# Optimalizace výkonnosti DB2 databáze

Bc. Ladislav Stonawský

Diplomová práce
2015

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2014/2015

# ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Ladislav Stonawský** |
| Osobní číslo: | **A13505** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Informační technologie** |
| Forma studia: | **kombinovaná** |
| Téma práce: | **Optimalizace výkonnosti DB2 databáze** |
| Téma anglicky: | **DB2 Performance Tuning** |

Zásady pro vypracování:

1. **Analyzujte a popište základní databázové mechanismy, jejich funkce a důležitost při ladění.**
2. **Zabývejte se samotnou architekturou databáze, tabulkového prostoru, tabulek, indexů a správou paměti.**
3. **Realizujte použití nových postupů k ladění výkonnosti databáze optimalizací dotazů v jazyce SQL.**
4. **Proveďte vyhodnocení efektivity nové použitých postupů.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:


1. **CHEN, Whei-Jen et al. High Availability and Disaster Recovery Options for DB2 on Linux, Unix, and Windows. United States: IBM, International Technical Support Organization, 2009, 856 s. ISBN 07-384-3138-9.**
2. **CHEN, Whei-Jen. DB2 integrated cluster environment deployment guide. 1st ed. San Jose, Calif.: International Technical Support Organization, c2004, 408 s. IBM redbooks. ISBN 07-384-9082-2.**
3. **ARNOLD, Danny et al. Unleashing DB2 10 for Linux, UNIX, and Windows. United States: IBM, International Technical Support Organization, 2012, 162 s. ISBN 07-384-3710-7.**
4. **CHEN, Whei-Jen et al. Leveraging DB2 10 for High Performance of Your Data Warehouse. United States: IBM, International Technical Support Organization, 2014, 218 s. ISBN 07-384-3897-9.**
5. **IBM DB2 10.5 for Linux, UNIX, and Windows: Troubleshooting and Tuning Database Performance. United States, 2015, 793 s.**
6. **CHEN, Whei-Jen. Database partitioning, table partitioning, and MDC for DB2 9. 1st ed. United States: IBM, International Technical Support Organization, 2007, 252 s. ISBN 07-384-8922-0.**
7. **NEAGU, Adrian a Robert PELLETIER. IBM DB2 9.7 Advanced Administration Cookbook. United States: Packt Publishing, 2012, 480 s. ISBN 9781849683326.**


Vedoucí diplomové práce: **doc. Ing. Zdenka Prokopová, CSc.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **6. února 2015**

Termín odevzdání diplomové práce: **15. května 2015**

Ve Zlíně dne 6. února 2015

L.S.

doc. Mgr. Milan Adámek, Ph.D.
*děkan*

doc. Mgr. Roman Jašek, Ph.D.
*ředitel ústavu*

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně 15. 5. 2015

## ABSTRAKT

Diplomová práce se zabývá laděním výkonu DB2 LUW databází. V teoretické části je uveden popis struktury databáze, popis objektů databáze, jejich vzájemné relace a využití při ladění. Dále jsou představeny jednotlivé parametry databáze a jejich vliv na výkon aplikací a databázových funkcí.

Praktická část se věnuje identifikaci slabých míst SQL dotazů, jejich optimalizaci pomocí indexů a nahrazení neefektivních operátorů. Následně jsou zkoumány nároky databáze na operační paměť a testování jednotlivých parametrů. Poslední část se zaměřuje na nový přístup k ladění databází DB2.



Klíčová slova: DB2, SQL, databáze, ladění, BLU akcelerace, index, tabulkový prostor, bufferpool, efektivita



## ABSTRACT

This thesis is concerned with DB2 LUW database performance tuning. The theoretical part reflects a description of the database structure, description of database objects and their relationship to each other use for debugging. It presents individual database parameters and their impact on application performance and database functions.

The practical part is focused on identifying SQL query vulnerabilities, optimization using indexes and replacing inefficient operators. It examines the demands on database memory and is testing various parameters. The last section focuses on a new approach of tuning DB2 databases.



Keywords: DB2, SQL, database, performance tunning, BLU acceleration, index, tablespace, bufferpool, efficiency

*„Přičiň se, abys byl opravdu tím, čím se chceš stát.“*

J. W. Goethe

## Poděkování

Touto cestou bych rád poděkoval doc. Ing. Zdence Prokopové, CSc., vedoucí mé práce, za odborné vedení, cenné rady a podněty, které mi poskytla při zpracování této práce. V neposlední řadě bych chtěl poděkovat své rodině a manželce za trpělivost, zázemí a podporu v průběhu celého mého studia.

# OBSAH

# ÚVOD

DB2 je jedním z předních komerčních databázových systémů vyvíjený nadnárodní korporací IBM. Původně se název produktu skrýval za akronymem UDB (Universal database), ale později byl především z marketingových důvodů změněn na dnes užívané DB2. Tento multiplatformní systém relačních databází je podporován v širokém spektru operačních systémů: DB2 pro iSeries, DB2 mainframe pro servery z/OS a middleware DB2 LUW (Linux, Unix, Windows). V současnosti je tento databázový systém podporován řadou Linuxových a Unixových distribucí, mezi nimiž lze jmenovat např. Red Hat, SUSE, AIX, HP/UX či Solaris (IBM Knowledge Center, 2015).

První verze DB2 pro LUW v5 byla vydána 27. června 1997, kdy přinesla v tehdejší době inovativní zavedení BLOB objektů. Mezi dnešní oficiálně podporované verze řadíme v9.7 (Cobra), v10.1 (Galileo) a nejnovější v10.5 (Kepler), který s sebou přináší evoluční metodu ukládání dat zaměřenou na databáze typu warehouse (Gandhi , 2011).

Singulární verze jsou licenčně omezeny. Jednotlivé licence se liší množstvím podporovaných služeb a jsou rozdílně nákladné na finanční zdroje. Korporace IBM nabízí k vyzkoušení volnou verzi DB2 Express-C podporující pouze základní služby. Pro využití v typicky neprodukčním prostředí je doporučeno užití licence Workgroup Server Edition. Naproti tomu v produkčním prostředí na rozsáhlých databázových systémech, kde je třeba využít veškeré služby, které DB2 nabízí, je nutné licencovat produkt jako Advanced Enterprise Server Edition.

S licencí jsou úzce spjaty taktéž možnosti ladění výkonu databází, který rovněž ovlivňuje hardwarová konfigurace serveru a efektivnost kódu aplikací. Každý z nás v současné moderní společnosti využívá databáze nevědomky téměř každodenně. Malé ale i velké společnosti jsou existenčně závislé na svých databázových systémech, neboť jsou neoddělitelnou součástí jejich podnikání. Jakýkoliv neefektivní systém či postup negativně ovlivňuje celkovou funkci firmy a snižuje tak její konkurenceschopnost na trhu.

Optimální výkon databází je jedním ze základních stavebních kamenů úspěšného podnikání. Pokud databáze tuto úlohu nesplňuje, je v první řadě nutné určit, jakou databáze plní funkci. Posléze můžeme stanovit postupy diagnostiky současných

slabých míst databáze a hledat způsoby co nejefektivnější nápravy. Naštěstí DB2 poskytuje velké množství funkcí a služeb, díky kterým můžeme účinně ladit výkon jejích databází.

# I. TEORETICKÁ ČÁST

# 1 FAKTORY OVLIVŇUJÍCÍ VÝKONNOST DATABÁZE DB2

Existuje nepřeberné množství faktorů, které mají vliv na výkonnost databází. Ačkoliv databázový systém není schopen odstranit všechny elementy s negativním dopadem na výkon databáze, může do jisté míry přizpůsobit své parametry, jenž povedou k efektivnějším odezvám na dotazy, zpracování transakcí aplikacemi a využití zdrojů serveru. Proces zlepšování výkonnosti databáze je iterativním hledáním příčin a jejich řešení.

Obecný postup ladění výkonu je definován následujícími kroky:

1. Definování problému.
2. Stanovení hlavních ukazatelů slabých míst v systému.
3. Vypracování a provedení plánu sledování výkonnosti.
4. Průběžné sledování výsledků a identifikace negativních slabých míst.
5. Provedení změn k eliminaci slabých míst (IBM, 2015a).

Mezi všeobecné faktory ovlivňující výkon patří konstrukce databáze, správa alokované paměti, aplikace a jejich SQL dotazy a funkce pravidelné údržby databáze.

## 1.1 Konstrukce databáze

Databázové objekty a vztahy mezi nimi tvoří konstrukci relační databáze. Vhodně definované objekty, relace ale i omezení pozitivně ovlivňují výkon celé databáze. Autoři Administration Guide pro DB2 (columbia.edu, 2015) upozorňují, že konstrukce databáze nezůstává neměnná, mění se s časem (např. během vytváření nových tabulkových prostorů, tabulek nebo indexů). Z tohoto důvodu je nutné mít na zřetel konstrukci po celou dobu existence databáze.

### 1.1.1 Schémata

Schéma je logickým sdružením objektů, kterými jsou tabulky, pohledy, přezdívky, triggery, funkce, balíčky, atd. Vytváří se pomocí příkazu CREATE SCHEMA. Schéma umožňuje pojmenovávat databázové objekty stejným názvem, jenž je pro ně typický, aniž by docházelo k dvojznačným odkazům. Pro ilustraci lze uvést názvy schémat „firemní“ a „soukromé“. Databázový objekt „výdaje“ je možné pod každým schématem pojmenovat stejně: „firemní výdaje“ a „soukromé výdaje“.

### 1.1.2 Tabulky

Základním stavebním kamenem databází jsou tabulky, které jsou definovány řádky a sloupci. Řádek je sekvencí datových různých typů s interním identifikátorem RID. Skutečné rozmístění řádků v tabulce nelze ovlivnit, protože o něm rozhoduje DBM.

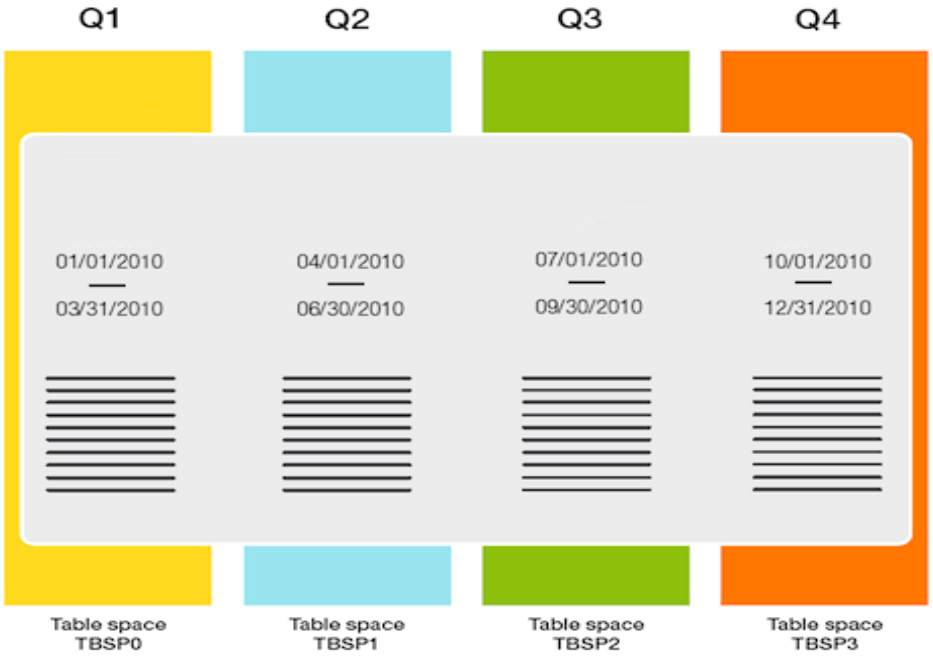
#### *1.1.2.1 Range-partitioned tabulky*

Range-partitioned tabulky se nejběžněji vyskytují u databáze typu warehouse. Mnohokrát se jedná o tabulky obsahující miliony řádků a o velikosti desítek GB. Tyto tabulky jsou rozděleny mezi jednotlivé kontejnery na základě hodnot v jednom nebo více sloupcích. Za typický příklad bývá dle Chena a kol (2014) uváděno rozdělení tabulky na oddíly na základě časového razítka zápisu transakce. Rozdělí-li se tabulka na 4 oddíly, z nichž bude každý reprezentovat jeden kvartál roku, bude v případě dotazu týkajícího se konkrétního měsíce načten pouze jeden oddíl celé tabulky, jak je doloženo na obr. 1.

```
CREATE TABLE INCOME(datum_transakce date, mesic int NOT NULL
GENERATED ALWAYS AS (month(datum_transakce)), rok INT NOT
NULL GENERATED ALWAYS AS ( year(datum_transakce)),
polozka int NOT NULL,
zakaznik int NOT NULL) PARTITION BY RANGE (rok,mesic)
(
PART Q1 STARTING (2010,1) ENDING (2010, 3) INCLUSIVE,
PART Q2 ENDING (2010, 6) INCLUSIVE,
PART Q3 ENDING (2010, 9) INCLUSIVE,
PART Q4 ENDING (2010,12) INCLUSIVE,
PART CURRENT ENDING (MAXVALUE, MAXVALUE)
);
```

*Obr. 1. Příklad vytvoření range-partitioned tabulky (dle Chen, 2014 upravil Stonawský, 2015)*

Každý oddíl range-partitioned tabulky lze taktéž umístit do jiného tabulkového prostoru (viz obr. 2). Výhoda tohoto typu tabulky spočívá v rozdělení kontejnerů tabulkových prostorů mezi jednotlivá fyzická úložiště, kde oddíly s větší prioritou a vyšším počtem transakcí umisťujeme na výkonnější úložiště (Chen, 2014).




*Obr. 2. Grafické rozdělení range-partitioned tabulky mezi tabulkové prostory (dle Chen, 2014 upravil Stonawský, 2015)*

#### *1.1.2.2 MQT tabulky*

MQT tabulky definuje Gandhi (2013) jako mezistupeň pohledu (view) a regulární tabulky. Na rozdíl od pohledů lze na MQT tabulkách vytvářet indexy nebo spouštět RUNSTATS pro získání nových statistik. Tyto tabulky se využívají v případě komplexních SQL dotazů, kdy DB2 optimalizátor vyhodnotí jako efektivnější použít tabulku MQT, na které je většina propočtů pro opakující se dotazy provedena jen jednou. Samotné vytvoření MQT tabulky má negativní dopad na diskovou kapacitu. Podobně jako u indexů se jedná o čistou redundanci dat za účelem dosažení vyššího výkonu databáze.

```
create table emp as (
select e.empno, e.firstnme, e.lastname, e.phoneno,
d.deptno,substr(d.deptname, 1, 12) as department, d.mgrno
from employee e, department d
where e.workdept = d.deptno
                    )
data initially deferred refresh immediate;
```

*Obr. 3. Příklad vytvoření MQT tabulky (Stonawský, 2015)*

### 1.1.3 Sekvence

Sekvence je databázový objekt, který umožňuje automatické generování hodnot, jako jsou např. kontrolní čísla. Vytváří se pomocí příkazu CREATE SEQUENCE. Sekvence se ideálně hodí k úkolu generování hodnot unikátního klíče (IBM, 2014a). Jeho nespornou výhodou je, že aplikace mohou k sekvenci přistupovat souběžně bez snížení výkonu, který je spojen s jinými metodami generování unikátních hodnot.

```
CREATE SEQUENCE "TEST"."SAMPSEQUENCE" AS BIGINT
MINVALUE 1 MAXVALUE 9223372036854775807
START WITH 1 INCREMENT BY 1
CACHE 20 NO CYCLE NO ORDER;
```

*Obr. 4. Příklad vytvoření sekvence (Stonawský, 2015)*

### 1.1.4 Constraints - omezení

V databázi je často nutné dodržovat určitá omezení nebo pravidla (např. rodná čísla obyvatel musí být jedinečná). Omezení poskytují správci databáze způsob, jak tato pravidla prosadit.

Omezení jsou jednou z definicí tabulek. Lze je specifikovat v příkazu CREATE TABLE při vytváření nové tabulky. Omezení stávajících tabulek se upravují pomocí příkazu ALTER TABLE. Stávající omezení je možné rovněž kdykoliv zrušit, aniž by došlo k ovlivnění struktury tabulky nebo dat v nich uložených.

Chen a kol. (2004) rozlišuje u DB2 5 druhů omezení:

- *Nenulové omezení* – pravidlo, které zabraňuje vkládat hodnoty null do jednoho nebo více sloupců v tabulce.
- *Unikátní omezení* (označovaný také jako unikátní klíč) – pravidlo, které zakazuje duplicitní hodnoty v jednom nebo více sloupců v tabulce.
- *Primární klíč* – sloupec nebo kombinace sloupců, které má stejné vlastnosti jako unikátní omezení. Pomocí primárního a cizího klíče jsou také definovány vztahy mezi jednotlivými tabulkami.
- *Cizí klíč* – logické pravidlo o vztahu mezi tabulkami. Hhodnota ve sloupci jedné tabulky musí odpovídat hodnotě ve sloupci druhé tabulky. Pomocí

cizího klíče lze definovat akce, které nastanou při mazání nebo změně záznamů.

- Cascade – v dceřiné tabulce se změní hodnoty stejným způsobem jako v rodičovské.
- Set null – reference se při změně nastaví na hodnotu NULL.
- Set default - reference se při změně nastaví na výchozí hodnotu.
- Restrict – záznam nelze z mateřské tabulky vymazat v případě, že je i v tabulce dceřiné.
- No action – je podobné akci restrict. Rozdíl je v tom, že ke kontrole referenční integrity dochází až po změně hodnot. Pokud změna neprojde kontrolou, dojde v obou případech k zobrazení chybové hlášky.

