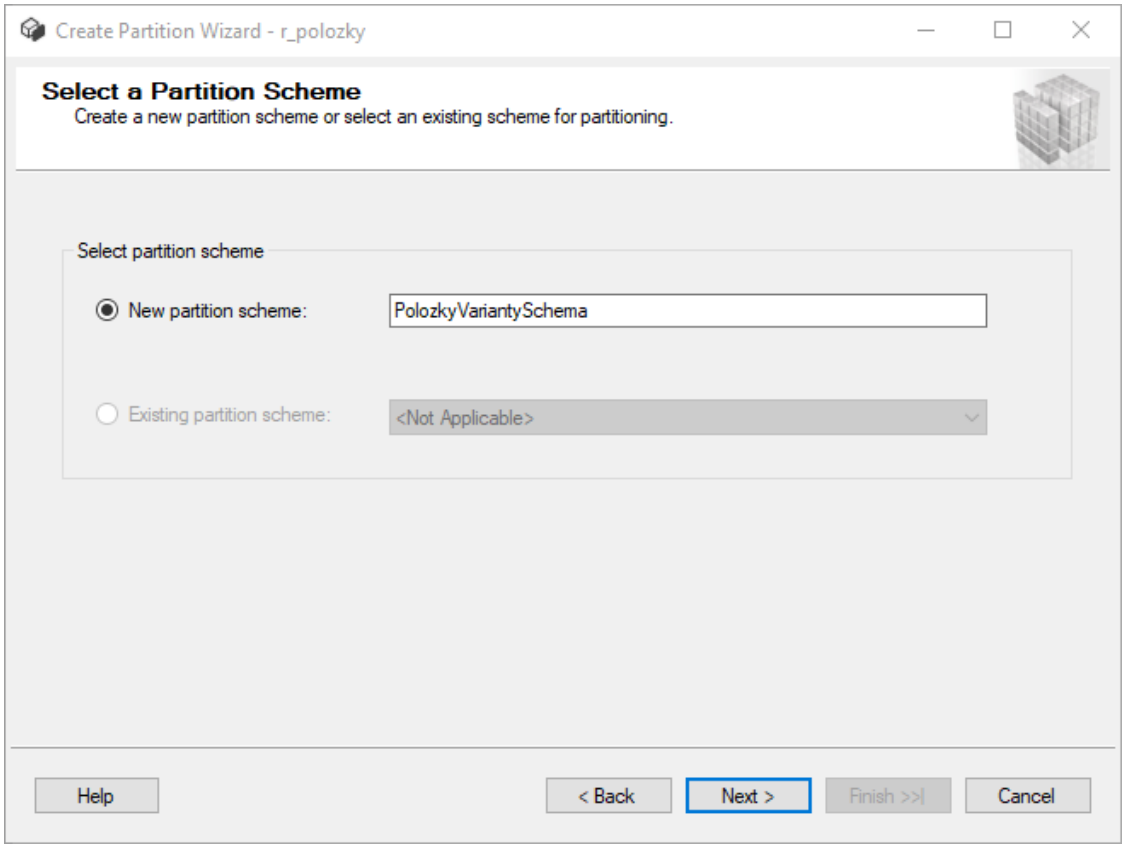
Obrázek 6 - wizard pro vytvoření partition tabulek pro r_polozky (1)