- *Tabulkové omezení* – brání ve vkládání různých dat do sloupců v tabulce. Např. do sloupce plat zaměstnance nelze vložit hodnotu nižší než 9 000 Kč.

### 1.1.5 Indexy

Index je jedním z objektů v databázi. Je tvořen pomocí příkazu CREATE INDEX. Proces vytvoření indexu vyžaduje volné místo ve fyzickém úložišti. Dochází k produkci nových datových struktur, odkazujících na tabulku, avšak nedochází ke změně dat tabulky. IBM Knowledge Center (2015) připodobňuje index databáze k indexu na konci knihy. Zabírá svůj vlastní prostor, je vysoce redundantní a odkazuje se na skutečné informace uložené na jiném místě. Po vytvoření indexu je nutné shromáždit aktuální statistická data pro jednotlivou tabulku pomocí funkce RUNSTATS.

Indexy jsou hlavně využívány pro zlepšení výkonu při přístupu k datům v tabulce. Neexistuje-li index pro danou tabulku, musí být skenována celá tabulka, na kterou se odkazuje dotaz SQL. Čím větší daná tabulka je, tím déle trvá její skenování, neboť prohledávání tabulky vyžaduje, aby byl postupně přístupný každý její řádek.

Zatímco skenování celé tabulky může být efektivnější pro komplexnější dotazy, které vyžadují většinu řádků v tabulce, skenování indexu je účinnější v případě dotazování se aplikace na některé konkrétní řádky tabulky.

Chainani a kol. (2013) uvádí, že v tomto případě optimalizační proces DB2 upřednostní před prohledáváním celé tabulky skenování indexu. Indexové soubory jsou obecně menší a vyžadují méně času na čtení, než celé tabulky, zejména pokud se tabulky podstatně zvětšují.

Každý nový vložený řádek do tabulky musí DB2 transponovat i do indexu. Zformujeme-li index na každém sloupci tabulky, bude dosaženo rychlého čtení, ale rychlost zápisu a velikost tabulky se může i více než zdvojnásobit.

Základní typy indexů (Chainani a kol, 2013):

- *Unikátní indexy* – vynucují si, aby hodnoty ve sloupci nebo více sloupcích byly jedinečné. Je to také primární klíč. Tyto klíče jsou vytvořeny pomocí příkazu CREATE TABLE nebo ALTER TABLE.
- *Cluster indexy* – definují se jako takové, kde jsou skutečná data v tabulce umístěna alespoň zhruba ve stejném pořadí jako v indexu. Jsou tvořeny pomocí příkazu CREATE INDEX s atributem CLUSTER. Pokud cluster index již v tabulce existuje, DB2 se pokusí vložit data do tabulky dle pořadí v indexu. Správné pořadí ale není zaručeno a může se s vývojem času v závislosti na objemu vložených dat zhoršovat. Tabulky nelze reorganizovat dle indexu, není-li v nich žádný cluster index, přičemž na jedné tabulce může existovat pouze jeden takový index. Pro některé SQL dotazy umožňují zrychlit přístup k datům tabulky, protože zajišťuje lineární přístup k datům, která jsou uložena na jednotlivých stránkách tabulkových prostorů, díky rychlejšímu vkládání do dočasných tabulek.
- *Obousměrné indexy* – od verze DB2 v9.1 se indexy vytváří ve výchozím nastavení s klauzulí ALLOW REVERSE SCANS. DB2 tak může číst indexy v obou směrech a usnadňuje tak použití MAX a MIN, eliminuje nutnost vytvářet dočasné tabulky pro zpětné čtení a také redundantní zpětné čtení indexu.

Údržbu indexů provádí DB2 pomocí asynchronního čištění, jenž je součástí operace REORG. Asynchronní čištění indexů (AIC) je odložená operace označující nevalidní záznamy indexu. SQL dotazy ignorují tyto položky a později jsou odstraněny. Dojde-li k deaktivaci databáze během čištění, po opětovné aktivaci na něj DB2 opět naváže. Proces čištění zamyká tabulkové prostory zámkem IX

a tabulky zámkem IS. Tyto zámky ovšem odstraní v případě, že jakákoliv aplikace čeká na jejich zrušení. Poté se doba čištění zastaví na 5 minut.

Údržbu indexů je možné sledovat pomocí příkazu LIST UTILITES znázorněné na obr. 5 níže.

```
ID                                   = 1
Type                                 = ASYNCHRONOUS INDEX CLEANUP
Database Name                        = SAMPLE
Partition Number                     = 0
Description                          = Table: DB2.PROJNO, Index:
DB2.I1
Start Time                           = 12/3/2015 10:11:01.978554
State                                = Executing
Invocation Type                      = Automatic
Throttling:
   Priority                          = 50
Progress Monitoring:
      Total Work                     = 8 pages
      Completed Work                 = 0 pages
      Start Time                     = 12/3/2015 10:11:01.982354
```

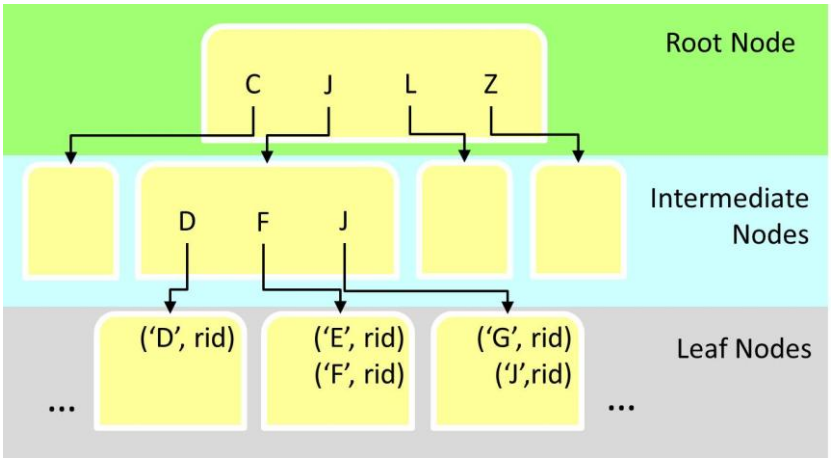
*Obr. 5. Průběh čištění indexů (Stonawský, 2015)*

Čištění indexů je důležitým aspektem pro udržení výkonnosti databáze. DB2 se tak zbavuje ukazatelů v leaf node odkazujících na prázdné řádky v tabulce.

Součástí REORG operace je taktéž defragmentace indexů. V případě vymazání záznamu v leaf node B-stromu, nedojde automaticky k uvolnění alokovaného místa, které index fyzicky zabírá v úložišti. Pro dosažení optimálního využití alokovaného místa je nutné spustit REORG operaci na všech tabulkách a indexech, kde došlo k výraznému výmazu dat.

#### *1.1.5.1 Struktura indexu*

DB2 podporuje pouze strukturu indexu typu B-strom. V nejvyšší vrstvě nazývané root node jsou obsaženy hodnoty a ukazatele. Každá tato hodnota se odkazuje pomocí ukazatele na prostřední vrstvu (intermediate node), kde je poslední hodnotou odkazovaného bloku. Podobně odkazuje prostřední vrstva na nejnižší vrstvu. Tady ovšem jednotlivé hodnoty odkazují na již dané číslo řádku v tabulce nebo na rozsah řádků. Počet prostředních vrstev je dán kardinalitou hodnot ve sloupci a velikostí tabulky. S rostoucí kardinalitou a velikostí tabulky také přibývá prostředních vrstev. Pro přehlednost uvádím obr. 6.




*Obr. 6. Grafická struktura B-stromu indexu (IBM Knowledge Centre, 2015)*

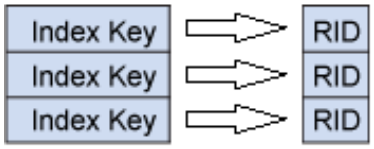
#### *1.1.5.2 Design advisor*

Proces design advisor je možné použít pro doporučení indexů na základě vstupního SQL dotazu. Po zpracování dotazu a vypočítání nových přístupových plánů navrhne advisor indexy společně s procentuálním dopadem na efektivitu dotazu. Na druhou stranu navrhuje i vymazání nepoužívaných indexů pro daný SQL dotaz.

Kardinalita sloupce v tabulce je jedním z hlavních parametrů, které je nezbytné sledovat při vytváření indexů. Pokud jsou ve sloupci pouze unikátní hodnoty (např. rodné číslo občana ČR), je kardinalita sloupce totožná s počtem řádků. Vyskytují-li se ve sloupci pouze dvě rozdílné hodnoty (např. pohlaví jedinců), je kardinalita rovná 2. Protože tabulky obsahují často několik až několik milionů řádků, je přehlednější uvádět kardinalitu jako poměr mezi četností unikátních hodnot ve sloupci ku počtu řádků tabulky (IBM, 2015b).

Hodnota kardinality indexu je zaznamenána v tabulce SYSCAT.INDEXES ve sloupci FULLKEYCARD.

Kardinalita přímo souvisí s leaf node indexu. V případě unikátních hodnot ve sloupci odkazuje každý ukazatel leaf node na konkrétní řádek v tabulce, jak je patrné ze schématu na obr. 7.




*Obr. 7. Kardinalita indexu je shodná s kardinalitou znaku (Stonawský, 2015)*

Jestliže je kardinalita nižší než počet řádků, je patrné, že jeden ukazatel v leaf node bude okazovat na větší počet řádků (viz schéma na obr. 8).


*Obr. 8. Kardinalita indexu je nižší než kardinalita znaku (Stonawský, 2015)*

V případě výrazně nízké kardinality indexu může docházet k přečtení většího počtu datových stránek, než tabulka vůbec obsahuje. Indexy s nízkou kardinalitou tedy nelze doporučovat v žádném případě. Nejenže neplní svou roli u SELECT dotazů, ale jako jakýkoliv jiný index alokují místo v úložišti a negativně ovlivňují UPDATE, INSERT a DELETE dotazy (IBM, 2015b).

### 1.1.6 Tabulkový prostor

Chen a kol. (2007) chápe tabulkový prostor jako strukturu úložiště obsahující tabulky, indexy, velké objekty a long data. Každá databáze musí mít prostor, kam bude ukládat systémová data, data uživatelská a dočasná. Seznam všech tabulkových prostorů je možné nalézt v systémové tabulce SYSCAT.TABLESPACES.

#### *1.1.6.1 Tabulkový prostor typu SMS*

Velikost tabulkového prostoru typu SMS je spravována pomocí správce souborů operačního systému. Místo na fyzickém úložišti je přidělováno na základě žádosti DBM k operačnímu systému a není předem alokováno při vytvoření tabulkového

prostoru. Typickým zástupcem SMS typu jsou dočasné tabulkové prostory, které v průběhu času prostor alokují, ale také jej uvolňují. Databázový výkon tohoto typu je nižší než typů DMS a autostorage.

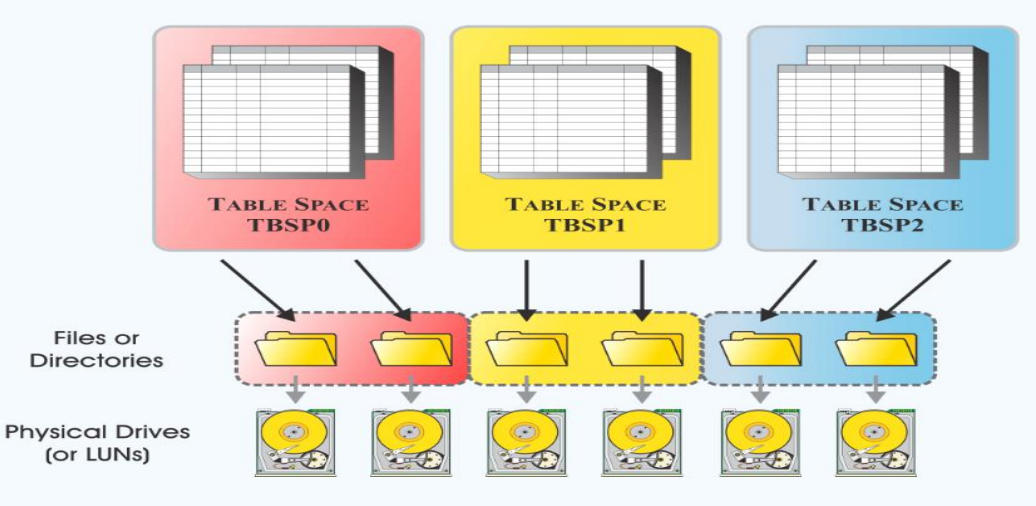
#### *1.1.6.2 Tabulkový prostor typu DMS*

Tento typ tabulkového prostoru spravuje DBM. Na rozdíl od SMS typu je fyzické úložiště předem alokováno na základě definice kontejnerů, které jsou vytvářeny společně s tabulkovým prostorem. S prostory typu DMS přichází možnost umístit kontejnery na různá fyzická úložiště za účelem využití paralelismu I/O operací. Typickým zástupcem jsou uživatelské tabulkové prostory. Databázový výkon typu DMS je srovnatelný s autostorage.

#### *1.1.6.3 Tabulkový prostor typu autostorage*

Typ autostorage je výchozím typem tabulkových prostorů. Pokud nejsou specifikovány přesné požadavky při vytváření autostorage prostoru, je jejich správa značně zjednodušena, protože o velikost kontejnerů, pojmenování a umístění se stará DBM. Typickým příkladem jsou uživatelské tabulkové prostory.

Tabulkové prostory jsou tvořeny jedním nebo více kontejnery. Kontejnerem může být složka, souborový systém nebo soubor. Pro zvýšení výkonu databáze můžeme kontejnery rozmístit mezi jednotlivé disky serveru, jak je znázorněno na obr. 9. Dosáhneme tak u disků snížení I/O operací.




*Obr. 9. Architektura tabulkových prostorů, kontejnerů a jejich umístění mezi disky serveru (Sanders, 2015)*

Výkonnost diskového pole lze hodnotit podle parametru overhead a přenosové kapacity (Sanders, 2015):

**Overhead** – udává dobu, kterou potřebuje jednotlivý kontejner předtím, než jsou data načtena do paměti.

$$OVERHEAD = \emptyset\ doba\ vyhledávání\ disku + \ 0{,}5 \times rotační\ prodleva \quad ms \tag{1}$$

$$rotační\ prodleva = \ \frac{1}{RPM} \ \times 60 \times 1000 \tag{2}$$

**Přenosová kapacita (transferrate)** – udává přibližnou dobu nutnou na přenesení jedné datové stránky do paměti.

$$TRANSFERRATE = \ \frac{1}{přenosová\ rychlost\ disku} \ \times \frac{1000}{102400} \times velikost\ stránky \tag{3}$$

Při přenosové rychlosti disku 150 MB/s a velikostí stránky 32 KB

Využití tabulkových prostorů má mnoho výhod. Jedním z nejdůležitějších je možnost jejich obnovy jakožto části databáze, což rovněž umožňuje přenést veškeré objekty tabulkového prostoru do jiné databáze. Zároveň lze ukládat indexy a tabulky do separovaných tabulkových prostorů.

#### *1.1.6.4 Katalogový tabulkový prostor*

Tento tabulkový prostor je vytvořen společně s databází. Nese název SYSCATSPACE a jsou zde uloženy systémové katalogové tabulky, jako jsou např.:

- SYSCAT.TABLES – tabulka obsahující seznam všech tabulek databáze,
- SYSCAT.TABLESPACES – tabulka obsahující seznam tabulkových prostorů,
- SYSCAT.DBAUTH – tabulka obsahující práva uživatelů a skupin na databázi (IBM, 2014a).

#### *1.1.6.5 Regulární tabulkové prostory*

Umožňují ukládat trvalá data, tabulky a indexy. Každý řádek tabulky je uložen do jedné datové stránky prostoru. Velikost datové stránky může nabývat hodnot 4KB, 8KB, 16KB, 32KB. Nelze tedy vložit řádek o velikosti 4 100 Bytů do prostoru, jenž užívá datové stránky o velikosti 4KB.

#### *1.1.6.6 Velké tabulkové prostory*

DB2 používá tyto tabulkové prostory k ukládání dat stejně jako u regulárních prostorů. Rozdílem je, že do velkých tabulkových prostorů lze ukládat LOB data. Navíc je možné uložit na jednu datovou stránku až 255 řádků tabulky, čímž bude dosaženo lepší efektivity ve využívání alokovaného místa. Při vytvoření databáze je vytvořen vždy výchozí velký tabulkový prostor USERSPACE1.

#### *1.1.6.7 Dočasné systémové tabulkové prostory*

DB2 využívá tyto prostory k ukládání dočasných dat vytvořeným na základě SQL dotazů, jako je operace SORT, reorganizace tabulek, vytváření indexů nebo operace JOIN mezi jednotlivými tabulkami. Výchozím dočasným prostorem při vytvoření databáze je TEMPSPACE1.

#### *1.1.6.8 Dočasné uživatelské tabulkové prostory*

Tyto tabulkové prostory nejsou vytvářeny společně s databází. Je nutné je vytvořit manuálně a jen v případě, že aplikace vyžadují jejich použití.

## 1.2 Správa operační paměti

Operační paměť je jedním z hlavních prvků zajišťující dostatečně rychlou odezvu databáze. V případě nesprávného nastavení parametrů využívající operační paměť může docházet ke snížení efektivity databáze, rychlosti zpracovávání dotazů aplikací nebo operací jako jsou REORG, RUNSTATS a monitorovacích mechanizmů.

Je nutné zvlášť rozlišit parametry na úrovni instance a databáze. Parametry instancí jsou sdíleny mezi veškeré databáze, které jsou zakatalogované pod danou instancí. Naproti tomu parametry databáze je potřeba nastavit pro každou databázi zvlášť. Ačkoliv je naprostá většina parametrů doporučena nastavit na automatickou hodnotu, některé parametry je třeba za určitých okolností nastavit manuálně k docílení požadovaného efektu.

### 1.2.1 Parametry instance

Administration Guide pro DB2 (columbia.edu, 2015) přehledně rozvádí charakteristiku parametrů instance:

**INSTANCE_MEMORY** je parametrem určujícím maximální využití paměti celé instance. Je tedy horním limitem paměti pro všechny její databáze. Výchozí doporučovaná hodnota je nastavena na automatic, a instance alokuje až 95 % volné operační paměti. V případě, že by bylo potřeba její využití paměti limitovat, lze nastavit parametr fixně, maximálně však na 256TB pro 64-bitové systémy.

**AUDIT_BUF_SZ** je parametr přímo spojen se službou auditu instance. Pokud je tento parametr vypnutý, jsou veškeré SQL dotazy zaznamenávány do logu auditu synchronně s jejich průběhem na databázi. V případě, že tento parametr spustíme, budou záznamy vkládány asynchronním způsobem. Služba audit tedy nečeká na dokončení SQL dotazu, což zvyšuje výkon celé instance.

**MON_HEAP_SZ** parametr alokuje místo v paměti pro monitorovací služby databáze. Mezi tyto služby můžeme zařadit zapínání různých monitorovacích módů, snapshot databáze a instance, resetování monitoru nebo aktivace monitoru událostí. Stejně jako ostatní výše zmíněné parametry, alokuje paměť při spuštění instance. Nastavíme-li tento parametr na příliš nízkou hodnotu, může dojít ke ztrátě některých monitorovací funkcí, jako jsou příkazy DB2TOP, SNAPSHOT nebo DB2PD.

### 1.2.2 Parametry databáze

Příkazem ACTIVATE DATABASE aktivujeme databázi a všechny její potřebné služby. Jakmile je databáze aktivovaná, je k dispozici pro připojení a použití jakékoliv aplikace. Gandhi (2013) blíže vysvětluje činnost databáze: v případě neaktivované databáze musí první aplikace, která se k databázi připojí, čekat, než se tyto služby aktivují. Tímto příkazem rovněž aktivujeme všechny parametry databáze. Dochází tedy k alokaci potřebného místa v operační paměti.

**BUFFERPOOL** je základním stavebním kamenem DB2 databází. Při vytváření databáze vytvoří DBM výchozí bufferpool IBMDEFAULTBP. Další bufferpooly můžeme vytvořit sami pomocí příkazu CREATE BUFFEPOOL. Každý tabulkový prostor je svázán s jedním bufferpoolem o stejné velikosti datové stránky. Jeden

bufferpool ale může využívat vícero tabulkových prostorů (IBM Knowledge Center, 2015). Jeho funkce tkví ve snížení I/O operací, což je ve většině případů nejméně efektivním způsobem čtení a úpravy dat. Jakákoliv operace s daty je nejdříve zapsána do bufferpoolu a teprve potom až na disk. Toto je výhodné zvláště pro opakující se čtení stejných dat. Jakmile jsou data v paměti, jsou rychleji přečtena dalšími aplikacemi, než jak by tomu bylo v případě čtení z disku. Velikost bufferpoolu je možno opět nastavit na fixní nebo automatickou hodnotu.

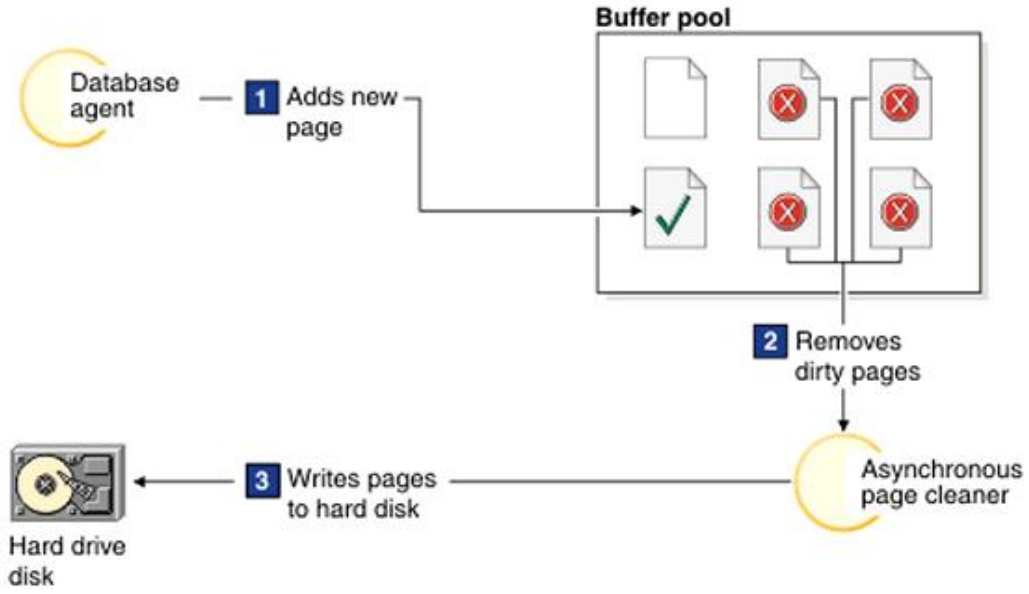
```
db2 create bufferpool BUF32K immediate size 1000
automatic pagesize 32k
```

*Obr. 10. Příklad vytvoření bufferpoolu o velikosti datové stránky 32KB (dle IBM Knowledge Center, 2015 upravil Stonawský, 2015)*

| Bufferpool Name | # of DBP | BP Pages |
|---|---|---|
| BUF32K | 1 | 57568 |
| BUF8K | 1 | 63692 |
| IBMDEFAULTBP | 1 | 1569 |
| IBMSYSTEMBP16K | 1 | 16 |
| IBMSYSTEMBP32K | 1 | 16 |
| IBMSYSTEMBP4K | 1 | 16 |
| IBMSYSTEMBP8K | 1 | 16 |

*Obr. 11. Zobrazení bufferpoolu v DB2TOP (Stonawský, 2015)*

Bufferpooly jsou spravovány databázovým agentem, který do něj vkládá jednotlivé datové stránky a také se stará o jejich výmaz a zápis na disk. Tato operace je synchronní, proto musí aplikace vždy počkat na databázového agenta, než vyčistí stránky z bufferpoolu a uvolní tak místo pro další stránky. Pomocí parametru DB2_USE_ALTERNATE_PAGE_CLEANING však můžeme zapojit do správy bufferpoolu tzv. page-cleaner agenta, který převezme roli databázového agenta a začne proaktivně a asynchronně odstraňovat stránky z bufferpoolu. Dosáhneme tak snížení doby čekání aplikací na vyčištění bufferpoolu. Stránky, které agenti odstraňují z bufferpoolu, označujeme jako znečištěné (dirty). Proces jejich vyčištění spočívá v zapsání na disk (IBM Knowledge Center, 2015).




*Obr. 12. Zapojení asynchronního agenta do procesu čištění stránek z bufferpoolu (IBM Knowledge Center, 2015)*

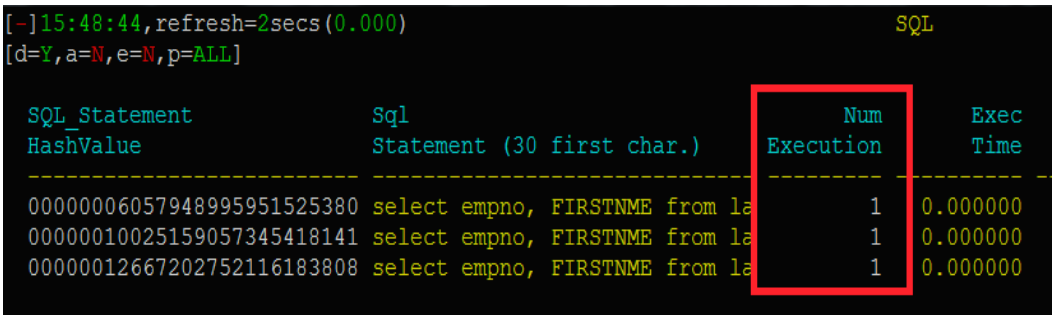
U bufferpoolu sledujeme 3 nejdůležitější ukazatele:

- Logické čtení – aplikace načítají datové stránky z bufferpoolu.
- Fyzické čtení – stránky v bufferpoolu chybí a jsou čteny z disků.
- Hit ratio – poměr mezi logickým a fyzickým čtením.

**DATABASE_MEMORY** parametrem specifikujeme celkové nároky databáze na velikost alokovaného místa v operační paměti serveru. V IBM Knowledge Center (2015) se lze dočíst, že tento parametr zahrnuje velikost bufferpoolu, parametr UTIL_HEAP_SZ, UTIL_HEAP_SZ, LOCKLIST a package cache. Je tedy zřejmé, že je ovlivněn dalšími parametry databáze a jeho nastavení na fixní hodnotu je užitečné pouze v případě, kdy jsou fixní i ostatní parametry. Výchozí hodnota je nastavena na automatic.

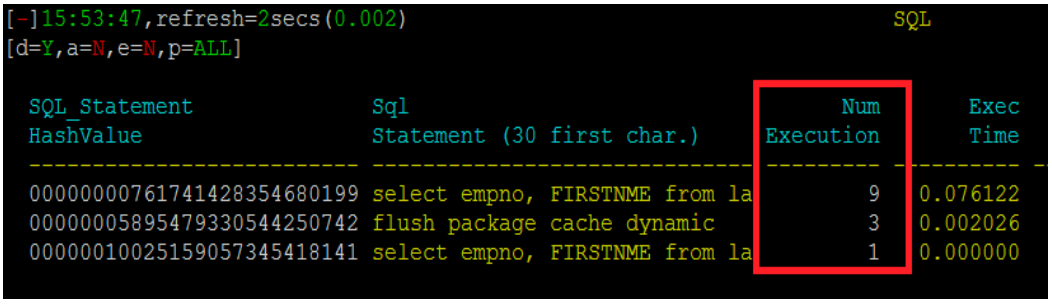
**STMT_CONC** (Statement concentrator) přímo ovlivňuje velikost package cache. Každý SQL dotaz, který spustíme, je zaznamenán právě do package cache. A každý záznam zabírá místo v operační paměti. V některých případech může tato část alokované paměti nabývat několika desítek GB. Pokud bude tento parametr nastaven na LITERALS, každá proměnná dotazu bude nahrazena zástupným znakem

a záznamy pro stejné dotazy s různými proměnnými budou do package cache uloženy jen jednou (IBM Knowledge Center, 2015).




*Obr. 13. Výpis package cache při vypnutém STMT_CONC parametru (dle IBM Knowledge Center, 2015 upravil Stonawský, 2015)*

Je však nezbytné uvážit skutečnost, že pro každý záznam je vypočítán jeden přístupový plán. Spuštěním tohoto parametru vlastně vnucujeme podobným SQL dotazům stejný přístupový plán, což může mít za následek snížení výkonu databáze při vyhodnocení SQL dotazu.




*Obr. 14. Výpis package cache při zapnutém parametru STMT_CONC (dle IBM Knowledge Center, 2015 upravil Stonawský, 2015)*

Package cache je vyčištěna po deaktivaci databáze nebo online pomocí příkazu FLUSH PACKAGE CACHE DYNAMIC. Arnold (2012) však upozorňuje, že potom ale musí být znovu propočteny všechny přístupové plány, což opět vede ke snížení výkonnosti a mělo by být prováděno v produkčním prostředí pouze v době údržby.

**UTIL_HEAP_SZ** je ukazatelem maximálního využití paměti pro zálohování, obnovu a LOAD operace databáze. Tento parametr není využit, pokud žádná

z těchto operací zrovna neběží. Proto je ho vhodné ponechat na výchozí hodnotě 5 000 datových stránek.

**LOCKLIST** alokuje v paměti přidělené severu stránky o velikosti 4KB. Pokud se DBM rozhodne pro zámky na jednotlivých řádcích tabulky, každý tento řádek je zaznamenán v locklistu. V případě naplnění locklistu, dochází k eskalaci zámků. Eskalací DBM snižuje velikost locklistu a zároveň nahradí zámky na řádcích jen jedním zámkem na celé tabulce.

## 1.3 Údržba databáze

Údržba databáze je proaktivním procesem, který nejčastěji provádí manuálně administrátor databáze, nebo automaticky DBM v případě nastavení příslušných parametrů. DB2 obsahuje 2 hlavní funkce pro udržování databáze v optimálním stavu. Jednou z nich je funkce RUNSTATS, která aktualizuje statistiky o objektech, druhou funkcí je REORG, která funguje podobným způsobem jako defragmentace disku. Obě tyto funkce je nutné spouštět v pravidelných intervalech.

### 1.3.1 RUNSTATS

DB2 optimalizátor používá katalogové tabulky databáze k získání informací o databázi, objemu dat a dalších vlastností. Tyto informace následně využívá k volbě nejlepšího způsobu přístupu k datům. Chen a kol. (2009) varuje, že pokud máme neaktuální statistiky, může se stát, že optimalizátor na základě nepřesné výchozí statistiky zvolí neefektivní přístupový plán. Pro získání aktuálních statistik o tabulkách a indexech spouštíme RUNSTATS, zejména pokud od posledního spuštění příkazu došlo ke značnému množství aktualizací nebo k vytvoření nových indexů. To optimalizátoru zajišťuje nejpřesnější informace, pomocí kterých pak lze určit nejvhodnější přístupový plán (Chen a kol., 2009).

```
Retrieval Time: 03/11/2015 10:36:34
TbspaceID: 3        TableID: 969
Schema: TEST   TableName: MAXEMP
Status: Completed     Access: Allow write
Start Time: 03/10/2015 21:31:21   End Time: 03/10/2015 21:31:52
Total Duration: 00:00:31
Prev Index Duration [1]: 00:00:20
Prev Index Duration [2]:  -
Prev Index Duration [3]:  -
Cur Index Start: 03/10/2015 21:31:22
```

*Obr. 15. Monitorování RUNSTATS na tabulce TEST.MAXEMP (Stonawský, 2015)*

RUNSTATS spouštíme i po aktualizaci tabulek. V opačném případě může optimalizátor považovat tabulku za prázdnou a kardinalitu v katalogových tabulkách rovnu nule. Neaktuální kardinalita má za následek nesprávné rozhodnutí optimalizátoru, což negativně ovlivňuje celkovou výkonnost databáze.

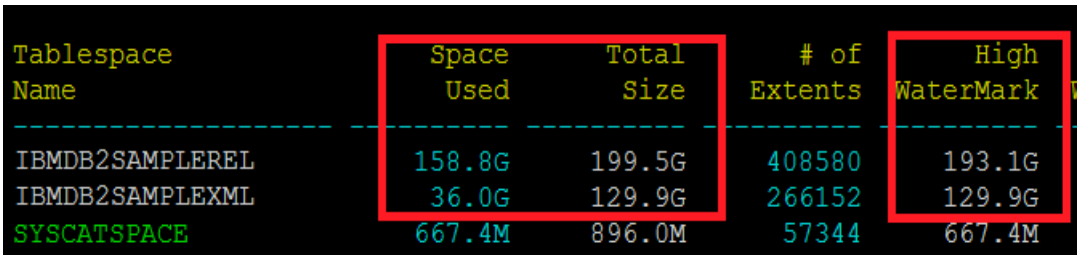
- RUNSTATS je vhodné spouštět u všech tabulek a indexů, které se mohou aplikace využívat v SQL dotazech.
- WITH DISTRIBUTION RUNSTATS zahrnuje statistiku kvantilů a statistiku často opakujících se hodnot. Počet těchto hodnot je určen proměnnou NUM_FREQVALUES v konfiguraci databáze.
- Kromě statistiky mohou způsob výběru přístupového plánu ovlivňovat také další faktory (například pořadí příslušných řádků, velikost tabulky a velikost vyrovnávacích pamětí).
- Po spuštění příkazu RUNSTATS nebo změně parametrů konfigurace je potřeba spustit příkaz BIND pro opětovné svázání aplikací umožňující využívat aktualizované přístupové plány (Chen a kol., 2009).

### 1.3.2 REORG

Funkce REORG je jedním z klíčů pro udržení výkonnosti DB2 databáze. Jeden ze základních pravidelných úkolů, které administrátor musí provádět, je spouštění REORG na tabulkách a indexech, kde proces REORGCHK indikuje vysokou fragmetaci. Chen a kol. (2009) doporučuje REORG spouštět v pravidelných intervalech, nejlépe každý týden. Současný souběh REORG a vyššího vytížení databáze aplikacemi může ale způsobit problémy ve výkonu databáze. Proto Chen a kol. (2009) nabádá ke spouštění REORG v čase, kdy je využití aplikací databáze nej-

menší, nebo v případě, kdy dočasný pokles výkonosti nezpůsobí možnou nedostupnost dat. REORG je tedy možné spouštět v offline i online režimu. V případě offline režimu je databáze přístupná pro všechny aplikace, ale tabulky, na kterých REORG probíhá, jsou zpřístupněny pouze pro čtení, nikoliv však pro zápis.