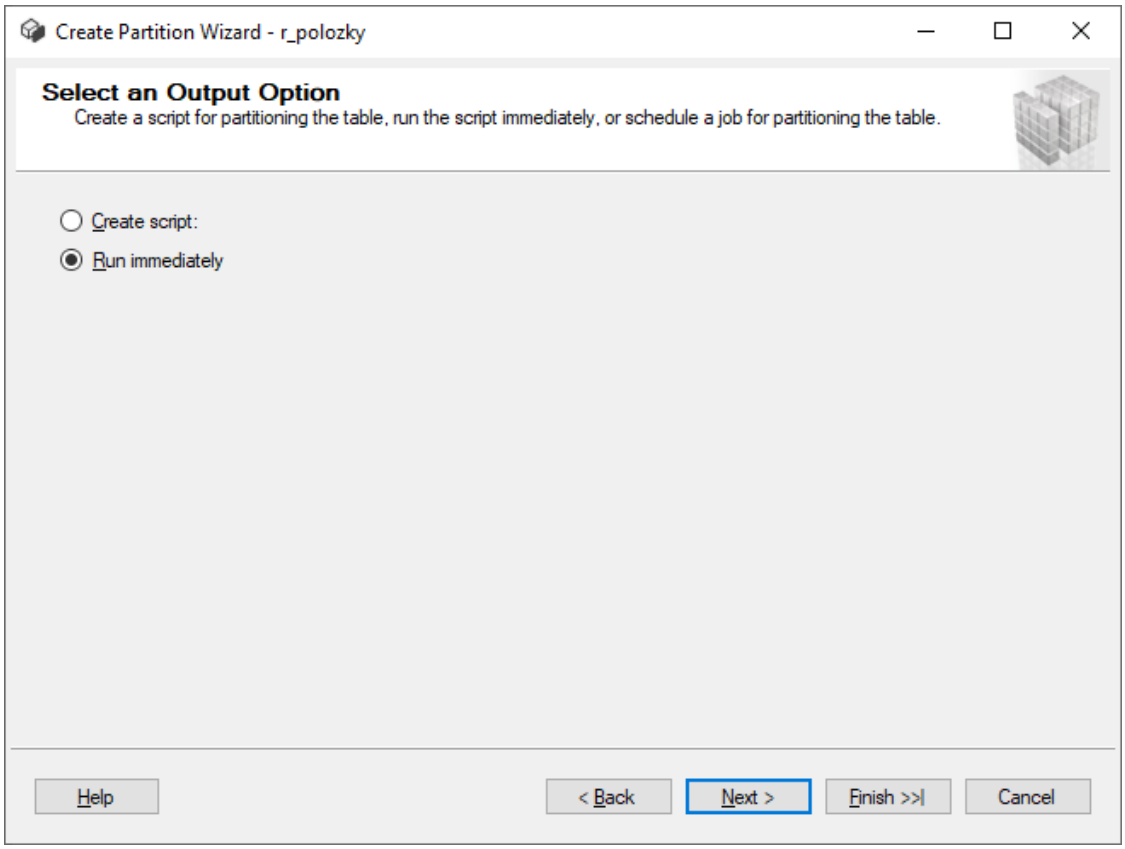
Obrázek 7 - wizard pro vytvoření partition tabulek pro r_polozky (2)



Obrázek 8 - wizard pro vytvoření partition tabulek pro r_polozky (3)



Obrázek 9 - wizard pro vytvoření partition tabulek pro r_polozky (4)



Obrázek 10 - wizard pro vytvoření partition tabulek pro r_polozky (5)

Stejný postup se provede i pro další zmíněné tabulky, u kterých je vhodné vytvořit partition.

6. In-Memory OLTP a databáze

Další možností optimalizace databázového systému je použití In-Memory databáze a tabulek. Pojem In-Memory označuje zpracování dat přímo v operační paměti RAM, čímž se docílí rychlejšího přístupu k datům oproti běžným datovým uložištím. Nevýhodou tohoto systému je volatilita paměti RAM, což při výpadku proudu znamená ztrátu dat.

6.1 In-Memory databáze

In-Memory databáze se nejčastěji využívají u vestavěných systémů, které vyžadují rychlou odezvu. Typickým příkladem jsou telekomunikační zařízení. Oproti běžnému server-klient řešení se databáze vytváří přímo jako součást klientské aplikace. Udržuje v paměti kompletní strukturovanou sadu dat a poskytuje výhody databázového rozhraní jako rychlý přístup k datům pomocí dotazů, kurzory, indexaci dat atp. a poskytuje možnost přistupovat s celkým a komplexním datovým strukturám bez potřeby serveru.

Výhodami In-Memory databáze jsou minimalizace času pro přenos dat z databáze na klienta, absence potřeby fyzického serveru či jiných systémů a aplikací, jednoduchou správu a přenos datové struktury v jednou souboru a také možnost on-disk uložiště, čímž umožňují aplikaci přistupovat k většímu množství dat, než umožňuje kapacita paměti RAM dedikovaná pro aplikaci.

Nevýhody tvoří přístup pouze jednomu uživateli, neexistence stálé verze dat, problémovou správu dat, kdy nelze libovolně přistupovat k datům, a vyšší nároky na systémové prostředky klientské stanice.