Postupem času jsou do tabulek vkládány jednotlivé řádky, ale jsou z nich také odstraňovány. Samotné odstranění řádku neznamená automaticky uvolnění místa. Velikost každé tabulky není určena objemem jejich dat, ale hodnotou high wattermark. V případě vkládání dat tato hodnota roste, v případě jejich odstranění však nijak neklesá. Jediný způsob, jak tuto hodnotu snížit a tím i snížit tabulkou alokovaný prostor, umožňuje REORG (Chen a kol., 2009).



| Tablespace Name | Space Used | Total Size | # of Extents | High WaterMark |
|---|---|---|---|---|
| IBMDB2SAMPLEREL | 158.8G | 199.5G | 408580 | 193.1G |
| IBMDB2SAMPLEXML | 36.0G | 129.9G | 266152 | 129.9G |
| SYSCATSPACE | 667.4M | 896.0M | 57344 | 667.4M |

*Obr. 16. Sledování high wattermark pomocí DB2TOP (Stonawský, 2015)*

## 1.4 SQL dotazy aplikací

Aplikace komunikují s databázemi pomocí jazyka SQL, který byl vyvinut již v 70. letech 20. století pod názvem SEQUEL. SQL příkazy rozdělujeme na 4 základní slupiny (IBM, 2014b):

- DML (definice dat) – select, insert, update, delete, call
- DDL (manipulace s daty) – create, alter, drop, truncate, comment, rename
- DCL (řízení přístupových práv) – grant, revoke
- TCL (řízení transakcí) – commit, rollback, savepoint

Kompilační proces SQL dotazu musí projít několika fázemi, než dojde k vytvoření přístupového plánu a spuštění dotazu. Jedná se o interní proces databáze, který probíhá v alokované paměti.

- *Parsováním dotazu* ověřuje kompilátor samotnou syntaxi dotazu. Pokud najde chybu, odešle chybovou hlášku aplikaci nebo uživateli a zastaví celý

proces. Pokud je dotaz v pořádku, zaznamená dotaz do interního grafu modelu.

- *Ověřováním sémantiky* kompilátor kontroluje konzistenci SQL. Např. zda datové typy odpovídají dotazu.
- *Přepsáním dotazu* kompilátor převádí SQL pro snadnější optimalizaci. Taktéž ukládá graf modelu dotazu, který můžeme vidět pomocí db2expln příkazu. Kompilátor může také manipulovat s predikáty dotazu pro zvýšení optimalizace a snížení jeho timeron jednotek.
- *Optimalizace přístupového plánu* je část kompilace, o kterou se stará DB2 optimalizátor. Vypočítává nejefektivnější přístupový plán s přihlédnutím na faktory, které jej ovlivňují.
- *Vytvoření spustitelného kódu* je posledním krokem. Kompilátor využívá přístupového plánu a grafu modelu dotazu k vytvoření spustitelné sekce pro SQL dotaz (IBM, 2014b).

### 1.4.1 DB2 optimalizátor

DB2 optimalizátor je proces, který rozhoduje o přístupovém plánu k datům uloženým v databázi na základě SQL dotazů. Snaží se určit nejefektivnější přístupovou cestu. Propočítává tedy nároky na cykly procesoru, I/O operace disků a operační paměť. Každá přístupová cesta je potom ohodnocena v timeron jednotkách a vždy je zvolena cesta, která je ohodnocena co nejmenším počtem těchto jednotek.

IBM (2014) timeron charakterizuje jako jednotku měření DB2 optimalizátoru, kterou ale nemůžeme na základě jakýchkoliv okolností převádět na čas běhu SQL dotazu, i když v době zavedení jednotky se jeden timeron rovnal přibližně jedné milisekundě. V současnosti nám pouze udává poměr náročnosti jednotlivých SQL dotazů.

### 1.4.2 Optimalizační úrovně

Optimalizační úroveň je volitelná pro každou databázi. Pomocí databázového parametru DFT_QUERYOPT určujeme, zda DBM věnuje více času vypočítávání přístupového plánu nebo samotnému běhu SQL dotazu. Podle typu databáze a dotazů volíme i optimalizační úroveň. V případě, že se jedná o databázi čistě OLTP typu s jednoduchými dotazy, není třeba volit vysokou optimalizační úro-

veň, protože samotný proces optimalizace nepřináší dostatečný benefit. Naproti tomu databázi typu WAREHOUSE s komplexními a málo opakovanými dotazy prospívá mnohem lépe vysoká optimalizace SQL dotazů.

> Úroveň 0 - minimální optimalizace vhodná pro SQL dotazy na tabulkách s výbornými indexy.
>
> Úroveň 5 - výchozí hodnota a doporučená pro většinu databází.
>
> Úroveň 9 - maximální optimalizace pro velmi náročné SQL dotazy na velkých tabulkách (Chen a kol., 2014).

### 1.4.3 Zámky a ochrana před souběžností

Pro ochranu před souběžností a nekontrolovaného přístupu k datům zamyká DBM řádky v tabulkách nebo celé tabulky, buffer pooly a datové oddíly. Zdali dojde k zamčení celé tabulky nebo jen několika řádků záleží na izolační úrovni, přístupovém plánu a LOCKLIST parametru.

DB2 rozlišuje několik typů zámků:

- *S zámek* – další aplikace mohou zamknuté objekty pouze číst.
- *U zámek* – používá se v případě, že aplikace, která drží zámek na objektu ještě nezačala nijak měnit data a ostatní aplikace mají možnost tato data číst.
- *X zámek* – vlastník zámku může data číst i upravovat. Ostatním aplikacím je dovoleno data pouze číst a to jen v UR režimu.
- *IN zámek* – vlastník zámku může číst data, nikoliv však upravovat. Ostatní aplikace mohou data i upravovat.
- *IX zámek* – vlastník zámku může číst i upravovat data. Ostatní aplikace mohou číst i upravovat tabulku.
- *Z zámek* – používá se jen v případě DROP operace nebo určitého typu REORG. Žádná další aplikace nemá přístup k datům tabulky (IBM Knowledge Center, 2015).

Zámky mají taktéž určité atributy:

- *Typ* – určuje práva přístupu k uzamčeným datům aplikaci, která zámek drží i ostatním aplikacím, které čekají, než bude zámek uvolněn.
- *Objekt* – tedy zdroj, který je zamčen. Typicky jsou to tabulky, řádky tabulek nebo bufferpooly.
- *Doba trvání zámku* – ovlivňuje ji izolační úroveň aplikace (IBM Knowledge Center, 2015).

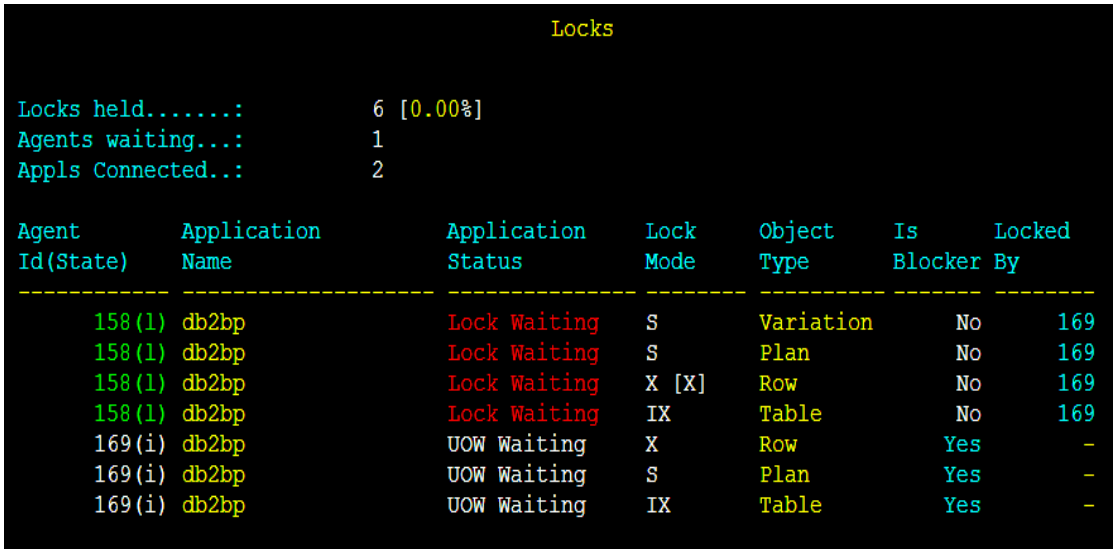
DB2 používá různé izolační úrovně pro zajištění přístupu k datům, s kterými aplikace aktuálně pracují. Snaží se tak zabránit souběhu aplikací, který by mohl vést ke ztrátě dat nebo phantom čtení. Izolační úrovně přímo ovlivňují velikost LOCKLISTU.

Typy izolačních úrovní:

- *UR (uncommitted read)* – nejnižší forma izolačních úrovní. Používá se jen v případě SELECT SQL dotazů. Řádky jsou zamknuty pouze v případě, kdy je aplikace upravuje, nebo v případě úmyslu smazání a změny celé tabulky další aplikací.
- *CS (cursor stability)* – aplikace drží zámek na konkrétním řádku odkazujícího na její ukazatel. Všechny ostatní řádky, které odpovídají SQL dotazu aplikace, mohou další aplikace libovolně upravovat a mazat. CS lze označit za výchozí izolační úroveň.
- *RS (read stability)* – aplikace drží zámek jen na řádcích odpovídajících danému SQL dotazu. Ostatní aplikace s těmito řádky pracovat nemohou, mohou však vkládat další řádky do tabulky.
- *RR (repetable read)* - jedná se o nejvyšší stupeň izolační úrovně. Práci aplikace s RR stupněm nemohou nijak ovlivnit jiné aplikace. Není možno vkládat, mazat nebo měnit řádky již načtené aplikací s izolační úrovní RR, bez ohledu na to, zdali odpovídají SQL dotazu. Tato úroveň se používá pro spuštění rozsáhlých SQL dotazů a v případě, že není žádoucí vracení různých hodnot dotazu, spustíme-li jej vícekrát (IBM Knowledge Center, 2015).

Právě doba trvání zámku (viz níže obr. 17) může být kritickým místem pro ostatní aplikace čekajících na uvolnění zámku. Z uživatelského hlediska to vypadá, jako

kdyby aplikace nebo databáze zamrzla. LOCKTIMEOUT parametrem však můžeme nastavit dobu čekání aplikace na uvolnění zámku. Po uplynutí této doby dojde k zobrazení chybové hlášky v aplikaci nebo v konzoli uživatele. Nelze však nijak limitovat dobu držení zámku. V krajní situaci jsou administrátoři nuceni aplikace držící zámky na tabulkách příliš dlouho odpojit.




*Obr. 17. Monitorování zámků pomocí DB2TOP (Stonawský, 2015)*

### 1.4.4 Stupně predikátů SQL dotazů

Pro efektivní formulování SQL dotazů jsou stupně predikátů velice důležité. Podle těchto stupňů rozhoduje DB2 optimalizátor, zda dojde k použití indexu tabulky či nikoliv. Predikáty jsou definovány ve WHERE klauzuli SQL dotazu

#### *1.4.4.1 První stupeň – skenování datových stránek indexu*

- Jednoduché operátory ( =, >, >=, <, <=, <>, !=, !>, !< )
- IN ( výčet hodnot )
- BETWEEN ( pouze mezi 2 hodnotami tabulky )
- NULL ( testování null hodnoty )
- LIKE

Tyto predikáty mohou být rovněž spojeny pomocí boolean operátorů AND, OR, NOT (Neagu a Pelletier, 2012).

#### *1.4.4.2 Druhý stupeň – skenování datových stránek tabulky*

- CASE
- Aritmetické a skalární funkce ( DATE, YEAR, TIMESTAMP )
- EXIST, NOT EXIST
- BETWEEN (srovnávání hodnot v tabulce – WHERE A1 BETWEEN C1 and D1)

Pro predikáty druhého stupně nemůže DB2 optimalizátor efektivně využít stránky indexů a dochází tak ke skenování datových stránek tabulky. Použití predikátů druhého stupně v dotazech, které aplikace často a opakovaně užívají ke čtení nebo úpravě dat, výrazně zvyšuje nároky na zdroje databáze. Ukazuje se proto jako vhodné, je používat jen v omezené míře nebo v případě komplexních a neopakujících se SQL dotazů (Neagu a Pelletier, 2012).

# 2 DB2 BLU AKCELERACE

BLU akcelerace je poslední velkou inovací DB2 relačních databází. Byla představena ve verzi v10.5. Zaměřuje se hlavně na výkon SQL v analytickém prostředí, kde jsou SELECT dotazy na BLU tabulkách zpracovávány více než 50krát rychleji bez nutnosti tyto dotazy jakkoliv měnit. Tuto novou funkci nelze zatím efektivně využít u OLTP databází s vysokým počtem INSERT, UPDATE a DELETE operací.

## 2.1 Sloupce vs. řádky

Zatímco tradiční způsob organizování tabulek je určen řádky, tabulky vytvořené v BLU jsou organizované pomocí sloupců. Orientací rozumíme rozdělení dat tabulky do jednotlivých datových stránek tabulkového prostoru. Přehledněji znázorněno níže na schématu v obr. 18.


*Obr. 18. Rozdělení sloupů tabulek mezi jednotlivé datové stránky tabulkového prostoru (dle IBM, 2015b upravil Stonawský, 2015)*

Jednoznačné interní identifikátory řádků (RID) byly nahrazeny identifikátorem TSN. Nelze proto pro tento druh tabulek vytvářet indexy, které RID identifikátor používají. TSN zde odkazuje na přímo na řádek v tabulce, nikoliv na datovou stránku s příslušným řádkem.

V instanci zpřístupňujeme BLU akceleraci nastavením parametru DB2_WORKLOAD na hodnotu ANALYTICS. Struktura databáze a použití některých služeb je ve velké míře zjednodušeno:

- Absence design advisoru a indexů – RID neexistuje.
- Absence REORG a RUNSTATS (automatizováno při LOAD operaci).
- Žádné MQT a partitioned tabulky (IBM, 2015b).

Tabulky vytváříme stejným způsobem, jako regulární tabulky, ale s klauzulí ORGANIZE BY COLUMN (viz obr. 19).

```
CREATE TABLE DB2TEST.EMPN
(
ID SMALLINT NOT NULL,
NAME VARCHAR(9),
. . . .
COMM DECIMAL(7,2)
)
ORGANIZE BY COLUMN
IN USERSPACE1;
```

*Obr. 19. Příklad vytvoření tabulky s BLU akcelerací (Stonawský, 2015)*

## 2.2 Synopsis tabulky

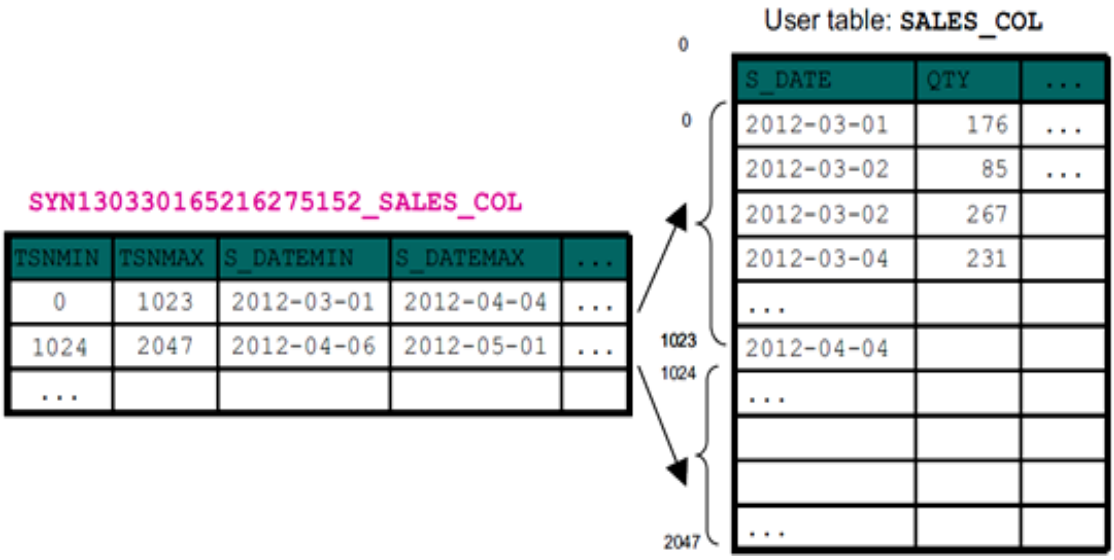
Metadata a jejich správu pro tabulky s BLU akcelerací obstarávají sysnopsis tabulky (viz obr. 20). Tyto tabulky jsou rovněž sloupcově orientované a lze je najít pod schématem SYSIBM. Pro každou tabulku s BLU akcelerací existuje jedna synopsis tabulka.

```
TABLE_NAME                      SCHEMA_NAME TABLEORG
------------------------------- ----------- --------
ZAMCOL                          DB2TEST     C
SYN150512213525873585_ZAMCOL    SYSIBM      C
```

*Obr. 20. BLU a synopsis tabulka (Stonawský, 2015)*

Synopsis tabulky obstarávají podobnou funkci jako indexy. Na rozdíl od indexů ale popisují pomocí TSN rozpětí řádků, než jak je jeden řádek běžný u indexů. V případě, že použijeme s klauzulí WHERE S_DATE = 2012-05-01, DB2 pře-

skočí první segment řádků a začne vyhledávat data ve druhém segmentu, jak je znázorněno na obr. 21 (IBM, 2015b).




*Obr. 21. Relace mezi BLU tabulkou a synopsis tabulkou (IBM, 2015b)*

## 2.3 BLU komprese

Sloupcový způsob orientace tabulky má výhodu v mnohem vyšší kompresi dat. Komprese orientovaná na sloupce tabulky je daleko efektivnější než na jejich řádcích. Sloupce tabulky mají totiž stejný datový typ a je proto mnohem pravděpodobnější, že najdeme stejnou hodnotu ve sloupci, než celý stejný řádek. Navíc dochází k mnohem efektivnějšímu využít velikosti datové stránky tabulkového prostoru, protože se jednotlivý počet uložených hodnot jednoho sloupce do datové stránky nemusí shodovat s počtem hodnot jiných sloupců.

Každá tabulka s BLU akcelerací je komprimována. V příkazu pro vytvoření tabulky totiž zcela chybí COMPRESS klauzule. Vyšší komprese s sebou přináší nejen úspory v podobě alokovaného místa na fyzickém úložišti, ale také počet I/O operací, náklady na chlazení, energetickou náročnost a správu úložiště. Rovněž zkracuje dobu záloh, obnovy databáze (IBM, 2015b).

## 2.4 Podpora ostatních funkcí DB2

Ačkoliv můžeme BLU akceleraci považovat za výrazný evoluční krok k efektivnímu zpracování SELECT dotazů a úspoře alokovaného prostoru úložišť, je třeba si uvědomit, že není vhodná pro veškeré typy databází a SQL dotazy. Je-

likož se stále jedná o čerstvou novinku, BLU nepodporuje řadu kritických funkcí, které můžeme v DB2 využívat:

- HADR – řešení vysoké dostupnosti pro dosažení SLA.
- Prostředí pureScale – škálovatelné řešení zdrojů.
- DPF – databázové oddíly.
- Federace dat.
- Replikace tabulek.
- Transport BLU tabulek do jiné databáze (IBM, 2015b).

BLU akcelerace je rovněž zahrnuta pouze v licenci AESE, která s sebou přináší vyšší nároky na finanční zdroje.

# II. PRAKTICKÁ ČÁST

# 3 LADĚNÍ VÝKONU DB2 DATABÁZÍ V PRAXI

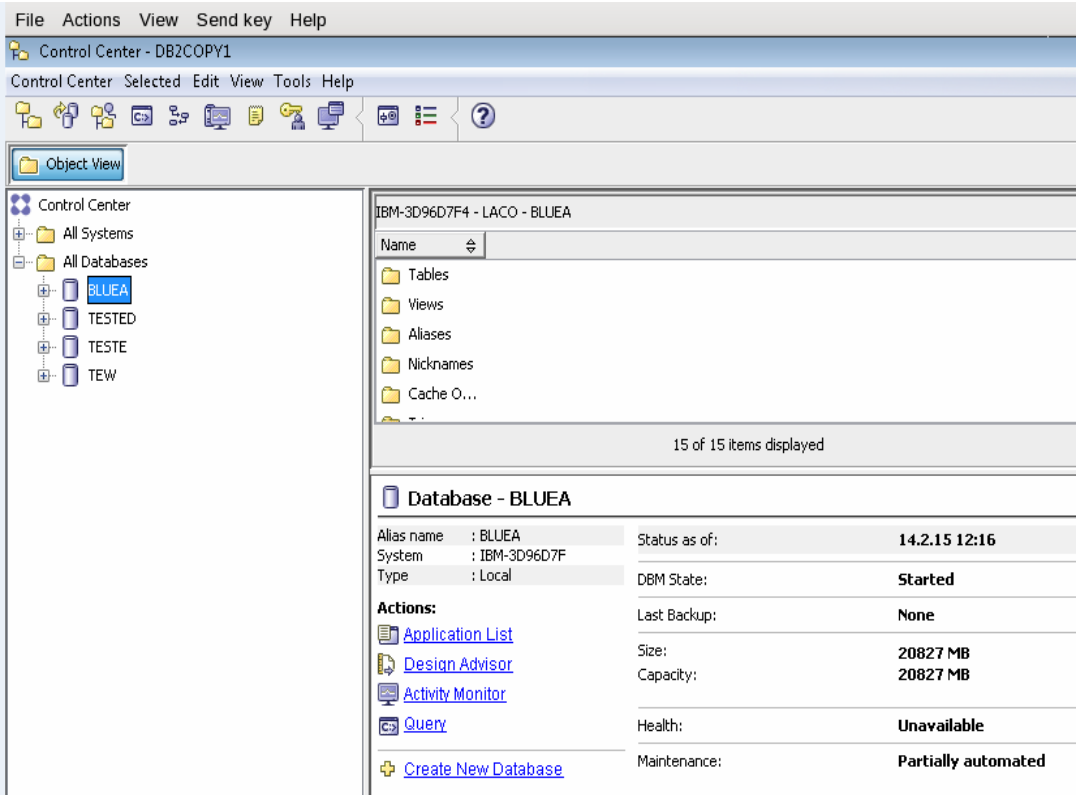
Pro konkurenceschopnost zákazníků využívající databázové systémy k podnikání jsou funkční a vyladěné databáze naprostou nezbytností. V opačném případě zákazník vlivem pomalých a neefektivních systémů může přicházet o dobré jméno, potenciální klienty a zisk z podnikání. Ladění databází je jedním z postupů jak databáze optimalizovat pro aplikační prostředí.

## 3.1 Metodika

Testování vlivu jednotlivých parametrů na výkon databází probíhalo na testovacím serveru firmy IBM. Testovací server v průběhu všech operací nepoužívala žádná jiná aplikace nebo databáze, která by ovlivňovala vytížení zdrojů serveru. Statistická data získaná pomocí klientského programu jsem vyhodnocoval programem Excel MS Office. Testovací SQL dotazy byly převzaty z praxe a přeformulovány.

Konfigurace testovacího serveru:

- Operační systém AIX 7.1 TL03 64bit.
- 4 CPU PowerPC_POWER7.
- 8GB RAM.
- Software DB2 LUW v10.5 FP5 licence AESE.
- Klientský program Control Center (viz obr. 22).
- TSM server v7.1.




*Obr. 22. Program Control Center s katalogovanými databázemi (Stonawský, 2015)*

## 3.2 Indexovatelné SQL dotazy

Chybějící indexy jsou ve většině případů důvodem dlouhé doby běhu SQL dotazů. Testovaný SQL dotaz (viz obr. 23) jsem spustil společně s Unixovým příkazem TIME, abych mohl změřit jeho dobu zpracování databází. Přestože kritéria dotazu splňovalo pouze 1 473 záznamů, spotřeboval na svůj běh 236 806 timeron jednotek.

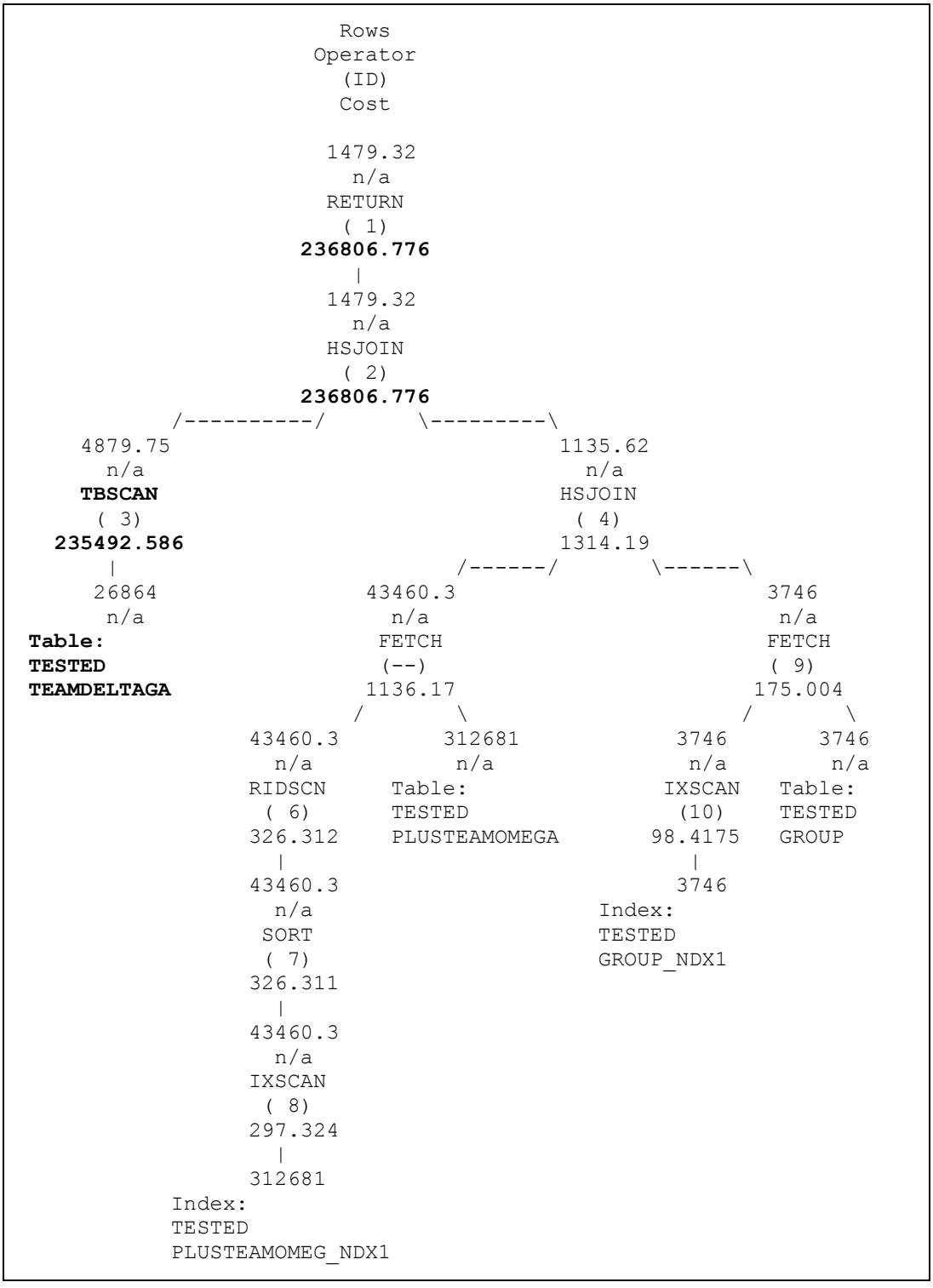
```
SELECT DISTINCT *
  FROM TESTED.TEAMDELTAGA A JOIN TESTED.GROUP B ON
  A.GROUP =B.GROUP JOIN TESTED.PLUSTEAMOMEGA C
  ON B.GROUPID =C.OWNERID
  AND C.OWNERTABLE ='GROUP'
WHERE C.CUSTOMER IN 'MINOT' AND A.ROWSTAMP > 1715536735;

  1473 record(s) selected.

Estimated Cost = 236806.776
```

*Obr. 23. Testovaný SQL dotaz a doba běhu (Stonawský, 2015)*

Funkce DB2explain, která zobrazuje grafický plán SQL dotazu (znázorněno na obr. 24), pomáhá k rychlejší identifikaci problémového místa. Při čtení grafu sledujeme hodnotu COST, která reprezentuje kumulativní množství timeron jednotek všech předchozích operací.




*Obr. 24. Plán SQL dotazu (Stonawský, 2015)*

Nejnáročnější operací SQL dotazu bylo skenování datových stránek tabulky TESTED.TEAMDELTAGA, které spotřebovalo 235 492 timeronů. WHERE klauzule dotazu obsahuje predikáty prvního stupně. Pokud tedy zvolíme vhodné indexy, nebude je optimalizační proces ignorovat. Vytváření indexů do jisté míry zjednodušuje funkce Design advisor , který propočítává kardinalitu indexovaných sloupců ve WHERE klauzuli dotazu, efektivitu nového řešení a potřebné místo na disku. Názvy navržených indexů jsou pouze orientační a jsou definovány pomocí vnitřní sekvence DB2. Obr. 25 ilustruje doporučené indexy funkci Design advisor.

```
Trying variations of the solution set.
  2  indexes in current solution
 [236806.776] timerons  (without recommendations)
 [1455.7] timerons  (with current solution)
 [99.99%] improvement


-- LIST OF RECOMMENDED INDEXES
-- ===========================
-- index[1],    95.30MB
   CREATE INDEX "TESTED"."IDX1505121042260"
   ON "TESTED"."TEAMDELTAGA" ("ROWSTAMP")
   ALLOW REVERSE SCANS;
   COMMIT WORK ;
-- index[2],    71.90MB
   CREATE INDEX "TESTED"."IDX1505121051216"
   ON "TESTED"."PLUSTEAMOMEGA" ("CUSTOMER")
   ALLOW REVERSE SCANS;
   COMMIT WORK ;
```

*Obr. 25. Indexy doporučené funkcí Design advisor (Stonawský, 2015)*

Před samotným vytvořením indexů je nutné manuálně zkontrolovat kardinalitu navržených indexů. Design advisor je totiž automatizovaný proces doporučující i indexy s nízkou kardinalitou a jakoukoliv možnost zlepšení výkonnosti. Kardinalitu jednotlivých sloupců tabulek lze nalézt v systémové tabulce SYSCAT.COLUMNS. Leaf node případného index na sloupci ROWSTAMP tabulky TESTED.TEAMDELTAGA bude odkazovat na konkrétní RID, což je ideální stav, jak je i dokresleno na obr. 26.

| TABSCHEMA | TABNAME | COLNAME | COLCARD | TABCARD | CARD_PERC |
|---|---|---|---|---|---|
| TESTED | TEAMDELTAGA | TEAMDELTAG | 26864. | 26864. | 1.00 |
| **TESTED** | **TEAMDELTAGA** | **ROWSTAMP** | **26864.** | **26864.** | **1.00** |
| TESTED | TEAMDELTAGA | RESPPARTYG | 13952. | 26864. | 0.51 |
| TESTED | TEAMDELTAGA | RESPPARTY | 13952. | 26864. | 0.51 |
| TESTED | TEAMDELTAGA | RESPPARTYG | 3265. | 26864. | 0.12 |
| TESTED | TEAMDELTAGA | RESPPARTYS | 3264. | 26864. | 0.12 |
| TESTED | TEAMDELTAGA | GROUP | 3012. | 26864. | 0.11 |
| TESTED | TEAMDELTAGA | USEFORORG | 8. | 26864. | 0.00 |
| TESTED | TEAMDELTAGA | USEFORSITE | 35. | 26864. | 0.00 |
| TESTED | TEAMDELTAGA | GROUPDEFAU | 122. | 26864. | 0.00 |
| TESTED | TEAMDELTAGA | ORGDEFAULT | 1. | 26864. | 0.00 |
| TESTED | TEAMDELTAGA | SITEDEFAUL | 5. | 26864. | 0.00 |

*Obr. 26. Kardinalita sloupců tabulky TESTED.TEAMDELTAGA (Stonawský, 2015)*

Naproti tomu každý ukazatel leaf node indexu na sloupci CUSTOMER tabulky TESTED.PLUSTEAMOMEGA bude průměrně odkazovat na 1 628 RID tabulky (viz obr. 27). Vytvoření takového indexu nelze doporučit. Nároky na alokované místo nevyváží jeho slabou účinnost při hledání konkrétních řádků tabulky.

| TABSCHEMA | TABNAME | COLNAME | COLCARD | TABCARD | CARD_PERC |
|---|---|---|---|---|---|
| TESTED | PLUSTEAMOMEGA | PLUSTEAMOA | 312681. | 312681. | 1.00 |
| TESTED | PLUSTEAMOMEGA | ROWSTAMP | 311296. | 312681. | 0.99 |
| TESTED | PLUSTEAMOMEGA | OWNERID | 143360. | 312681. | 0.45 |
| TESTED | PLUSTEAMOMEGA | OWNERTABL | 9. | 312681. | 0.00 |
| **TESTED** | **PLUSTEAMOMEGA** | **CUSTOMER** | **192.** | **312681.** | **0.00** |
| TESTED | PLUSTEAMOMEGA | ISPRIMARY | 2. | 312681. | 0.00 |
| TESTED | PLUSTEAMOMEGA | TYPE | 1. | 312681. | 0.00 |