6.2 Využití In-Memory databáze v aplikaci Aspe 10

Program Aspe 10 je možné využívat v režimu server-klient i jako lokální řešení (Single). Jak je zmíněno v předchozím odstavci, data In-Memory databáze jsou přístupná pouze jednomu uživateli, tedy je tento systém nepoužitelný v případě server-klient reřimu. Režim Single ale je určen pro práci právě jednoho uživatele, díky čemuž je možné tento systém použít.

Výhodami použití tohoto systému je rychlejší práce s daty, jednodušší zálohování a přenos jednotlivých staveb a je ekonomicky výhodnější pro některé uživatele, neboť není potřeba platit výkonný server.

Hlavní nevýhody z technického hlediska tvoří nutnost vývoje 2 různých způsobů zpracování dat pro oba režimy aplikace a jejich vzájemná kompatibilita a přenositelnost dat a nemožnost připojení dalších aplikací na stejnou databázi. Z uživatelského hlediska pak je toto řešení ekonomicky nevýhodné pro větší počet uživatelů, neboť vyžaduje vyšší nároky na uživatelské stanice.

Použití In-Memory databáze pro optimalizaci sytému je tedy z hlediska vývoje a provozu nevhodné, neboť převažují nevýhody nad výhodami. Nicméně je možné využít In-Memory databáze pro dílčí úlohy, jako je například přecenění stavby. V tomto případě se cenová soustava nahraje do In-Memory databáze pro rychlejší párování položek. Podrobnější analýza použití In-Memory databáze pro dílčí úlohy ale nespadá do rámce této práce.

6.3 In-Memory OLTP

In-Memory OLTP jsou tabulkyuchovávané přímo v paměti RAM pro zvýšení efektivity práce s jejich daty. Jejich předností je optimalizace práce s nimi, což umožňuje efektivnější přístup k datům a vykonávání transakcí a dále odstraňují nutnost uzamykání zdrojů na disku pro současném vykonávání transakcí.

Využívají se 4 typy objektů:

- In-Memory tabulky – tabulky optimalizované pro práci v paměti,
- Non-durable tabulky – náhrada za dočasné tabulky
- Datové typy optimalizované pro práci v paměti
- Nativně zkompilované T-SQL moduly – procedury, triggery a funkce, které jsou zkompilovány při vytvoření

Nevýhodu užití In-Memory OLTP tvoří chybějící podpora DLL triggerů a užití cizích klíčů v T-SQL.

6.4 Využití In-Memory OLTP v aplikaci Aspe 10

In-Memory OLTP je chodné použít pro paralelní zpracování velkého množství dotazů. Aplikace Aspe 10 obsahuje modul Syncho, který slouží pro editaci dat reálném čase. Pro tento modul je vhodné využít In-Memory tabulky, které minimalizují latenci editace dat a zaručí tak práci nad aktuálními daty. Problém ale nastává v podpoře cizích klíčů. Jelikož nejsou podporovány, musela by většina datového jádra databáze být vytvořena jako In-Memory tabulky. Z hledisla paměťové náročnosti a chybějící podpory triggerů u T-SQL toto není vhodné řešení. Reálnější užití je v dílčích částech programu, jako jsou například cenové soustavy, které jsou oddělené od datového jádra databáze.

Mimo tabulek se nabízí jako vhodné využití u často používaných procedur pomocí kompilovaných modulů. U stávajících dat je to například načítání tabulky *r_strom* či funkce pro přepočet ceny.

Další použití je nahrazení dočasných tabulek za non-durable tabulky. Zrychlily by se tím pomocné tabulky pro párování či filtrování záznamů v procedurách, a navíc by práce s nimi nezahlcovala IO logy.

7. Indexy

Smysl indexů je urychlit čas při vyhledávání záznamů serverem. Z toho důvodu se indexy vytváří pro všechny sloupce tabulky, podle kterých se vyhledává, třídí či se tabulky spojují. Běžně se data ukládají do databáze tak, v jakém pořadí jsou vloženy, což znamená, že tabulka není nijak organizovaná. Vhodně vytvoření index tak urychlí vyhledání nejen konkrétního záznamu, ale také hodnot v určitém rozsahu, neboť není potřeba procházet všechna data. Setřídění dat také umožňuje rychlé vyhledání číselných hodnot jako jsou minima a maxima nebo vyhledávání řetězců pomocí operace *LIKE.*

Vytváření indexů a jejich údržba s sebou přináší i úskalí. S každou změnou v tabulce (*UPDATE* či *INSERT*) je potřeba provést aktualizaci indexu. Z toho důvodu není vhodné vytvářet indexy u tabulek, ze kterých se zřídka čte a často se do nich zapisuje. Typickým případem jsou logy. Dále by se nemělo vytvářet příliš mnoho indexů. Nejen že indexy zvyšují nároky databázového serveru na operační paměť a diskový prostor, databázový server sám řídí, který index pro vykonání dotazu použije. Tím hrozí možnost volby špatného (neoptimálního) indexu.