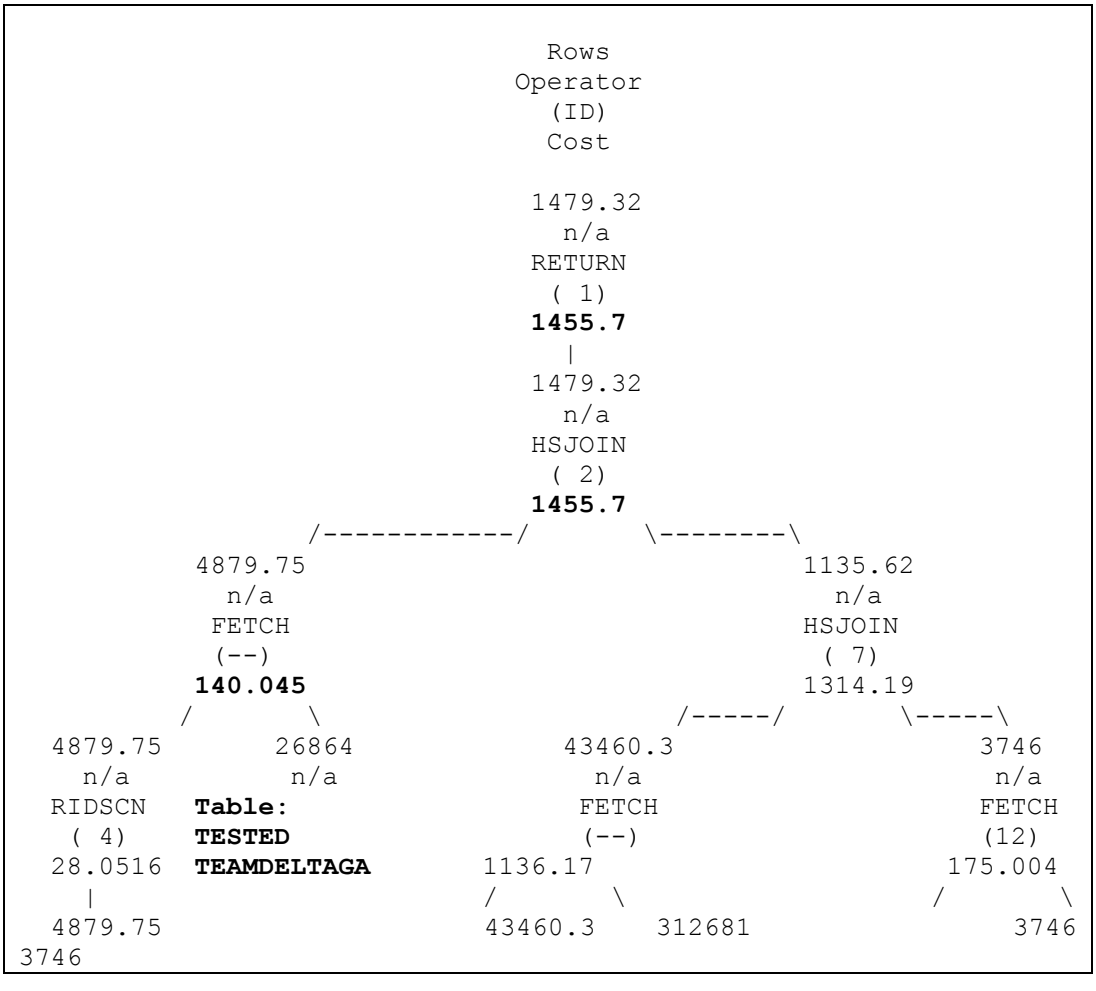
*Obr. 27. Kardinalita tabulky TESTED.PLUSTEAMOMEGA (Stonawský, 2015)*

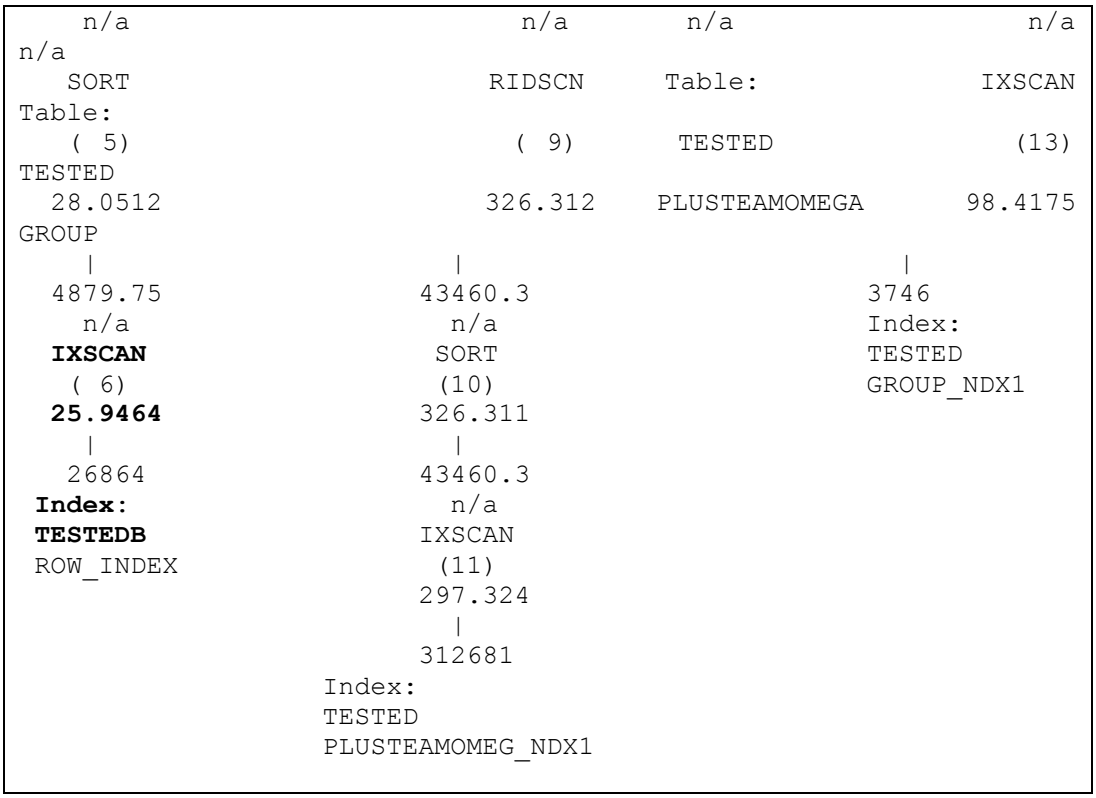
Nový index jsem pojmenoval jako ROW_INDE a spustil funkci RUNSTATS. Bez statistických dat by index nebyl použit. Pomocí tabulkové funkce MON_GET_INDEX jsem verifikoval, že databáze má statistická data o indexu a také jeho velikost, která se shodovala s návrhem Design advisoru, jak je patrné z obr. 28.

| SCHEMA | TABLE | INDEX | CARD | SIZE_MB |
|---|---|---|---|---|
| TESTED | TEAMDELTAGA | SQL140602090818 | 964 | 2.30 |
| TESTED | TEAMDELTAGA | **ROW_INDE** | 26964 | **95.30** |
| TESTED | TEAMDELTAGA | IDX17_AQP_DARSC | 16264 | 26.50 |

*Obr. 28. Alokované místo na úložišti a kardinalita vytvořeného indexu (Stonawský, 2015)*

Po vytvoření indexu jsem si potvrdil pomocí funkce DB2explain plán SQL dotazu (viz obr. 29). Na první pohled je vidět, že došlo k výraznému snížení náročnosti na 1 455 timeron jednotek. DB2 optimalizátor upřednostnil skenování datových stránek indexu před stránkami tabulky.






*Obr. 29. Plán SQL dotazu po vytvoření indexu (Stonawský, 2015)*

Po vytvoření indexu ROW_INDE jsem stejným způsobem měřil dobu trvání SQL dotazu. Provedl jsem celkem 20 kontrolních měření pro minimalizaci peaků. Z tab. 1 je patrné snížení doby běhu SQL dotazu novým indexem v průměru o více než 2 minuty.

*Tab. 1. Měření doby běhu SQL dotazu (Stonawský, 2015)*

| Měření | Bez indexu (min:s:ms) | S indexem (min:s:ms) | Měření | Bez indexu (min:s:ms) | S indexem (min:s:ms) |
|---|---|---|---|---|---|
| 1 | 2:12:780 | 0:03:932 | 11 | 2:11:720 | 0:01:280 |
| 2 | 2:21:514 | 0:01:083 | 12 | 2:06:123 | 0:02:615 |
| 3 | 2:24:954 | 0:03:172 | 13 | 2:14:498 | 0:01:116 |
| 4 | 2:12:082 | 0:01:668 | 14 | 2:14:220 | 0:02:026 |
| 5 | 2:13:719 | 0:01:916 | 15 | 2:12:720 | 0:02:672 |
| 6 | 1:59:737 | 0:02:028 | 16 | 2:15:792 | 0:02:707 |

| | | |
|---|---|---|
| 7 | 2:31:274 | 0:01:650 |
| 8 | 2:08:879 | 0:01:997 |
| 9 | 1:58:949 | 0:01:977 |
| 10 | 2:04:071 | 0:01:655 |

| | | |
|---|---|---|
| 17 | 2:01:834 | 0:02:736 |
| 18 | 2:09:516 | 0:03:773 |
| 19 | 2:14:194 | 0:01:343 |
| 20 | 2:11:380 | 0:02:808 |
| **Průměr** | **2:11:149** | **0:02:257** |

## 3.3 Nexindexovatelné SQL dotazy

Přestože index je výborným nástrojem k optimalizaci běhu SQL dotazů, jsou situace, kdy nelze vytvořit žádný index, který by zmenšil dobu běhu dotazu, neboť je optimalizační proces DB2 donucen procházet všechny datové stránky tabulek. Zvláště se tak děje v případě predikátů druhého stupně jako je operátor OR (viz obr. 30). Takové dotazy je nutné přepsat a umožnit optimalizačnímu procesu skenování indexů.

```
SELECT z.LISTID, MAX(z.CHANGEDATE) as CHANGEDATE
  FROM (SELECT DISTINCT T.LISTID, T.CHANGEDATE
    FROM DB2TEST.TICKET T
    LEFT OUTER JOIN DB2TEST.IRDPARTY TP
    ON (TP.LISTID = T.LISTID and TP.SYSTEMNAME='MEDY'
    and TP.BRIDGED IN (0))
    WHERE ('2014-12-08-07.35.07.405000' < T.CHANGEDATE)
    and T.CHANGEBY NOT IN ('MAXHEN' )
    and T.STATUS NOT IN('CLOSED')
    and T.CUSTOMER = 'MONTO'
    and (
      (TP.TICKETNUM is null AND T.OWNERGROUP IN ('CUSTOMER-DE'))
      OR (TP.TICKETNUM is not null AND TP.SYSTEMNAME = 'MEDY')
        )

   OR T.LISTID IN
   (SELECT LISTID FROM DB2TEST.BRIDGETRANS
   where RIDGESTAT = 'RETRY'
   and SYSTEMNAME='MEDY');

  196 record(s) selected.

Estimated Cost = 236492.54524
```

*Obr. 30. SQL dotaz s operátorem OR (Stonawský, 2015)*

Pomocí operátoru UNION (viz obr. 31) lze rozdělit SQL dotaz na jednotlivé SELECT dotazy čímž umožníme optimalizačnímu procesu používání indexů.

```
SELECT z.LISTID, MAX(z.CHANGEDATE) as CHANGEDATE
  FROM (SELECT DISTINCT T.LISTID, T.CHANGEDATE
    FROM DB2TEST.TICKET T
      LEFT OUTER JOIN DB2TEST.IRDPARTY TP
      ON (TP.LISTID = T.LISTID and TP.SYSTEMNAME='MEDY'
      and TP.BRIDGED IN (0))
      WHERE ('2014-12-08-07.35.07.405000' < T.CHANGEDATE)
      and T.CHANGEBY NOT IN ('MAXHEN' )
      and T.STATUS NOT IN('CLOSED')
      and T.CUSTOMER = 'MONTO'
      and TP.TICKETNUM is null
      AND T.OWNERGROUP IN ('CUSTOMER-DE')

UNION
SELECT DISTINCT T.LISTID, T.CHANGEDATE
  FROM DB2TEST.TICKET T
  LEFT OUTER JOIN DB2TEST.IRDPARTY TP
  ON (TP.LISTID = T.LISTID and TP.SYSTEMNAME='MEDY'
  and TP.BRIDGED IN (0))
  WHERE ('2014-12-08-07.35.07.405000' < T.CHANGEDATE)
  and T.CHANGEBY NOT IN ('MAXHEN' )
  and T.STATUS NOT IN('CLOSED')
  and T.CUSTOMER = 'MONTO'
  and TP.TICKETNUM is not null AND TP.SYSTEMNAME = 'MEDY'

UNION
SELECT DISTINCT T.LISTID, T.CHANGEDATE FROM DB2TEST.TICKET T
  INNER JOIN DB2TEST.BRIDGETRANS BR
  ON T.LISTID = BR.LISTID AND BR.RIDGESTAT = 'RETRY'
  and BR.TICKETSYSTEMNAME='SHARED'
  LEFT OUTER JOIN DB2TEST.IRDPARTY TP
  ON (TP.LISTID = T.LISTID and TP.SYSTEMNAME = 'SHARED');

  196 record(s) selected.

Estimated Cost = 1354.62319
```

*Obr. 31. SQL dotaz s UNION (Stonawský, 2015)*

Rozdíl v timeron jednotkách jsem verifikoval měřením doby původního i přepsaného SQL dotazu. Průměrná doba běhu se snížila o více než 3 minuty, jak je zaznamenáno níže v tab. 2. Samotné nahrazení operátoru OR operátorem UNINON nezpůsobilo naměřené zrychlení. Plán SQL dotazu byl změněn a optimalizační proces tak mohl využít již dříve vytvořené indexy na databázi.

*Tab. 2. Měření doby běhu SQL dotazu s operátorem OR a UNION (Stonawský, 2015)*

| Měření | OR (min:s:ms) | UNION (min:s:ms) | Měření | OR (min:s:ms) | UNION (min:s:ms) |
|---|---|---|---|---|---|
| 1 | 3:12:739 | 0:06:964 | 11 | 2:52:905 | 0:04:076 |
| 2 | 2:21:764 | 0:04:839 | 12 | 2:59:456 | 0:07:266 |
| 3 | 3:24:936 | 0:08:627 | 13 | 3:14:414 | 0:03:334 |
| 4 | 3:12:138 | 0:04:702 | 14 | 2:56:224 | 0:04:306 |
| 5 | 3:13:348 | 0:06:823 | 15 | 3:12:755 | 0:04:416 |
| 6 | 3:12:567 | 0:04:633 | 16 | 3:12:233 | 0:02:580 |
| 7 | 2:49:597 | 0:09:188 | 17 | 3:23:574 | 0:03:731 |
| 8 | 2:55:079 | 0:04:639 | 18 | 2:41:184 | 0:04:084 |
| 9 | 2:58:773 | 0:06:345 | 19 | 3:11:837 | 0:04:012 |
| 10 | 3:21:376 | 0:03:695 | 20 | 3:30:947 | 0:04:067 |
| | | | **Průměr** | **3:05:892** | **0:05:266** |

O trochu lépe než predikáty druhého stupně jsou na tom dotazy typu sub SELECT znázorněné na obr. 32. Část sub SELECT dotazu je zpracovávána přednostně. Pokud není tato část dostatečně komplexní nebo nespecifikujeme silné omezující podmínky v klauzuli WHERE, dochází často k opakovanému procházení stejných řádků v tabulce.

```
SELECT DISTINCT T.LISTID, T.CHANGEDATE
  FROM DB2TEST.TICKET T
  LEFT OUTER JOIN DB2TEST.IRDPARTY TP
  ON (TP.LISTID = T.LISTID and TP.SYSTEMNAME='MEDY'
  and TP.BRIDGED IN (0))
  WHERE T.LISTID IN
  (SELECT LISTID FROM DB2TEST.BRIDGETRANS
  where RIDGESTAT = 'RETRY'
  and SYSTEMNAME='MEDY');

  2231 record(s) selected.

Estimated Cost = 77528.89662
```

*Obr. 32. Sub SELECT SQL dotaz (Stonawský, 2015)*

Sub SELECT dotaz, který nelze omezit specifičtější klauzulí můžeme přepsat pomocí operátoru INNER JOIN (viz obr. 33), který zabrání opakovanému načítání stejných řádků tabulky.

```
SELECT DISTINCT T.LISTID, T.CHANGEDATE
  FROM DB2TEST.TICKET T
  INNER JOIN DB2TEST.BRIDGETRANS BR
  ON T.LISTID = BR.LISTID AND BR.RIDGESTAT = 'RETRY'
  LEFT OUTER JOIN DB2TEST.IRDPARTY TP
  ON (TP.LISTID = T.LISTID and TP.SYSTEMNAME='MEDY'
  and TP.BRIDGED IN (0));

  2231 record(s) selected.

Estimated Cost = 995.56316
```

*Obr. 33. Přepsaný SQL dotaz (Stonawský, 2015)*

Přepsaný SQL dotaz měl zhruba o 76 tis. timeron jednotek nižší nároky na HW zdroje. Provedl jsem 20 měření, abych odstranil případné extrémní hodnoty. V tab. 3 lze vidět, že tato úprava vykázala průměrnou časovou úsporu 33s. Narozdíl od předchozího případu nedošlo ke zpřístupnění již vytvořených indexů. Pouze jsem zabránil redundantnímu čtení stejných dat.

*Tab. 3. Měření doby běhu dotazu se sub SELECT a JOIN (Stonawský, 2015)*

| Měření | subSELECT (min:s:ms) | JOIN (min:s:ms) |
|---|---|---|
| 1 | 0:37:844 | 0:04:476 |
| 2 | 0:35:804 | 0:01:067 |
| 3 | 0:36:319 | 0:04:487 |
| 4 | 0:39:603 | 0:01:137 |
| 5 | 0:32:903 | 0:02:269 |
| 6 | 0:37:710 | 0:01:692 |
| 7 | 0:34:930 | 0:03:490 |
| 8 | 0:32:322 | 0:01:792 |
| 9 | 0:29:470 | 0:02:174 |
| 10 | 0:31:584 | 0:03:432 |

| Měření | subSELECT (min:s:ms) | JOIN (min:s:ms) |
|---|---|---|
| 11 | 0:32:439 | 0:03:583 |
| 12 | 0:37:331 | 0:01:934 |
| 13 | 0:37:818 | 0:05:694 |
| 14 | 0:30:872 | 0:02:567 |
| 15 | 0:38:777 | 0:01:287 |
| 16 | 0:37:173 | 0:01:901 |
| 17 | 0:39:746 | 0:03:753 |
| 18 | 0:30:215 | 0:01:322 |
| 19 | 0:32:990 | 0:04:714 |
| 20 | 0:35:069 | 0:01:234 |
| **Průměr** | **0:35:650** | **0:02:650** |

## 3.4 Fyzické a logické čtení datových stránek

Buffepooly jsou velmi důležitým prvkem databáze, který má enormní vliv na dobu zpracování SQL dotazů. Mají obecně největší podíl na operační paměti alokované databází, proto je důležité určit jejich optimální velikost, aby nedocházelo ke zbytečnému mrhání paměti, kterou by mohly efektivněji využít ostatní aplikace nebo databáze instance.