Indexy se implementují pomocí datové struktury B-strom. Každý vrchol stromu obsahuje od t-1 do 2t-1 prvků, kde t reprezentuje faktor stromu. Každý vrchol odkazuje na n(x)+1 následujících vrcholů (listů). Oboustranné omezení a stejná délka od kořene k listům zaručují rychlost orperace vyhledávání. Oboustranné omezení zařizuje, že ukazatel na prvky vlevo ukazuje na prvky s menší hodnotou a pravý ukazatel zase na prvky s vyšší hodnotou, než je daný prvek vrcholu.

7.1 Indexy v databázi Aspe 10

Indexy v databázi Aspe 10 tvoří problém. Byly původně navrženy při vzniku samotné databáze (rok 2016 a některé i starší) a od té doby došlo k mnoha rozšířením tabulek a obecně změnám v databázové struktury. To má za následek, že se v databázi vyskytuje mnoho starých a již nevhodných indexů.

Správu indexů lze aktuálně považovat za minimální. Při aktualizaci aplikace dojde k obnově statistik a indexů, ale uživatel může pracovat na jedné verzi měsíce, než provede aktualizaci. Tím dochází k omezení rychlosti aplikace. Tento případ byl vypozorován u tiskových sestav. Zatímco s nově vytvořenými indexy trvá vytvoření

tiskové sestavy několik vteřin, neudržované indexy na stejných datech způsobí chybu aplikace, kdy vyprčí časová platnost pro operaci (timeout na desítky minut). Z toho důvodu je žádoucí vytvořit časový interval, ve kterém se budou obnovovat indexy.

Pro údržbu nejen indexů je možné použít volně dostupného scriptu, jehož autorem je Ola Hallengren[6].

7.2 Database Engine Tuning Advisor

Další možností pro optimalizaci indexů databáze aplikace Aspe 10 je využít programu Database Engine Tuning Advisor od společnosti Microsoft. Database Engine Tuning Advisor prozkoumá, jak jsou dotazy dané databáze zpracované a na základě těchto dat doporučí, jakým způsobem lze databázi vylepšit. Kromě indexů doporučí, jak rozdělit tabulky do partitionů nebo jak vylepšit výkon zpracování dotazů.

Vstupními daty pro tento program je například výstup logu z programu SQL Server Profiler. Profiler zaznamenává jednotlivé operace prováděné nad databází na základě parametrů nastavených u tzv. tracu. Díky tomuto programu lze sledovat aktuální operace nad databází, jaký uživatel je vykonává, kolik času operace trvala a podobně. Tento log pak slouží jako vstup pro program Tuning Advisor pro analýzu a vyhodnocení vhodných postupů pro zlepšení výkonu databázového systému.

Místo úpravy stávajících indexů, které jsou neaktuální, je nejvhodnějším postupem odstranit všechny indexy, spustit Profiler a několik hodin pracovat v aplikaci Aspe 10. Práce vyžaduje zaměření na celou aplikaci, a ne jenom konkrétní sekce. Tím se zajistí trace log s dostatečným množstvím informací pro vytvoření nových indexů za pomoci programu Tuning Advisor.

Tuning Advisor není podporován na Express edici SQL serveru a vyžaduje SQL server 2019 (verze 15.x)[4]. Z licenčních důvodů není možné tuto variantu vyzkoušet.

8. Statistiky

Zatímco indexy vytváří programátor sám, statistiky jsou zcela v režii SQL serveru. Server si udržuje statistiky nad jednotlivými tabulkami, ve kterých jsou informace o zastoupení různých hodnot a jejich množství v polích. To je vhodné například pro vyhledávání v tabulce, kde je velké množství stejných hodnot. Jelikož SQL server zná poměr hledaných hodnot, vyhodnotí vhodný postup vykonání dotazu, tedy jestli se provede scan tabulky nebo se použije index.