Sledovaný bufferpool IBMDEFAULTBP měl počáteční kapacitu 200 000 datových stránek o velikosti 32KB. Bufferpool jsem v každém kroku snížil o 10 tis. stránek a pomocí příkazu TIME jsem sledoval dobu běhu paralelně spuštěných testovacích skriptů SELECT.SQL, DELETE.SQL a LOAD.SQL, jak je patrné z obr. 34.

```
CONNECT to TESTE;

--ALTER BUFFERPOOL
ALTER BUFFERPOOL IBMDEFAULTBP SIZE <velikost_stránky>;
```

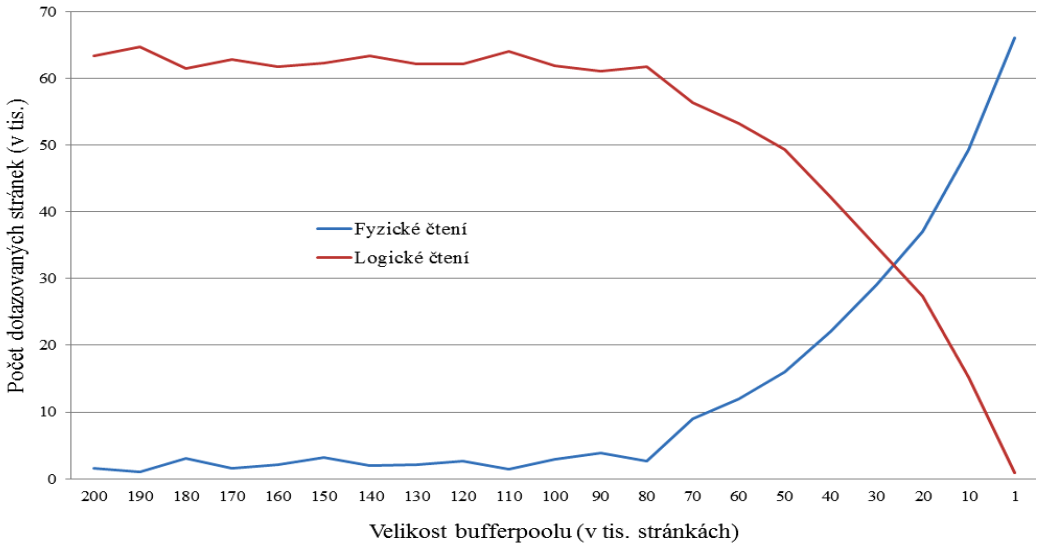
```
--SELECT.SQL
SELECT * FROM DB2TEST.ZAMCI WHERE EMPNO BETWEEN 52364 and 4362335;
```

```
--DELETE.SQL
DELETE FROM DB2TEST.ZAMCI2 WHERE EMPNO BETWEEN 52364 and 4362335;
```

```
--LOAD.SQL
LOAD FROM /tmp/export.ixf of ixf replace into DB2TEST.ZAMCI3;
```

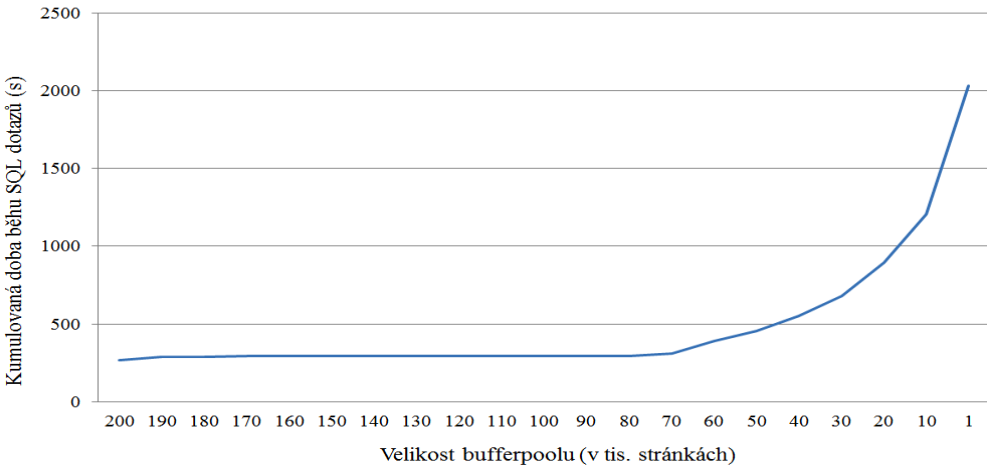
*Obr. 34. Testovací skripty pro SELECT, DELETE a LOAD (Stonawský, 2015)*

Nižší hit-ratio bufferpoolu se začalo projevovat až od hodnoty 70 tis. stránek, protože se množství všech dotazovaných stránek se již nevešlo do kapacity bufferpoolu. Což mělo za následek lineární nárůst počtu I/O operací (obr 35).




*Obr. 35. Fyzické a logické čtení stránek v závislosti na velikosti bufferpoolu (Stonawský, 2015)*

Snižování stránek bufferpoolu pod hodnotu 70 000 mělo významnější dopad na kumulovanou dobu běhu SQL skriptů. Na obr. 36 je zaznamenáno exponenciální zvyšování od této hranice. Bufferpool o velikost 80 tis. stránek by byl ideálním řešením, které by bylo schopné obstarat testované SQL dotazy s akceptovatelným počtem I/O operací.




*Obr. 36. Kumulovaná doba běhu SQL dotazů (Stonawský, 2015)*

Snížením velikosti bufferpoolu jsme ušetřili více než 3GB operační paměti, která tak může být využita jinak. Neúměrně velké bufferpooly s sebou dále nesou riziko přehlédnutí kumulujících se problémů vyplývajících z konstukce databáze v podobě chybějících indexů. To se stává v případě, kdy aplikace opakovaně dotazuje stejná neindexovaná data z tabulky, která se celá vejde do bufferpoolu. V bufferpoolu potom dochází k opakovanému logickému čtení celé tabulky. Skenování celých tabulek je ovšem velmi náročné na cykly procesoru. Proto je nutné sledovat počet transakcí a jejich podíl na logickém čtení. Úplná absence I/O transakcí není nikdy dobrým signálem.

# 4 DB2 BLU VS TRADIČNÍ ORGANIZACE TABULEK

Abychom mohli rovnocenně testovat výkon jednotlivých prvků databáze, je jim nutné nastavit stejné podmínky. Pro obě testované tabulky (DB2TEST.ZAMROW a DB2TEST.ZAMCOL) jsem vytvořil databázi s názvem BLUEA. Tabulky jsou umístěny v tabulkových prostorech (TBS_ROW a TBS_COL) typu autostorage.

Po nainstalování DB2 a vytvoření instance je nutné nastavit proměnnou DB2_WORKLOAD a restartovat instanci. Tyto kroky jsou vizuálně zaznamenány na obr. 37.

```
db2set DB2_WORKLOAD=ANALYTICS
db2stop; db2start
```

*Obr. 37. Příkaz pro nastavení parametru a restartování instance (Stonawský, 2015)*

## 4.1 Vytvoření databáze a objektů

Jakmile je instance restartována, můžeme pokročit k vytvoření databáze. Příkaz pro vytvoření databáze s BLU akcelerací se nijak neliší od standartní syntaxe. Specifikujeme pouze cesty k tabulkovému prostoru a umístění databáze (obr. 38). Musíme mít však na paměti, že BLU je podporováno pouze v tabulkových prostorech typu autostorage.

```
CREATE DATABASE BLUEA ON
'/home/db2test/BLUEA/d1',
'/home/db2test/BLUEA/d2'
dbpath on '/home/db2test/BLUEA';

DB20000I  The CREATE DATABASE command completed successfully.
```

*Obr. 38. Příkaz pro vytvoření databáze BLUE (Stonawský, 2015)*

Příkazem CREATE TABLESPACE (viz obr. 39) si připravíme tabulkové prostory pro obě tabulky. Je možné použít i tabulkový prostor USERSPACE1 vytvořený společně s databází. Pro přehlednější sledování statistik je však výhodnější umístit každou tabulku do jiného tabulkového prostoru.

```
CREATE TABLESPACE TBS_ROW;
DB20000I  The SQL command completed successfully.

CREATE TABLESPACE TBS_COL;
DB20000I  The SQL command completed successfully.
```

*Obr. 39. Příkaz pro vytvoření tabulkových prostorů (Stonawský, 2015)*

Příkaz pro vytvoření tabulek má pouze jednu odlišnost. Na jeho konci pouze specifikujeme klauzuli ORGANIZE BY, kterou určíme, podle jakého kritéria má být tabulka organizována (viz obr. 40 a 41).

```
CREATE TABLE "DB2TEST"."ZAMROW"
                  (
                  "ID" SMALLINT NOT NULL ,
                  "NAME" VARCHAR(9 OCTETS) ,
                  "EMPNO" INTEGER ,
                  "DEPT" SMALLINT ,
                  "JOB" CHAR(5 OCTETS) ,
                  "YEARS" SMALLINT ,
                  "SALARY" DECIMAL(7,2) ,
                  "COMM" DECIMAL(7,2),
                  "PRICE" DECIMAL(30,2) ,
                  "PROMOPRICE" DECIMAL(30,2) ,
                  "PROMOSTART" DATE ,
                  "PROMOEND" DATE
                  )
                 IN "TBS_ROW"
                 ORGANIZE BY ROW;

DB20000I  The SQL command completed successfully.
```

*Obr. 40. Příkaz pro vytvoření tabulky orientované po řádcích (Stonawský, 2015)*

```
CREATE TABLE "DB2TEST"."ZAMCOL"
                  (
                  "ID" SMALLINT NOT NULL ,
                  "NAME" VARCHAR(9 OCTETS) ,
                  "EMPNO" INTEGER ,
                  "DEPT" SMALLINT ,
                  "JOB" CHAR(5 OCTETS) ,
                  "YEARS" SMALLINT ,
                  "SALARY" DECIMAL(7,2) ,
                  "COMM" DECIMAL(7,2),
                  "PRICE" DECIMAL(30,2) ,
                  "PROMOPRICE" DECIMAL(30,2) ,
                  "PROMOSTART" DATE ,
                  "PROMOEND" DATE
```

```
                    )
                   IN "TBS_COL"
                   ORGANIZE BY COLUMN;

DB20000I  The SQL command completed successfully.
```

*Obr. 41. Příkaz vytvoření tabulky sloupcově orientované (Stonawský, 2015)*

## 4.2 Testování výkonu BLU

Výkon obou tabulek jsem se rozhodl otestovat na běžných operacích typu LOAD, SELECT a DELETE. Každá operace byla spuštěna společně s Unixovým příkazem TIME, který měří dobu běhu operace. Mezi jednotlivými testy jsem vždy deaktivoval databázi za účelem vyčištění bufferpoolu, abych zamezil přístupu k datům, které byly uloženy během předchozí operace v paměti.

### 4.2.1 LOAD

Data pro LOAD operaci (viz obr. 42) jsem exportoval z již vytvořené pracovní databáze TEW, která byla vzhledem k velkým tabulkám neobsahující interní data nejlepším kandidátem.

```
connect to TEW ;

declare export cursor database TDW user db2test using for select *
from TDW.EMPLOYE;
DB20000I  The SQL command completed successfully.


connect to BLUEA;

LOAD FROM export OF cursor insert INTO DB2TEST.ZAMCOL;
DB20000I  The SQL command completed successfully.



LOAD FROM export OF cursor insert INTO DB2TEST.ZAMROW;
DB20000I  The SQL command completed successfully.
```

*Obr. 42. Příkaz pro LOAD operaci na obou testovaných tabulkách (Stonawský, 2015)*

REORG a RUNSTATS jsou v případě BLU tabulek součástí LOAD operace, což mělo vliv na dobu jejího trvání, která byla více než 3x delší (viz tab. 4). Výsledky potvrzují fakt, že BLU tabulky najdou lepší využití u datových skladů, kde LOAD

operace probíhají v nižší míře a pravidelnějších intervalech, než jak je tomu v případě OLTP databází. Manuální nastavení parametru UTIL_HEAP_SZ nemělo na dobu trvání LOAD vliv.

*Tab. 4. Měření rychlosti LOAD operace (Stonawský, 2015)*

| Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) |
|---|---|---|
| 1 | 22:40:924 | 7:28:550 |
| 2 | 21:16:282 | 7:02:994 |
| 3 | 20:51:930 | 6:53:657 |
| 4 | 21:28:362 | 6:03:520 |
| 5 | 23:14:220 | 6:35:208 |
| 6 | 25:08:927 | 7:27:338 |
| 7 | 20:48:102 | 6:57:930 |
| 8 | 19:07:301 | 8:11:621 |
| 9 | 22:50:787 | 6:03:225 |
| 10 | 23:33:818 | 6:23:208 |

| Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) |
|---|---|---|
| 11 | 22:31:278 | 6:29:848 |
| 12 | 20:05:703 | 7:15:684 |
| 13 | 19:09:900 | 6:29:384 |
| 14 | 19:18:649 | 7:22:264 |
| 15 | 17:36:187 | 6:11:965 |
| 16 | 23:35:537 | 9:03:495 |
| 17 | 15:22:587 | 6:06:637 |
| 18 | 17:03:566 | 8:07:501 |
| 19 | 17:18:645 | 8:58:753 |
| 20 | 24:23:280 | 6:58:810 |
| **Průměr** | **20:52:299** | **7:06:579** |

### 4.2.2 SELECT

Jelikož je tabulka DB2TEST.ZAMROW orientovaná po řádcích, je nutné před zahájením testování SELECT operace vytvořit index na sloupci ve WHERE klauzuli SQL dotazu. V případě jeho absence by docházelo ke skenování celé tabulky. Pro BLU tabulku DB2TEST.ZAMCOL indexy vytvořit nelze, protože za stejným účelem používají synopsis tabulky.

Před samotným vytvořením indexu je ale nutné zkontrolovat kardinalitu jednotlivých sloupců tabulky (znázorněno na obr. 43). Pokud vytvoříme index s nízkou kardinalitou, kterou vidíme např. na sloupci SALARY, docházelo by k procházení všech datových stránek tabulky spolu se stránkami indexu.

```
TABSCHEMA  TABNAME    COLNAME    COLCARD   TABCARD   CARD_PERC
---------- ---------- ---------- --------- --------- ---------
DB2TEST    ZAMROW     EMPNO       5232354.  5232354.      1.00
DB2TEST    ZAMROW     ID          5232354.  5232354.      1.00
DB2TEST    ZAMROW     NAME        5232301.  5232354.      0.99
DB2TEST    ZAMROW     DEPT             39.  5232354.      0.00
DB2TEST    ZAMROW     JOB             523.  5232354.      0.00
```

```
DB2TEST     ZAMROW      YEARS          9924.    5232354.      0.01
DB2TEST     ZAMROW      SALARY       452648.    5232354.      0.10
DB2TEST     ZAMROW      COMM            1.      5232354.      0.00
DB2TEST     ZAMROW      PRICE         192.      5232354.      0.00
DB2TEST     ZAMCOL      PROMOPRIC     652.      5232354.      0.00
DB2TEST     ZAMCOL      PROMOSTAR       3.      5232354.      0.00
DB2TEST     ZAMCOL      PROMOEND        9.      5232354.      0.00
```

*Obr. 43. Kardinalita jednotlivých řádků tabulky DB2TEST.ZAMROW (Stonawský, 2015)*

Po vytvoření indexu ve sloupci EMPNO je nutné spustit operaci RUNSTATS za účelem aktualizace statistických dat o indexu. Bez operace RUNSTATS by databáze index nepoužila. Tato akce je doložena na obr. 44 níže.

```
CREATE INDEX "DB2TEST"."ZAMROW_ND1"
   ON "DB2TEST"."ZAMROW" ("EMPNO")
   ALLOW REVERSE SCANS;

RUNSTATS ON TABLE "DB2TEST"."ZAMROW" FOR SAMPLED DETAILED INDEX
"DB2TEST"."ZAMROW_ND1";

DB20000I  The SQL command completed successfully.
```

*Obr. 44. Příkaz pro vytvoření indexu na tabulce DB2TEST.ZAMROW (Stonawský, 2015)*

Jako první testovaný SQL dotaz je jednoduchý SELECT s WHERE klauzulí na sloupci EMPNO ideální pro index vytvořený v předchozím kroku (viz obr. 45).

```
SELECT  EMPNO, count(*)
FROM
DB2TEST.ZAMCOL
WHERE EMPNO BETWEEN 231462 and 562335
group by DEPT
with ur;

  242615 record(s) selected.



SELECT  EMPNO, count(*)
FROM
DB2TEST.ZAMROW
WHERE EMPNO BETWEEN 231462 and 562335
group by DEPT
with ur;

  242615 record(s) selected.
```

*Obr. 45. Testovaný dotaz SELECT s WHERE klauzulí na obou tabulkách (Stonawský, 2015)*

Přestože má tento index vysokou kardinalitu zaručující, že bude skenován pouze zlomek jeho datových stránek, je zřejmé, že BLU tabulka DB2TEST.ZAMCOL vykazuje lepší hodnoty (viz tab. 5). Přeskakování dat pomocí synopsis tabulek je mnohem efektivnější operací než procházení datových stránek indexu.