Statistiky se automaticky vytváří při tvorbě indexu k jeho prvnímu sloupci a je možné je následně rozšířit o další sloupce indexu. Aktualizace statistik je provádí ručně spuštěním příkazu *update statistics*. Pokud je vyžadováno vytvářet vlastní upravené statistiky, je nutné při každém založení nového indexu spustit uložený script s požadovaným nastavením pro dodatečnou aktualizaci nově vytvořených statistik nebo pro poižít podmínku *with statistics using 0 values*.

Jelikož se statistiky vytváří synchronně a nad každým vyhledávaným sloupcem v dotazu (pokud neexistují), způsobují tak zpomalení systému. Z toho důvodu je doporučené mít statistiky vytvořené před zahájením samostatné práce.

8.1 Statistiky v databázi Aspe 10

Problém se starými neaktualizovanými indexy se přenáší i do statistik. Každý takový index má vytvořené statistiky, které mohou negativně ovlivnit vyhodnocení optimizéru pro vhodný execution plan. Z toho důvodu je vhodné provádět pravidelnou údržbu statistik obzvlášť u tabulek, kde dochází k častým změnám.

9. Závěr

Tato práce byla původně myšlena pro optimalizaci databáze jiného produktu. Bohužel vlivem změny firemní politiky v průběhu práce bylo vyžadováno od původní databáze upustit, a tak se jako náhrada zvolila databáze produktu Aspe 10. Vlivem této náhlé změny došlo odbočení od původního zadání a práce se místo konkrétních dotazů zaměřila na možnosti optimalizací, které by bylo možné přenést do nově vyvíjeného systému.

V této práci jsou uvedené optimalizace několika vybraných funkcí, které byly vybrány na základě výstupu z programu SQL Server Profiler. Po dlouhodobé práci v programu Aspe 10 se tyto funkce projevily jako velmi často používané. Společně s nimi byla analyzovaná vazební tabulka *r_strom*, která zpomaluje chod databáze, a byl navržen systém její náhrady. Jelikož systém vyžaduje velký zásah ve zdrojovém kódu aplikace, nebude s největší pravděpodobností tato úprava zanesena do programu.

Dále se práce zabývá využitím dělení tabulek na partitiony na základě vhodných sloupců. Příkladem je tabulka položek rozpočtu a její dělení na základě cizího klíče varianty stavby.

V poslední řadě se teoreticky zaobírá aktualizací a údržbou statistik a indexů a možnosti využití In-Memory technologií. Tyto technologie přinášejí velké množství komplikací z důvodu absence podpory cizých klíčů, ale bylo by možné je využít pro některé samostatné moduly, a nebo pro práci s 3D modely do nově vyvíjeného systému.

Seznam zdrojů

[1] ZANIOLO, Carlo. *Advanced database systems*. San Francisco: Morgan Kaufmann Publishers, 1997. The Morgan Kaufmann series in data management systems. ISBN 1-55860-443-X.

[2] STANCZYK, Stefan K. *Theory and practice od relational databases*. London: UCL Press, 1990. ISBN 1-85728-232-9.

[3] STARKS, Joy L., Philip J. PRATT a Mary Z. LAST. *Concepts of database management*. Ninth Edition. Australia: Cengage, [2019]. ISBN 978-1-337-09342-2.

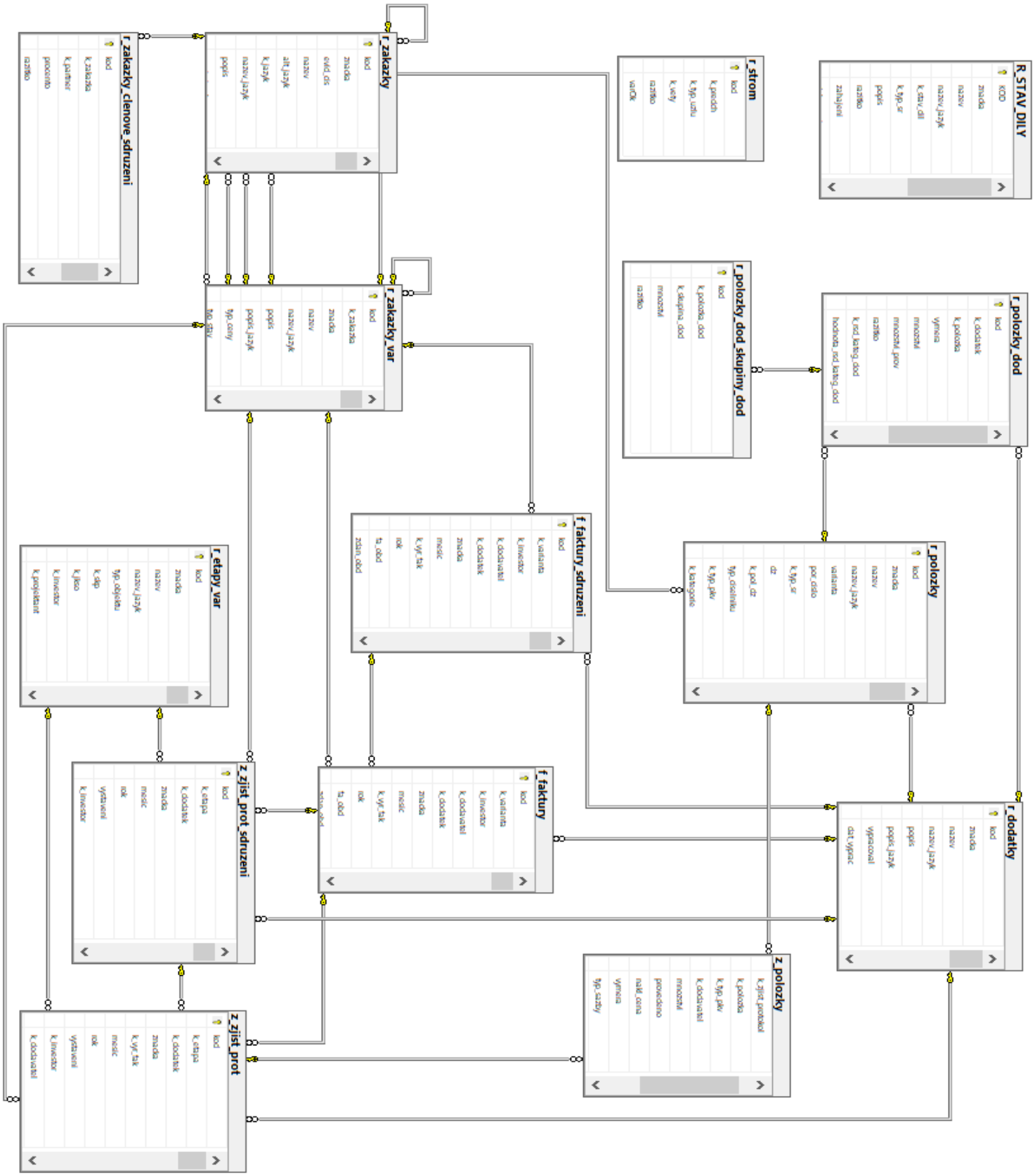
[4] Database Features - SQL Server 2014 | *Microsoft Docs* [online]. Redmond (Washington): Microsoft, 2020 [cit. 2020-04-28]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/database-features?view=sql-server-2014

[5] Overview and Usage Scenarios - SQL Server | *Microsoft Docs* [online]. Redmond (Washington): Microsoft, 2020 [cit. 2020-04-28]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/overview-and-usage-scenarios?view=sql-server-ver15

[6] SQL Server Index and Statistics Maintenance. *SQL Server Maintenance Solution* [online]. Landskrona: Ola Hallengren, 2020 [cit. 2020-05-07]. Dostupné z: https://ola.hallengren.com/sql-server-index-and-statistics-maintenance.html

[7] Tutorial: Database Engine Tuning Advisor - SQL Server | Microsoft Docs. *Microsoft Docs* [online]. Redmond (Washington): Microsoft, 2020 [cit. 2020-05-07]. Dostupné z: https://docs.microsoft.com/en-us/sql/tools/dta/tutorial-database-engine-tuning-advisor?view=sql-server-ver15

Příloha A


r_zakazky_clenove_sdruzeni
r_zakazky
r_strom
R_STAV_DILY
r_zakazky_var
r_polozky_dod_skupiny_dod
r_polozky_dod
f_faktury_sdruzeni
r_polozky
r_etapy_var
r_dodatky
z_zjist_prot_sdruzeni
f_faktury
z_polozky
z_zjist_prot