*Tab. 5. Měření rychlosti SELECT dotazu s WHERE klauzulí (Stonawský, 2015)*

| Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) |
|---|---|---|
| 1 | 0:03:371 | 0:29:675 |
| 2 | 0:04:094 | 0:28:517 |
| 3 | 0:03:388 | 0:26:276 |
| 4 | 0:02:995 | 0:23:818 |
| 5 | 0:03:253 | 0:29:862 |
| 6 | 0:04:635 | 0:28:932 |
| 7 | 0:02:130 | 0:28:199 |
| 8 | 0:03:771 | 0:28:505 |
| 9 | 0:02:216 | 0:27:970 |
| 10 | 0:02:282 | 0:28:836 |

| Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) |
|---|---|---|
| 11 | 0:03:215 | 0:27:963 |
| 12 | 0:02:305 | 0:27:206 |
| 13 | 0:04:966 | 0:25:913 |
| 14 | 0:03:223 | 0:23:041 |
| 15 | 0:02:558 | 0:24:692 |
| 16 | 0:04:709 | 0:29:606 |
| 17 | 0:03:361 | 0:27:712 |
| 18 | 0:02:212 | 0:29:264 |
| 19 | 0:02:169 | 0:24:399 |
| 20 | 0:02:424 | 0:26:071 |
| **Průměr** | **0:03:163** | **0:27:322** |

Druhý testovaný dotaz SELECT ALL je soustředěn na měření rychlosti skenování celé tabulky a je zaznamenán na obr 46.

```
SELECT *
FROM
DB2TEST.ZAMCOL
with ur;

  5232354 record(s) selected.


SELECT *
FROM
DB2TEST.ZAMROW
with ur;

  5232354 record(s) selected.
```

*Obr. 46. Testovaný dotaz SELECT ALL na obou tabulkách (Stonawský, 2015)*

Naměřené hodnoty trvání obou dotazů jsou téměř totožné, jak je patrné z tab. 6. SELECT je na tabulkách s BLU akcelerací rychlejší pouze při omezení počtu vybraných řádků. V obou případech dochází ke skenování všech datových stránek tabulky. Synopsis tabulky a index nejsou využity ke zrychlení běhu dotazu.

*Tab. 6. Měření rychlosti SELECT ALL dotazu (Stonawský, 2015)*

| Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) | Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) |
|---|---|---|---|---|---|
| 1 | 2:50:320 | 3:08:491 | 11 | 2:52:841 | 2:26:238 |
| 2 | 2:49:266 | 2:43:405 | 12 | 1:59:364 | 2:41:175 |
| 3 | 2:53:179 | 2:42:921 | 13 | 2:41:172 | 3:00:056 |
| 4 | 2:59:552 | 3:01:951 | 14 | 3:06:530 | 2:40:479 |
| 5 | 2:55:558 | 2:32:660 | 15 | 2:54:168 | 2:37:433 |
| 6 | 3:01:727 | 2:44:620 | 16 | 2:58:973 | 2:32:905 |
| 7 | 2:55:842 | 2:45:684 | 17 | 2:58:990 | 3:04:401 |
| 8 | 2:44:141 | 3:12:436 | 18 | 3:02:591 | 2:23:964 |
| 9 | 2:57:997 | 2:30:015 | 19 | 2:46:719 | 2:40:599 |
| 10 | 2:44:009 | 2:34:429 | 20 | 2:46:571 | 2:42:968 |
| | | | **Průměr** | **2:50:975** | **2:44:341** |

### 4.2.3 DELETE

Jako poslední jsem měřil dobu běhu DELETE dotazu v rozsahu 242 615 řádků. Z obr. 47 je patrné, že se jedná o stejný počet řádků jako u dotazu SELECT s WHERE klauzulí.

```
DELETE FROM
DB2TEST.ZAMCOL
WHERE EMPNO BETWEEN 231462 and 562335;
DB20000I  The SQL command completed successfully.


DELETE FROM
DB2TEST.ZAMROW
WHERE EMPNO BETWEEN 231462 and 562335;
DB20000I  The SQL command completed successfully.
```

*Obr. 47. Testovaný dotaz DELETE na obou tabulkách (Stonawský, 2015)*

Ačkoliv má DB2 za úkol vymazat stejný počet řádků, které načítá v případě SELECT dotazu, je tato operace prováděna mnohem pomaleji na tabulce s BLU akcelerací a naopak rychleji na tabulce organizované pomocí řádků (viz tab. 7). Úprava synopsis tabulek je tedy mnohem náročnější akcí nežli smazání záznamů z indexů.

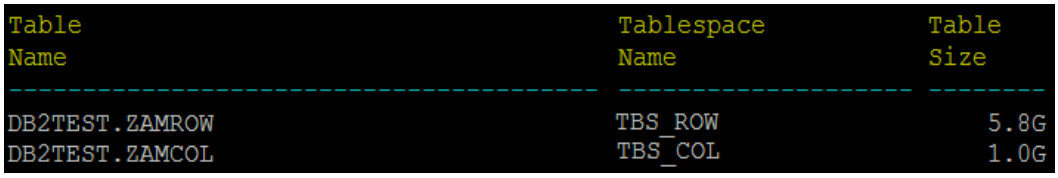
*Tab. 7. Měření rychlosti DELETE dotazu (Stonawský, 2015)*

| Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) | Měření | ZAMCOL (min:s:ms) | ZAMROW (min:s:ms) |
|---|---|---|---|---|---|
| 1 | 0:35:955 | 0:14:234 | 11 | 0:27:459 | 0:16:509 |
| 2 | 0:24:922 | 0:13:303 | 12 | 0:33:210 | 0:12:672 |
| 3 | 0:35:504 | 0:14:243 | 13 | 0:36:705 | 0:14:021 |
| 4 | 0:29:487 | 0:12:262 | 14 | 0:32:457 | 0:14:528 |
| 5 | 0:22:430 | 0:17:400 | 15 | 0:27:052 | 0:16:116 |
| 6 | 0:26:763 | 0:13:672 | 16 | 0:23:623 | 0:14:570 |
| 7 | 0:23:156 | 0:14:956 | 17 | 0:34:482 | 0:16:565 |
| 8 | 0:29:938 | 0:17:974 | 18 | 0:25:733 | 0:14:014 |
| 9 | 0:30:447 | 0:17:686 | 19 | 0:24:697 | 0:16:239 |
| 10 | 0:24:447 | 0:17:100 | 20 | 0:25:078 | 0:14:809 |
| | | | **Průměr** | **0:28:677** | **0:15:693** |

## 4.3 Komprese

Při LOAD fázi bylo do obou tabulek importováno 5 232 354 řádků. Pomocí monitorovací funkce DB2TOP je možné porovnat velikost testovaných tabulek (graficky znázorněno na obr. 48). Rozdíl velikosti mezi tabulkami je patrný na první

pohled. K velikosti tabulky DB2TEST.ZAMROW je připočten i vytvořený index DB2TEST.ZAMROW_ND1. Obě tabulky jsou komprimovány stejným způsobem pomocí slovníkové metody. Rozdíl je pouze v generaci hodnot slovníku. Pro tabulku DB2TEST.ZAMROW jsou tvořeny pomocí řádků, u tabulky DB2TEST.ZAMROW hodnotami jednotlivých sloupců.



| Table Name | Tablespace Name | Table Size |
|---|---|---|
| DB2TEST.ZAMROW | TBS_ROW | 5.8G |
| DB2TEST.ZAMCOL | TBS_COL | 1.0G |

*Obr. 48. Velikost tabulek v DB2TOP (Stonawský, 2015)*

Pro určení míry komprese jsem vytvořil tabulku DB2TEST.ZAMBEZCOMP, která nevyužívá žádnou kompresi a podobně jako v předchozích příkladech použil operaci LOAD k importu dat. Data tabulky a index alokoval na úložišti v nekomprimované podobě 11,7GB. Informace o kompresi (obr. 49) zjistíme pomocí funkce SYSPROC.ADMIN_GET_TAB_COMPRESS_INFO.

```
select T.TABSCHEMA,T.TABNAME,
      sum(T.PCTPAGESSAVED_ADAPTIVE) as PCTPAGESSAVED_ADAPTIVE,
      sum(TA.DATA_OBJECT_P_SIZE)/1024 as OBJECTSIZE_MB
      from ta-
ble(sysproc.admin_get_tab_compress_info('DB2TEST','')) T
      join TABLE (sysproc.admin_get_tab_info('DB2TEST', ''))
TA
      ON T.TABSCHEMA = TA.TABSCHEMA AND T.TABNAME = TA.TABNAME
      group by T.TABSCHEMA,T.TABNAME;


TABSCHEMA  TABNAME       PCTPAGESSAVED_ADAPTIVE OBJECTSIZE_MB
---------- ------------ ---------------------- --------------
DB2TEST    ZAMBEZCOMP                       49          11878
```

*Obr. 49. Velikost tabulky a potenciální míra komprese (Stonawský, 2015)*

Porovnáním velikosti nekomprimované tabulky a tabulek testovaných, zjišťujeme, že tabulka s BLU akcelerací efektivněji využívá slovníkové metody komprese. Tento klíčový výsledek shrnuje tab. 8.

*Tab. 8. Velikost tabulek a míra komprese (Stonawský, 2015)*

| | **Alokované místo (MB)** | **Míra komprese (%)** |
|---|---|---|
| **DB2TEST.ZAMCOL** | 1 034 | 91,7 |
| **DB2TEST.ZAMROW** | 5 952 | 49 |

BLU akcelerace s sebou nese výhody i nevýhody oproti klasicky orientovaným tabulkám. Faktem ale zůstává, že tyto tabulky spolu mohou koexistovat v jedné databázi. Při navrhování nových databází se nám tedy nabízí možnost využít to nejlepší z obou systémů ukládání dat. Musíme však mít na paměti, že společně s implementací BLU akcelerace ztratí databáze spoustu ostatních funkcí.

# ZÁVĚR

Nedostatečná výkonnost databáze je nejčastěji identifikována podnětem od uživatelů o nevalné rychlosti jejich transakcí. Za pomalým během SQL dotazů se může skrývat více příčin.

V majoritě případů, s kterými jsem se v praxi setkal, se jen jednalo o absenci indexu. Před rozhodnutím o vytvoření indexu je nutné sledovat ukazatel kardinality. Procházení datových stránek indexů s nízkou kardinalitou je mnohdy náročnější a delší proces než čtení celé tabulky a tudíž značně neefektivní. Naproti tomu je příhodné vytvářet indexy na sloupcích s vysokou kardinalitou, které jsou vhodně specifikovány v predikátech WHERE klauzule. Bez přesného omezení rozsahu požadovaných hodnot, ztrácejí indexy smysl. Rovněž nemohu doporučit ledabylé generování indexů pro tabulky, které jsou určeny převážně pro operace INSERT a DELETE. Četnost použití jednotlivých indexů je taktéž ukazatelem jejich účinnosti, protože málo používaný index přinese ve většině případů víc škody než užitku.

Index též klade velké nároky na alokované místo ve fyzickém úložišti, zvláště při nesprávné identifikaci kardinality nebo s definicí zahrnutí většiny sloupců tabulky. Špatné indexy mohou dokonce přesáhnout velikost tabulky a způsobit vážné výkonnostní problémy při INSERT a DELETE operacích.

Jsou ale případy, kdy aplikace dotazuje již indexovaná data a přesto dochází ke skenování celé tabulky. Zde se jedná neúčinný kód aplikace. Přepisování chyb v SQL dotazech jako jsou predikáty druhého stupně, nedostatečně komplexní sub SELECT dotazy a nepříliš restriktivní WHERE klauzule, je sice časově náročnější, ale vede k lepším výsledkům než jak je tomu při prosté generaci nových indexů. Nevytváříme totiž nové objekty v databázi, které vyžadují další péči. Nepřímo tím snižujeme dobu pravidelných operací RUNSTATS a REORG, které mohou být příčinou delší odstávky databáze v době pravidelné údržby.

Délka trvání SQL dotazů je taktéž ovlivněna alokovanou operační pamětí databáze. Největší roli zde hrají bufferpooly. Příliš malé bufferpooly nejsou často schopny pojmout veškeré dotazované datové stránky, neustále k nim přistupují na disk, čímž dochází k nárůstu pomalých I/O operací. Naproti tomu neúměrně velké bufferpooly mají za následek zkreslení ukazatele hit-ratio, který je mnohdy

mylně interpretován. 100% hit-ratio bufferpoolu je sice vítanou hodnotou, která znamená minimální přístup k datovým stránkám na disku, ale také se za ni může pohodlně skrývat absence indexu. Proto je vhodnější zvolit menší bufferpooly a v případě nutnosti navyšovat počet jejich datových stránek.

S rozvojem nových a náročnějších aplikací zvláště v analytickém prostředí je třeba vyvíjet nové způsoby zpracování dat. DB2 dokazuje, že není nutné budovat další databázový systém. Do stávajícího implementovala BLU akceleraci, která přináší mnohem rychlejší zpracování SELECT dotazů pomocí sdružených synopsis tabulek díky přeskakování nepotřebných záznamů. Nová organizace tabulek orientovaná na sloupce zlepšuje účinnost slovníkové komprese, čímž dochází k úspoře a hardwarových a potažmo finančních zdrojů. Je třeba zmínit, že i tato mince má svoji druhou stranu. Jakákoli jiná manipulace s daty je o řád pomalejší než u indexovaných tabulek a absence podpory některých kritických funkcí DB2 ji odsouvá do pozadí. V současnosti je BLU akcelerace stále ve vývoji a na své uplatnění v praxi čeká.

# SEZNAM POUŽITÉ LITERATURY

[1] ARNOLD, Danny et al. *Unleashing DB2 10 for Linux, UNIX, and Windows*. United States: IBM, International Technical Support Organization, 2012, 162 s. ISBN 07-384-3710-7.

[2] columbia.edu. *Administration Guide for DB2*, IBM Česká republika, spol. s r.o. Praha. 2015 [online]. [cit. 2015-03-21]. Dostupné z: https://www.columbia.edu/sec/acis/db2/db2d0/db2d003.htm#Header_1

[3] GANDHI, Nitin. *DB2 :Programmer's Guide*. [online]. Chicago, Illinois, United States 2011 [cit. 2015-02-08]. Dostupné z: http://db2guide.blogspot.cz/

[4] GANDHI, Nitin. *DB2 Real Time* [online]. Chicago, Illinois, United States 2013 [cit. 2015-02-01]. Dostupné z: http://db2guide.blogspot.cz/

[5] CHAINANI, Naresh a kol.. *DB2 with BLU Acceleration: A rapid adoption guide*. United States: IBM DeveloperWorks. 2013, 115 s.. Dostupné také z: http://www.ibm.com/developerwo rks/data/library /techarticle/dm-130 9db2bluaccel/dm-130db2bluaccel-pdf.pdf

[6] CHEN, Whei-Jen et al. *High Availability and Disaster Recovery Options for DB2 on Linux, Unix, and Windows*. United States: IBM, International Technical Support Organization, 2009, 856 s. ISBN 07-384-3138-9.

[7] CHEN, Whei-Jen et al. *Leveraging DB2 10 for High Performance of Your Data Warehouse*. United States: IBM, International Technical Support Organization, 2014, 218 s. ISBN 07-384-3897-9.

[8] CHEN, Whei-Jen. *Database partitioning, table partitioning, and MDC for DB2 9*. 1st ed. United States: IBM, International Technical Support Organization, 2007, 252 s. ISBN 07-384-8922-0.

[9] CHEN, Whei-Jen. *DB2 integrated cluster environment deployment guide*. 1st ed. San Jose, Calif.: International Technical Support Organization, c2004, 408 s. IBM redbooks. ISBN 07-384-9082-2.

[10] IBM *DB2 10.5 for Linux, UNIX, and Windows SQL Procedural Languages: Application Enablement and Support.* United States, 2014b, 941 s.

[11] IBM *DB2 10.5 for Linux, UNIX, and Windows: Troubleshooting and Tuning Database Performance*. United States, 2015b,793 s.

[12] IBM Knowledge Center. *IBM Knowledge Center*. IBM Česká republika, spol. s r.o. Praha. 2015 [online]. [cit. 2015-03-13]. Dostupné z: http://www-01.ibm.com/support/knowledgecenter/

[13] IBM. *Database Administration Concepts and Configuration Reference.* United States, 2015a, 437 s.

[14] IBM. *DB2 for Linux, UNIX, and Windows: Globalization Guide*. United States, 2014a, 861 s.

[15] NEAGU, Adrian a Robert PELLETIER. *IBM DB2 9.7 Advanced Administration Cookbook*. United States: Packt Publishing, 2012, 480 s. ISBN 9781849683326.

[16] SANDERS, Roger. The DB2 Multi-Temperature Data Management Feature. *DBA Society* [online]. United States Vol.1, 2015 [cit. 2015-02-04]. Dostupné z: https://community.emc.com/community/connect/dba-society/blog/2015/01/19/the-db2-multi-temperature-data-management-feature

# SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

| | |
|---|---|
| AESE | Advanced enterprice server edition |
| CPU | Central processing unit |
| DBM | Database manager |
| DCL | Data control language |
| DDL | Data definition language |
| DMS | Database managed tablespace |
| DPF | Database partitioning feature |
| HADR | High availability disaster recovery |
| I/O | Input/Output |
| IXF | Integration exchange format |
| LOB | Large object |
| LUW | Linux Unix Windows |
| MQT | Materialized query table |
| OLTP | Online transaction processing |
| RID | Row indentification |
| SLA | Service level agreement |
| SMS | System managed tablespace |
| SQL | Structured query language |
| TCL | Transaction Control Language |
| TSM | Tivoli storage manager |
| TSN | Tuple sequence number |

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK