



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ

FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY

INSTITUTE OF INFORMATICS

**SAP REPORTING S VYUŽITÍM IN-MEMORY
DATABÁZE**

SAP REPORTING WITH THE USE OF IN-MEMORY DATABASE

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Václav Smejkal

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Jan Luhan, Ph.D., MSc

BRNO 2018

Zadání diplomové práce

Ústav: Ústav informatiky
Student: **Bc. Václav Smejkal**
Studijní program: Systémové inženýrství a informatika
Studijní obor: Informační management
Vedoucí práce: **Ing. Jan Luhan, Ph.D., MSc**
Akademický rok: 2017/18

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

SAP reporting s využitím in-memory databáze

Charakteristika problematiky úkolu:

Úvod
Cíle práce, metody a postupy zpracování
Teoretická východiska práce
Analýza současného stavu
Vlastní návrhy řešení
Závěr
Seznam použité literatury
Přílohy

Cíle, kterých má být dosaženo:

Cílem práce je navrhnout reportovací aplikaci v prostředí SAP (napsaná v jazyce ABAP), která uživateli prostřednictvím zvolených parametrů pro dotazy nad DB umožní využití připravených dotazů v systémech MaxDB a HANA DB (HDB). Výstupem práce bude porovnání rychlosti HDB a MaxDB v závislosti na typech dotazů, uspořádání dat (porovnání sloupcově a řádkově orientovaného uspořádání dat v DB) a možné aplikovatelnosti v reálném prostředí.

Základní literární prameny:

KÜHNHAUSER, K-H. ABAP: výukový kurz. 1. vyd. Brno: Computer Press, 2009. 368 s. ISBN 978-80-251-2117-7.

LABERGE, R. Datové sklady - Agilní metody a business intelligence. Praha: Computer Press, 2012. 352 s. ISBN 978-80-251-3729-1.

NOVOTNÝ, O., J. POUR a D. SLÁNSKÝ. Business Intelligence: jak využít bohatství ve vašich datech. 1. vyd. Praha: Grada, 2005. 256 s. ISBN 80-247-1094-3.

PLATTNER, H. A Course in In-Memory Data Management. 1st ed. Potsdam: Springer, 2013. 297 p. ISBN 978-3-642-44314-5.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2017/18

V Brně dne 28.2.2018

L. S.

doc. RNDr. Bedřich Půža, CSc.
ředitel

doc. Ing. et Ing. Stanislav Škapa, Ph.D.
děkan

Abstrakt

Diplomová práce se zabývá in-memory databází SAP HANA, která uchovává všechna data přímo v operační paměti s využitím sloupcově orientovaného uspořádání dat. Praktickou součástí práce je vytvoření aplikace v prostředí SAP R/3, s jejíž pomocí je porovnán výkon in-memory databáze SAP HANA a tradiční databáze MaxDB, včetně vlivu použitého typu uspořádání dat. Výsledky ukazují, že in-memory databáze je výhodná zejména pro analytické operace založené na agregačních funkcích.

Klíčová slova

reporting, in-memory databáze, SAP, HANA, MaxDB, ABAP, sloupcově orientované uspořádání dat

Abstract

The master's thesis deals with the in-memory database SAP HANA which keeps all data directly in main memory with the use of column-oriented data layout. Practical part of the thesis consists in development of application in SAP R/3 environment, with which performance of the in-memory database SAP HANA and the traditional database MaxDB is compared, including influence of used data layout. The results show that the in-memory database is advantageous especially for analytical operations based on aggregate functions.

Key words

reporting, in-memory database, SAP, HANA, MaxDB, ABAP, column-oriented data layout

Bibliografická citace

SMEJKAL, V. *SAP reporting s využitím in-memory databáze*. Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2018. 82 s. Vedoucí diplomové práce Ing. Jan Luhan, Ph.D., MSc.

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně dne 15. května 2018

podpis autora

Poděkování

Na tomto místě bych chtěl poděkovat svému vedoucímu diplomové práce Ing. Janu Luhanovi, Ph.D., MSc., za vstřícný přístup, cenné rady a odborné vedení při zpracování této práce.

OBSAH

ÚVOD	10
1 CÍLE PRÁCE	12
1.1 Dílčí cíle a metody práce	12
2 TEORETICKÁ VÝCHODISKA PRÁCE	14
2.1 Motivace.....	14
2.1.1 Nové požadavky	14
2.1.2 Nové schopnosti hardwaru	17
2.1.3 OLTP vs. OLAP	20
2.2 In-memory databáze SAP HANA	23
2.2.1 Architektura.....	23
2.2.2 Uspořádání dat.....	30
2.2.3 Slovníkové kódování.....	34
2.2.4 Další metody komprese dat	36
2.2.5 Partitioning	38
2.2.6 Operace v in-memory databázi.....	38
3 ANALÝZA SOUČASNÉHO STAVU	44
3.1 Shrnutí teoretických poznatků.....	44
3.2 Business Intelligence	46
3.2.1 Základní informace o BI	46
3.2.2 Architektura BI	48
3.2.3 Trendy vývoje v BI	49
3.3 SAP MaxDB.....	50
3.4 SAP S/4 HANA.....	51
3.5 Vývojové prostředí	52
3.5.1 Systémy	52
3.5.2 ABAP	53
3.6 Testovací data.....	54
3.6.1 Schéma databáze.....	55
3.6.2 Uspořádání dat.....	56

3.6.3	Objem testovacích dat	56
3.7	Modely testování	56
4	VLASTNÍ NÁVRH ŘEŠENÍ	58
4.1	Reportovací aplikace.....	58
4.1.1	Tabulky.....	58
4.1.2	Návrh a vývoj aplikace.....	59
4.2	Výsledky měření.....	64
4.2.1	Výsledky vkládání	64
4.2.2	Spotřeba paměti	65
4.2.3	Testovací dotazy	66
4.3	Zhodnocení výsledků.....	74
	ZÁVĚR.....	76
	SEZNAM POUŽITÝCH ZDROJŮ	78
	SEZNAM POUŽITÝCH OBRÁZKŮ	81
	SEZNAM POUŽITÝCH GRAFŮ	82

ÚVOD

Dnešní společnosti produkují velké množství dat, které potřebují zpracovat a analyzovat. Běžně používají dvě databáze, OLTP (On-line Transaction Processing) a OLAP (On-line Analytical Processing). Zatímco OLTP je optimalizovaná pro běh velkého objemu krátkých transakcí, provoz s vícenásobným přístupem a celkově uzpůsobena pro pracovní zátěž, kde převládají operace zápisu do databáze, OLAP je navržena pro menší množství déle trvajících operací, kde je kladen důraz na čtení z databáze. Zároveň OLAP typicky používá předdefinované a předem vypočítané agregace. Z důvodu existence těchto dvou konceptů je potřeba replikovat změny dat mezi oběma databázemi, konkrétně z OLTP na OLAP. K tomu se používá ETL proces, který je časově a výkonově značně náročný, a tak zvyšuje složitost celého systému a také přináší redundanci dat.

Když vezmeme v úvahu skutečnost, jakým vývojem v posledním desetiletí prošel hardware, zejména uvažujeme výkon procesorů (CPU) a pamětí, je již technologicky možné provozovat celou databázi přímo v operační paměti. V posledních letech se objevil nový přístup v podobě OLAP in-memory databáze, která používá sloupcově orientované uspořádání záznamů v paměti namísto tradičního řádkového uspořádání dat. Ve sloupcově orientovaném uspořádání jsou data uložena po jednotlivých attributech tabulky, což má zvýšit rychlost operací sekvenčně procházejících sloupce (atributy) v analytických operacích.

Profesor Hasso Plattner, jeden ze spoluzakladatelů společnosti SAP, která se zaměřuje na vývoj podnikových informačních systémů, spolu se svými kolegy přišel s nápadem spojit OLTP a OLAP v jednotný databázový systém, navzdory velmi rozdílným typům pracovních zátěží, které jsou typické pro jednu či druhou databázi. Chtěli k tomu také využít již zmíněné sloupcově orientované uspořádání dat, které by tak bylo použito i pro krátké a frekventované transakční operace.

Po letech výzkumu firma SAP představila in-memory databázi SAP HANA, která implementuje výše zmíněný koncept. Je tedy založena na in-memory databázové technologii, to znamená, že všechna data jsou uložena přímo v rychlé paměti (DRAM). Kromě sloupcově orientovaného uspořádání dat je také použito slovníkové kódování. To hraje velmi důležitou roli, protože má dosahovat vysoké míry komprese dat a zvyšovat

výkon analytických operací, zejména při selekci specifických atributů a u agregačních operací. S použitím jednotného databázového systému Navíc odpadá nutnost kopírovat data mezi oddělenými OLTP a OLAP systémy. To přináší výrazné snížení redundance dat a složitosti systému a zvýšení jeho flexibility. ETL proces již není potřeba. Použití sloupcového uspořádání dat má také umožnit spouštění analytických operací v reálném čase, což by odstranilo potřebu předem vypočítaných agregací, a tedy přineslo další snížení redundance. Na druhou stranu je potřeba brát v úvahu zdánlivou nevýhodu sloupcového uspořádání v transakčním provozu (OLTP). Tento problém je v databázi SAP HANA řešen heuristikami, které budou v této práci popsány (Plattner, 2013).

1 CÍLE PRÁCE

Cílem této práce je popsat architekturu a nové technologie, které jsou spjaty s in-memory databází SAP HANA a poskytnout tak teoretický základ pro zbytek práce. Dalším cílem je vytvořit reportovací aplikaci v prostředí SAP R/3 a programovacím jazyce ABAP, která bude měřit odezvu této databáze na analytické dotazy oproti tradiční databázi MaxDB, a také sloupcově orientovaného uspořádání dat ve srovnání s tradičním řádkovým uspořádáním. Měření budou provedena jak na operacích zápisu (vkládání záznamů) do databáze, tak zejména na analytických operacích, tedy dotazech nad databázemi. Jako testovací dataset budou použita veřejně dostupná data z měření meteorologické Hvězdárny Brno, která obsahují údaje o počasí za posledních pět let (Datahub, 2017). Na základě provedených testů a měření bude popsán možný přínos této nové technologie v praxi.

1.1 Dílčí cíle a metody práce

Prvním cílem práce je poskytnout teoretický základ k pochopení existence a funkce in-memory databáze SAP HANA. Nejprve budou popsány skutečnosti vedoucí k motivaci vytvořit takový typ databáze. Mezi ně patří požadavky dnešních velkých firem na analýzu databázových dat a také nové schopnosti hardwarových komponent. Následně bude charakterizována podoba současných databázových systémů a jejich součástí OLTP a OLAP.

Další cílem je popis základních charakteristik in-memory databáze SAP HANA. Bude popsána její architektura a další znaky, které jsou pro ni typické. Jedná se zejména o sloupcově orientované uspořádání dat, organizace paměťového prostoru a specifika implementace základních databázových operací.

Jelikož užití in-memory databáze úzce souvisí s oblastí Business Intelligence, budou o BI uvedeny základní informace, architektura a trendy vývoje. Dále budou přiblíženy základní charakteristiky tradiční databáze MaxDB a aktuální produkt SAP S/4 HANA, postavený na stejnojmenné in-memory databázi. Následně je cílem představit vývojové prostředí programovacího jazyku ABAP, ve kterém bude napsána reportovací aplikace, a sadu testovacích dat, která bude využita k měření výkonu obou databází.

Stěžejním cílem praktické části této práce je vytvořit reportovací aplikaci v prostředí SAP R/3, která je schopna běhu na obou zkoumaných databázích: tradiční MaxDB, in-memory HANA. Jejím účelem je testovat odezvu databází prostřednictvím předem připravených dotazů, jejichž parametry může uživatel nastavit. Dále nabídne funkcionalitu pro správu databázových tabulek, tj. možnost odstranění a importu testovacích dat. Výsledkem bude naměřená rychlost zpracování dotazu v řádkově a sloupcově orientovaném uspořádání dat a samotný výstup dotazu ve formě grafu. Aplikace bude postupně spuštěna na obou databázových systémech a jejich výkon bude změřen na 7 vybraných testovacích dotazech. Dalším kritériem hodnocení pak bude rychlost vkládání dat do tabulek a jejich spotřeba paměti. Na závěr budou získané výsledky zhodnoceny a nastíněn možný přínos technologie in-memory databáze v praxi.

2 TEORETICKÁ VÝCHODISKA PRÁCE

2.1 Motivace

V této části práce budou popsány aspekty, které vedly k motivaci profesora Hasso Plattnera a jeho spolupracovníky vytvořit nový koncept in-memory databáze se sloupcově uspořádanými daty, která spojuje OLTP a OLAP systémy do jednoho. Hlavním zdrojem inspirace této práce byla kniha od profesora Hasso Plattnera, *A Course in In-Memory Data Management* (Plattner, 2013), ve které vysvětluje důvody, proč je potřeba vyvinout nový databázový systém, později představený jako SAP HANA.

2.1.1 Nové požadavky

Firmy dnes produkují obrovské množství dat, které potřebují uchovávat v databázi. Příkladem mohou být různé druhy senzorů nebo výrobní roboti, hrající nezanedbatelnou roli v produkci těchto dat. Mnohé organizace navíc také spravují svá data v mnohem větším měřítku a perspektivě. To znamená, že monitorují trendy, chování konkurence nebo využívají systémy pro podporu rozhodování. Je také běžné, že svá data uchovávají v různých formách, a kombinují tak strukturovaná a nestrukturovaná data. Mimo jiné má tento trend za následek vznik nových, vyšších požadavků na podnikové informační systémy. Aby byly splněny požadavky na jednotu a flexibilitu, jednotný databázový systém by měl být schopen kombinovat všechna data z různých zdrojů. Jelikož dnešní doba vyžaduje rychlé rozhodování, takový systém musí poskytnout analýzu dat v reálném čase a výsledky výpočtů za běhu za účelem podpory interaktivního rozhodování (Plattner, 2013, s. 7).

2.1.1.1 Data událostí

Jak bylo zmíněno výše, organizace potřebují ukládat mnoho dat, které následně zpracovávají, uchovávají a analyzují s použitím ERP systému prostřednictvím podnikových aplikací. Jedním z nejvíce využitelných typů dat mohou být data událostí. V porovnání s tradičními podnikovými daty jsou jednotlivá data událostí obvykle velmi malá (v řádech bajtů a kilobajtů), avšak pro každou entitu se jich vygeneruje obrovské množství.

Pro lepší představu, nejtypičtějším typem dat událostí jsou data sbíraná ze senzorů. Ty jsou běžně používány jako sledovací mechanismus zboží při logistických operacích. Protože každá sledovaná položka obsahuje unikátní kód produktu, implementovaný s pomocí RFID (Radio Frequency Identification), může být jednoznačně identifikována v každém v každém kroku logistického řetězce, tj. během balení nebo odesílání položky. V dalším případě jsou tyto senzory využívány ve farmaceutickém průmyslu, kde je požadavek na sledování velkého množství léčiv v reálném čase a musí se zavádět opatření proti jejich padělání. Nebo si můžeme představit případ moderních závodních aut, která produkují nadměrná množství sensorových dat. Tato data pak musí být analyzována také v reálném čase, např. v průběhu závodu pro podporu rozhodování o výběru pneumatik či o zastávce v boxech (Plattner, 2013, s. 7-9).

2.1.1.2 Kombinace strukturovaných a nestrukturovaných dat

V databázích jsou obvykle ukládána strukturovaná data, která mohou být počítači jednoduše a automaticky zpracována, a jsou zde ve formě relačních databází či stromových strukturách v případě objektových databází. V kontrastu se strukturovanými daty jsou však data nestrukturovaná, která potřebujeme alespoň částečně, nebo v lepším případě plně, analyzovat. Ta ale nejdou počítači tak jednoduše rozpoznávat. Jsou to např. fotky, hudba, videa či prostý neformátovaný text. Pro entitu nestrukturovaných dat musí v databázi existovat tzv. metadata, která umožňují jejich identifikaci. Avšak metadata nestačí k tomu, abychom byli schopni provádět rychlé a detailní vyhledávání a získat požadované informace v reálném čase.

Příkladem nestrukturovaných dat jsou například informace o pacientech, zahrnující diagnózy a zdravotní zprávy. Ty je potřeba kombinovat se strukturovanými daty jako je krevní tlak a další měření. Kdyby existoval systém, který by umožnil kombinovat tyto typy dat spolu s dalšími zdroji informací a vznikla by jedna velmi rychlá interaktivní databázová aplikace, lékaři by mohli snadněji a přesněji identifikovat příčiny zdravotních potíží pacienta. Navíc by měli možnost přistupovat k těmto datům v reálném čase, a to i ze svých mobilních zařízení kdekoli v nemocnici. Dalším případem využití nestrukturovaných dat jsou dokumenty z údržby letadel, kde je potřeba uchovávat informace o poruše či údržbě, a jejich popisu včetně přesného místa. I zde by takový rychlý flexibilní systém našel využití, neboť by mohl být použit k hledání vzorců

a korelací v procesu údržby, a tak pomohl mechanikům mnohem lépe předvídat budoucí poruchy, což by mohlo mít za následek vyhnutí se letecké nehodě, a tedy ušetření lidských životů (Plattner, 2013, s. 9-10).

2.1.1.3 Plánování produkce a ATP

Ve výrobních firmách je jedním ze stěžejních problémů k rozhodování správné plánování produkce a také tzv. Available to Promise (ATP) plánovací algoritmus. Plánování produkce vyžaduje znalost aktuální poptávky a jeho součástí jsou složité výpočty, velmi podobné těm, které jsou realizovány v moderních BI (Business Intelligence) řešeních. Plánovací algoritmus Available to Promise zase slouží k vyvažování počtu vyrobených položek vzhledem k aktuálním zásobám, cenám, logistickým procesům a poptávce. V případě zjištěné nerovnováhy musí výpočetní systém vše přepočítat a příslušně přeplánovat dodání zboží. Algoritmus ATP dnes mimo jiné používá předem vypočítané agregace. S použitím in-memory databáze by tyto připravené výsledky agregace nebyly potřeba a celý algoritmus by byl spuštěn v reálném čase na vyžádání. Podobný princip lze aplikovat obecně na proces plánování produkce (Plattner, 2013, s. 13).

2.1.1.4 Další požadavky

Jak bylo nastíněno v předchozích částech, existuje spousta oblastí, ve kterých vznikají nové požadavky na výpočetní a databázové systémy. V této kapitole bude krátce popsáno ještě pár dalších případů, konkrétně z oblastí sociálních sítí, cloud computingu a BI systémů.

Web, a zejména sociální sítě, přichází s novými požadavky na vysoce schopný databázový systém. Čím dál více jsou sociální sítě využívány firmami pro účely marketingu a budování značky. Tudíž je pro mnoho společností klíčové analyzovat data generovaná těmito stránkami. Získané informace a z nich vyplývající znalosti by mohly být použity ke zlepšení svých služeb, vytváření kampaní nebo k získání konkurenčních výhod. Co se týče webu obecně, zvláště vyhledávací enginy by v případě využití vysokorychlostní in-memory databáze mohly čerpat ze zadávaných dotazů a z nich pramenících zajímavých vzorců.

Cloudové služby vyžadují odpovídající datovou integrační strategii. To znamená, že například při přechodu mezi poskytovateli cloudových služeb je důležité mít zavedenou efektivní metodu transformování dat během migračního procesu. Problém nastává, když pro danou aplikaci neexistuje standard zajišťující kompatibilitu. V takovém případě může in-memory databáze poskytnout svůj koncept transparentního databázového pohledu, který popisuje způsob, jakým se mají vstupní hodnoty převést na výstupní. Zároveň je uvažováno přímé mapování jednotlivých atributů. Z povahy tohoto konceptu vychází, že může nahradit drahý ETL (Extract, Transform, Load) proces, který je standardně používán v dnešních BI systémech.

Na závěr můžeme zmínit přímo použití Business Intelligence řešení jako prostředku pro detailní pohled na veškeré obchodní aspekty, poskytovaný představitelům dané společnosti. Avšak dnes, když bereme v úvahu dostupnost mobilních aplikací, je přístup k těmto informacím vyžadován i z nižších úrovní organizace. Například pro obchodní zástupce je velmi užitečné mít možnost pracovat s těmito daty během jejich komunikace se zákazníky. Proto je pro ně důležité získat tyto informace během pár sekund, aby se zvýšila pravděpodobnost jejich úspěchu. Technologie in-memory databáze by mohla být řešením tohoto požadavku (Plattner, 2013, s. 11-12).

2.1.2 Nové schopnosti hardwaru

Veškeré požadavky a případy užití popsané v předchozí kapitole vyžadují velmi silný hardware. Bez dostatečného výkonu by uvedené scénáře nebyly proveditelné. V poslední dekádě došlo opět k výrazné evoluci hardwarového průmyslu. Výsledky této evoluce otevřely nové možnosti v oblasti softwaru a vývoje aplikací, což mimo jiné také vedlo k představení in-memory databázového systému (SAP HANA).

2.1.2.1 Procesor

V dřívějších dobách procesory v počítačích (CPU) pracovaly na stejné frekvenci jako počítačové operační paměti. Pracovní frekvence procesorů stabilně vzrůstala zhruba podle principu Moorova zákona, ale u paměťových čipů tomu tak nebylo. Proto mezi nimi postupem času vznikla velká výkonnostní mezera. Přístup do paměti byl drahý, zatímco procesorová jednotka během svých cyklů často pouze čekala na odpověď paměti. V průběhu minulého desetiletí se frekvence procesorů přestaly navyšovat, protože

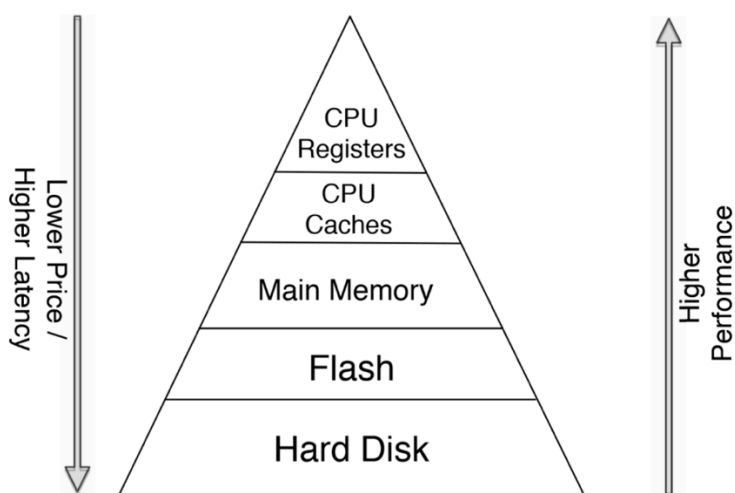
docházelo k přehřívání a bylo obtížné je efektivně chladit. Proto byly představeny vícejádrové procesory, které ve velkém měřítku přinášejí paralelismus. Pokud se podíváme na dnešní profesionální databázové systémy, běžně u nich najdeme 8 až 16jádrové procesory, přičemž typický uzel nebo základní deska těchto procesorů obsahuje 8. Větší databáze pak obsahují několik těchto uzlů. Problém nastává, když vezmeme v úvahu použití klasických pevných disků k ukládání dat v databázi. Výkon pevných disků se nemůže poměřovat s výkony procesorů, a proto se při výpočetních operacích většinu času čeká na ně. Z výše uvedeného vyplývá, že by nové databázové systémy měly v ideálním případě uchovávat primární kopii dat v operační paměti (Plattner, 2013, s. 19).

2.1.2.2 Paměť

Co se týče operační paměti, bylo by velmi drahé vyrábět paměťové čipy se stejným výkonem jako dnešní procesory za účelem zachování ideální rovnováhy mezi oběma prvky. Na druhou stranu jsou dnes paměti čím dál dostupnější a také jejich kapacita se dramaticky zvyšuje. V zásadě existují dva typy pamětí, mezi kterými můžeme volit. Jedná se o typ SRAM a DRAM. Zatímco SRAM buňky nepotřebují být periodicky obnovovány a jsou velmi rychlé, jsou však také mnohem komplikovanější, a tedy i dražší na výrobu než DRAM. Protože jsou DRAM buňky mnohem jednodušší a nezabírají tolik místa na desce, mohou DRAM paměti dosahovat podstatně vyšších kapacit. Z tohoto důvodu se právě tento typ používá v in-memory databázích jako hlavní úložný prostor.

Jedním z důležitých aspektů přístupu do paměti je lokalita dat. Ta se dělí na časovou a prostorovou, a obě hrají klíčovou roli v nejrychlejším možném přístupu k požadovaným datům. K dosažení co nejlepší lokality dat se používá mechanismus mezipaměti – cache. Vedle operační paměti, která tedy používá DRAM buňky, jsou zde malé vyrovnávací paměti cache, umístěné mezi hlavní pamětí a výpočetní jednotkou, a využívající rychlejší SRAM buňky. Jedná se tedy o jakési prostředníky pro dosažení co nejvyšší celkové rychlosti systému. Tyto cache paměti jsou rozděleny na L1 (umístěna přímo u každého procesorového jádra, komunikuje s registry), L2 a L3. Posledně jmenovaná je sdílena mezi všemi jádry v rámci jednoho procesoru. Nejmenší adresovatelná jednotka je tzv. cache line (řádek), a dále je tu zaveden systém překládání adres a předběžného načítání částí paměti za účelem dosažení efektivního přístupu.

Abychom výše zmíněné typy pamětí umístili do uceleného kontextu, můžeme celkovou hierarchii úložných médií popsat jako pyramidu, která je na Obrázek 1. Na spodu pyramidy je nejpomalejší typ paměti, pevný disk, který nabízí hodně úložného prostoru za nízkou cenu. Následuje paměť typu flash, která je rychlejší, ale používá v podstatě stejné softwarové principy založené na blokově orientovaných operacích. Hlavní paměť se nachází přímo uprostřed jak z hlediska výkonu, tak ceny. Na rozdíl od pevných disků a flash pamětí disponuje přímým přístupem, což je velkou výhodou. V poslední době je také ekonomicky dostupnější ve vyšších kapacitách. Na vrcholu pyramidy jsou pak vyrovnávací paměti cache (L3, L2 a L1), eliminující rychlostní rozdíl mezi CPU a hlavní pamětí, a špičku tvoří registry, které jsou nejrychlejší, ale také nejdražší. Tam jsou prováděny veškeré výpočty v počítači.



Obrázek 1 Hierarchie pamětí (Zdroj: Plattner, 2013, s. 24)

Důležitým faktem je, že přístup k datům na pevném disku je 100 000krát pomalejší než přístup do operační paměti. Tím můžeme snadno identifikovat, že tzv. úzkým hrdlem pro putování dat při databázových transakcích je pevný disk. Pokud by v databázovém systému byly pevné disky nahrazeny přímo operační pamětí, dosáhli bychom znatelného zrychlení při zpracování dat, i když vezmeme v úvahu vyrovnávací paměti, které tento rozdíl snižují. Při použití paměti DRAM je zde nevýhoda jejích volatilních buněk, tj. pro uchování dat musí být nepřetržitě připojeny ke zdroji napájení. Toto je však v in-memory databázích řešeno zálohováním dat na trvalá nevolatilní úložiště ve formě SSD nebo pevných disků, které v tomto konceptu slouží rovněž pro logování událostí a stavů a pro účely obnovy.

Pokud vezmeme v úvahu rozsáhlé databázové systémy velkých společností, paměťová struktura musí být implementována v odpovídajícím měřítku. V každém uzlu jsou použity SSD disky pro logování, archivování a zotavení. Jelikož je ale ve velkých databázových systémech mnoho takových uzlů, musí zde také existovat rychlá síťová infrastruktura, která je spojuje a umožňuje komunikaci mezi nimi. V současné době už existují síťové přepínače s propustností 100 Gb/s, které poskytnou dostatečný výkon pro splnění takových požadavků (Plattner, 2013, s. 19-24).

2.1.3 OLTP vs. OLAP

V této části práce budou popsány a srovnány dva základní typy databázových systémů, a to OLTP (On-line Transaction Processing) a OLAP (On-line Analytical Processing). Následně bude nastíněna myšlenka oba tyto systémy spojit v jeden jednotný systém, za současné eliminace ETL procesu.

2.1.3.1 Stávající koncept

Společnosti dnes obvykle používají dvě databáze. Na jednu stranu mají systém pro transakční operace, OLTP, který je uzpůsoben pro zpracování velkého množství krátkých online transakcí a nabízí prostředí pro víceuživatelský přístup tak, aby byla zachována integrita dat. Typickými případy užití OLTP jsou účetnictví, faktury, správa objednávek a prohlížení individuálních záznamů. Na druhou stranu firmy potřebují také systém, optimalizovaný pro analytické operace, OLAP. Pokud vezmeme v úvahu organizační strukturu a úroveň řízení firmy, je tento systém základem pro manažerský informační systém (MIS), systém pro podporu rozhodování (DSS), a dokonce pro exekutivní informační systém (EIS). Zásadně se tak liší od OLAP, který reprezentuje nižší úroveň transakčního zpracování. OLAP je běžně používán pro plánování, reporting, předpovědi a analýzu trendů (Kifer aj., 2005). Zatímco v systému OLTP převažují operace typu zápisu do databáze, provoz v OLAP databázi je jasně orientován na čtení. Z hlediska typů dotazů OLTP většinou provádí ty, které jsou více selektivní, tedy výsledkem je pouze pár záznamů, a OLAP zase využívá předem vypočítané agregace z velkého počtu záznamů a vybírá jen požadované atributy tabulek. Právě z důvodu těchto na prvních pohled protichůdných nároků existují dva zmíněné systémy, přičemž u každého je optimalizován výkon a flexibilita pro ten či onen daný typ provozu.

Novotný aj. (2005) též zmiňují základní rozdíly mezi OLTP a analytickými systémy. Architektura OLTP systémů je přizpůsobena jejich primárnímu účelu, čímž je pořizování dat. Tabulky jsou ukládány ve třetí normální formě, je zde hodně tabulek navzájem propojených a snaha o nulovou redundanci. OLTP systém je zatěžován kontinuálně, a to velkým počtem, klidně statisícem, transakcí za minutu. Tento provoz probíhá v reálném čase. Naproti tomu analytické systémy (OLAP) jsou optimalizovány pro podporu rozhodování. Jejich architektura je běžně rozdělena na více vrstev a je pro ně typický zmenšený počet denormalizovaných tabulek, zvýšení duplicity dat a vyšší frekvence indexů. Důležitou informací pak je, že na rozdíl od OLTP systémů aktualizují svá data periodicky, typicky v denních a měsíčních intervalech. Data v nich obsažená tak často nejsou aktuální.

Laberge (2012) uvádí, že ve světě OLAP systémů jsou běžně miliony a někdy dokonce i miliardy záznamů v tabulkách. Proto současné OLAP systému nemohou poskytnout zpracování dotazů v reálném čase, za krátkou dobu odezvy se považují minuty, ale často jsou to i hodiny.

Co se týče technického přístupu, oba systémy, OLTP a OLAP, jsou založeny na základním konceptu relačních databází, ale používají rozdílná schémata a metody pro uspořádání dat. OLTP ukládá n-tice v řádkově orientovaném schématu a řádky jsou organizovány v blocích, nacházejících se na pevných discích. Při přesunu dat z disků do operační paměti je pak využíván mechanismus vyrovnávacích pamětí, jak bylo zmíněno v kapitole 2.1.2.2. Z důvodu dosažení co nejvyššího výkonu jsou rovněž používány indexy, které umožňují velmi rychlé získání jednotlivých záznamů, avšak již nejsou tak efektivní při výběru většího počtu n-tic. Naproti tomu OLAP systémy často ukládají data do hvězdicových schémat a používají slovníky, které překládají názvy atributů tabulek na celá čísla za účelem rychlejších odpovědí na analytické dotazy. Nedávno se navíc v OLAP systémech začalo implementovat sloupcově orientované uspořádání dat, kde jsou záznamy uloženy podle atributů namísto jednotlivých řádků. V tomto schématu je použita nová technika komprese dat, tzv. slovníkové kódování, která kromě zvýšení výkonu analytických operací přináší znatelné snížení spotřeby paměti (Plattner, 2009).

Protože tu máme dva samostatné databázové systémy, data z OLTP systému, která zobrazují aktuální stav skutečnosti, musí být v pravidelných intervalech kopírována na

OLAP systém. Tento proces extrakce, transformace a nahrání dat do datového skladu se nazývá ETL (Extract, Transform, Load). Je poměrně drahý a časově náročný, a potřeba pravidelné replikace dat přináší další nevýhody, např. neaktuální data v OLAP a redundanci dat (Plattner, 2009). Další nevýhodou rozdělení OLTP a OLAP jsou předdefinované a předem vypočítávané agregace (materializované pohledy), které jsou nezbytné pro dostatečný výkon databáze, ale snižují flexibilitu a opět zvyšují redundanci systému (Plattner, 2013, s. 16).

Navzdory tomu, že jsou typy operací v OLTP a OLAP považovány za velmi rozdílné, se v nedávných výzkumech zjistilo, že četnost čtecích operací je dokonce o 10 % vyšší v OLTP. V operacích vkládání opět vede OLTP, ale OLAP taktéž často tento typ operace používá při aktualizování dat. Dalším zjištěným faktem bylo, že 88 % záznamů není ve velkých organizacích nikdy aktualizováno (Krueger aj., 2011). Jelikož aktualizace záznamu běžně spočívá ve smazání starého a vložení nového, zdá se tento proces jako zbytečný a mohlo by být efektivnější zavést přístup výhradně vkládacích operací, kde by staré záznamy byly pouze zneplatněny časovou značkou nebo příznakem validity, místo jejich smazání z databáze. Výše zmíněné nálezy vedou k myšlence spojit oba, OLTP a OLAP, systémy v jeden komplexní systém.

2.1.3.2 Kombinace OLTP a OLAP v jeden systém

Odstranit separaci OLTP a OLAP systémů a spojit je v jeden unifikovaný databázový systém byl nápad profesora Hasso Plattnera a jeho spolupracovníků. Cílem bylo dosáhnout velmi rychlého databázového systému s vysokým výkonem jak transakčních, tak analytických operací, s odezvou v řádu sekund. Jedním z prostředků použitých ke splnění tohoto cíle bylo uložení a běh celé databáze přímo v operační paměti. Avšak s využitím řádkově orientovaného uspořádání dat nedošlo k žádnému citelnému zrychlení, hlavně kvůli přítomnosti již existujícího a efektivního mechanismu vyrovnávacích pamětí cache. Tak přišel na řadu nápad použít sloupcově orientované uspořádání, které již bylo s úspěchem používáno v OLAP systémech, v celém unifikovaném databázovém systému běžícím v operační paměti (Plattner, 2009).

Velkým přínosem spojení obou systémů je odstranění ETL procesu, protože již není potřeba díky běhu transakčních i agregačních operací na jednom stroji, a tedy přítomnosti jediného zdroje dat. Navíc odpadá nutnost předem vypočítaných agregací

a materializovaných pohledů. Místo nich jsou využity schopnosti nového hardwaru, který byl popsán v kapitole 2.1.2. Agregace jsou spočítány za běhu v reálném čase a pohledy jsou poskytovány virtuálně. Výše zmíněnými vlastnostmi získáváme značné zjednodušení datových struktur a celého systému, radikální snížení redundance, zvýšení flexibility a vyhodnocování analytických dotazů v reálném čase (Plattner, 2013, s. 17).

2.2 In-memory databáze SAP HANA

Následující kapitola nastiňuje pohled na architekturu in-memory databáze SAP HANA. Její návrh a podoba vychází z kroků popsaných v předchozí kapitole 2.1. Je založena na SQL a využívá mnoha tradičních komponent, které jsou obvyklé i pro ostatní databáze, např. správce transakcí, tvůrce dotazů, spouštěcí engine a generátor běhového plánu. Jednou z vyčnívajících vlastností in-memory databáze je sloupcově orientované uspořádání dat. Navíc se v důsledku ukládání všech dat v operační paměti musí přizpůsobit způsob jejich organizace. Slovníkové kódování snižuje spotřebu paměti a zrychluje sekvenční procházení (skenování) atributů tabulek. Protože je kapacita operační paměti přece jenom omezená, využívají se další kompresní metody. Nakonec je použit tzv. partitioning logické databáze, který je navržen pro zavedení paralelismu na datové úrovni, a bude popsán v kapitole 2.2.5 (Plattner, 2013).

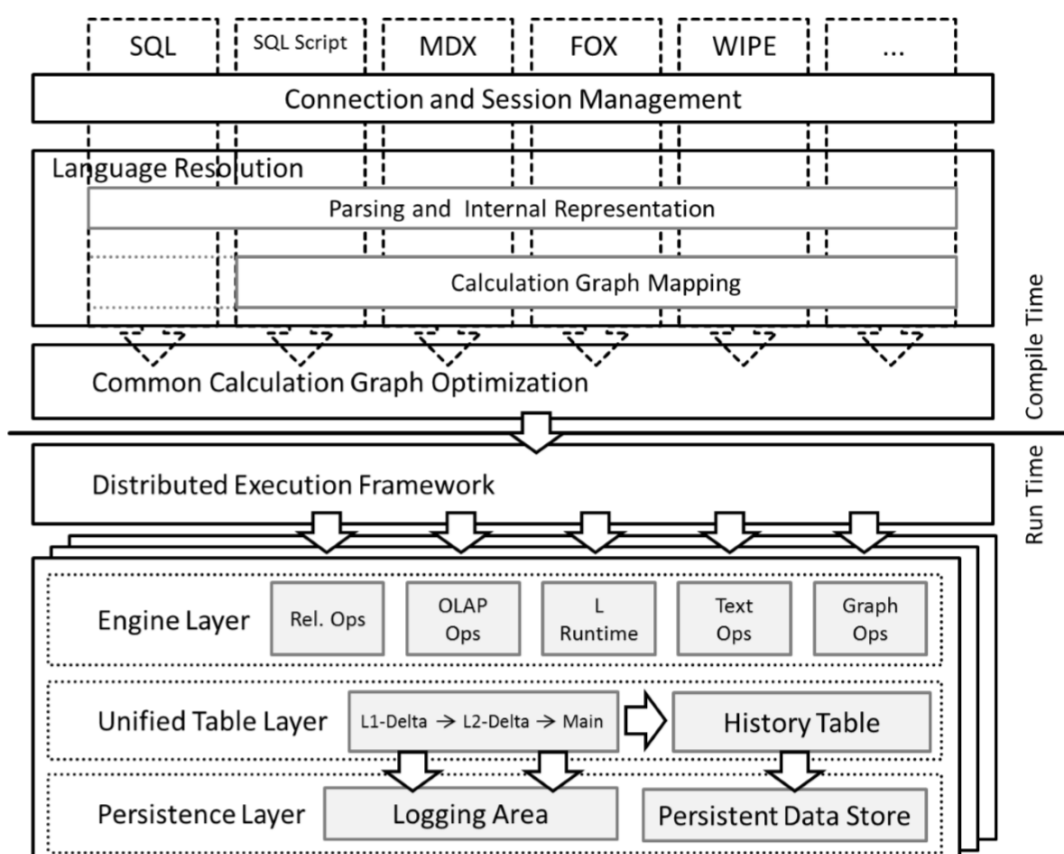
V článku od Färbera aj. (2012) jsou popsány hlavní vlastnosti a přínosy databáze SAP HANA. Ta nabízí komplexní prostředí s více enginy pro zpracování dotazů, reprezentaci aplikačně specifických business objektů, zužitkování síly současného hardwaru a efektivní komunikaci s aplikací.

2.2.1 Architektura

SAP HANA in-memory databáze umožňuje efektivní zpracování transakčních i analytických operací na jedné fyzické databázi za použití sloupcově orientovaného uspořádání dat, které musí být náležitě optimalizováno. Tato optimalizace spočívá ve správném řízení životního cyklu záznamu a množině efektivních kroků při přesunu záznamu z úložiště vhodného pro zápis (vkládání) do prostoru optimalizovaného pro čtení a komprimovaného úložiště sloužícího pro analytické dotazy.

Architektura databáze SAP HANA dovoluje platformě, řídící základní organizaci dat, měnit sémantiku aplikace podle potřeby, a to díky podpoře business objektů (např. OLAP kostky) a jejich logické reprezentace uvnitř databázového enginu. Toto přispívá ke snížení množství dat, která jsou přenášena mezi aplikací a databází. Navíc přináší prostředí s více enginy pro zpracování dotazů, což umožňuje podporu nejrůznějších typů dat, od jednoduchých textových dat po složitě strukturované (nastíněno v kapitole 2.1.1.2).

Z hlediska podoby architektury je in-memory databáze SAP HANA založena na vícevrstevném návrhu, zobrazeném na Obrázek 2. Je logicky rozdělena na fázi kompilační a běhovou pro zpracování databázových úloh. Navíc je navržena tak, aby aplikační a databázová vrstva byly co nejblíže sobě a mohlo být tedy dosaženo vyššího výkonu. Färber aj. (2012) zmiňují, že běžně používaný přístup přesunu všech dat z databázové do aplikační vrstvy a často zpět zabraňuje možnosti využití optimalizovaných datových struktur.



Obrázek 2 Vícevrstvá architektura databáze SAP HANA (Zdroj: Sikka aj., 2012)

Celkově je architektura doplněna komponentami, běžnými i v ostatních databázích, jako je správce metadat, správce autorizací a správce transakcí (Sikka aj., 2012). Správce transakcí garantuje splnění základních vlastností ACID (atomicita, konzistence, izolovanost, trvalost) a bezpečné souběžné řízení více verzí (MVCC) (Färber aj., 2012). Aby byl zajištěn požadavek na izolovanost transakcí, Lee aj. (2013) prezentují metodu izolace distribuovaných snímků momentálního stavu databáze a optimalizované protokoly dvoufázového potvrzení transakcí. Pro každou transakci je zde vytvořen unikátní token a ID transakce (TID), ale detailní popis tohoto mechanismu je již nad rámec této práce.

2.2.1.1 Fáze překladu

Jak můžeme vidět na Obrázek 2, horní část systému je tvořena vrstvou správce připojení a relace (Connection and Session Management), která přijímá dotazy v různých databázových jazycích, např. SQL, SQL Script, MDX nebo dalších doménově specifických jazycích. Aby byla zajištěna kompatibilita se všemi těmito jazyky, následuje vrstva zpracování jazyka (Language Resolution), která analyzuje vstupní řetězce a převádí je do interní reprezentace ve formě abstraktního syntaktického stromu. Nakonec je celý dotaz namapován do výpočetního grafu, kde dochází k samotnému logickému zpracování dotazu.

Dotazovací výraz (typicky ve formě příkazu *SELECT*) je ve výpočetním grafu reprezentován modelem tohoto grafu. Struktura tabulek a výsledky předchozích operací jsou umístěny ve zdrojových uzlech grafu, logické operátory jsou pak ve vnitřních uzlech. Kromě běžných SQL operátorů je model výpočetního grafu schopný pracovat i se speciálními business operátory, např. pro konverzi měn. Pro každý model je pak spuštěn optimalizační proces (Calculation Graph Optimization) a následně je logický plán provedení dotazu kompilován neboli přeložen, na fyzický plán, který je prováděn distribuovaným spouštěcím frameworkem (Distributed Execution Framework). Ten uzavírá fázi kompilace požadavku do databáze (Sikka aj., 2012).

2.2.1.2 Fáze běhu

Jak bylo popsáno v předchozí kapitole, distribuovaný spouštěcí framework má na starost provádění jednotlivých dotazů podle připraveného fyzického plánu. Spolupracuje se

strojovou vrstvou (Engine Layer), která poskytuje množiny databázových operátorů. Jsou to klasické relační operátory, OLAP operátory, které pracují s fakty a dimenzemi, speciální L-běhové operátory pro potřeby interního jazyka, textové operátory pro textovou analýzu a grafové operátory vhodné např. pro plánování zdrojů nebo analýzu sociálních sítí. Jelikož dochází k výměně dat mezi různými druhy operátorů, je potřeba implementovat specializované komunikační operátorové protokoly (Sikka aj., 2012). Další součástí strojové vrstvy je relační engine, který spravuje a kombinuje řádkově i sloupcově orientované schéma uložení dat, a také je schopen vzájemného převodu mezi těmito schématy (Färber aj., 2012).

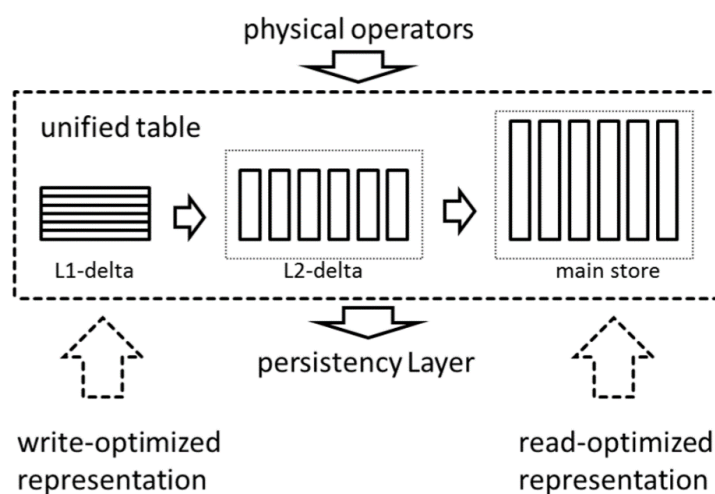
Vrstva unifikované tabulky (Unified Table Layer) poskytuje nástroj pro řízení životního cyklu záznamu, který je detailně popsán v následující kapitole 2.2.1.3. Je založena na kombinaci ukládacích prostorů s řádkově a sloupcově orientovaným uspořádáním dat, aby byly co nejvíce eliminovány rozdíly mezi transakčními a analytickými operacemi. Kromě toho také obsahuje historické tabulky pro dotazy na předchozí stavy, a relevantní metody přístupu pro různé typy operátorů.

Spodní část architektury se skládá z perzistentní vrstvy (Persistence Layer), kde se nachází prostor pro trvalé uložení dat. Ten slouží zejména pro logování a případnou obnovu dat. V případě poruchy či havárie systému jsou všechna data z operační paměti ztracena z důvodu jejích volatilních vlastností, a proto musí být poslední zaznamenaný stav databáze znovu nahrán do operační paměti z pevných disků či SSD. K tomuto účelu jsou potřeba časté snímky aktuálního stavu databáze, ale ty nevyžadují přílišnou režii a zdroje (Sikka aj., 2012). Färber aj. (2012) doplňují, že datové objekty jsou pravidelně zálohovány, a to během slučovacích (merge) a snímkových operací (snapshots).

2.2.1.3 Řízení životního cyklu záznamu

Každá jednotlivá datová entita, která vstoupí do databázového systému SAP HANA, prochází během svého životního cyklu několika fyzickými úložišti. Koncept řízení životního cyklu záznamu je zde nazýván také jako unifikovaná tabulka. Zatímco logicky je záznam upravován na jednom místě, ve skutečnosti je v různých fázích svého života fyzicky přesouván mezi různými úložišti. Cílem tohoto přístupu je poskytnout odpovídající výkon jak pro vysoce selektivní dotazy (OLTP), tak pro dotazy založené na agregacích a procházení atributů tabulek (OLAP).

Na Obrázek 3 můžeme vidět zmíněný koncept unifikované tabulky. Přístup k datům je realizován skrze vstupní fyzické operátory a výstup umožňuje přesouvat data do perzistentní vrstvy. Samotná unifikovaná tabulka obsahuje tři samostatné úložné prostory. Jsou tu úložiště L1-delta a L2-delta, také nazývané jako rozdílová vyrovnávací paměť (differential buffer) (Plattner, 2013, s. 163). L1-delta úložiště je optimalizováno pro operace zápisu a využívá řádkově orientované uspořádání dat k dosažení co nejlepšího výkonu pro obsluhu požadavků na vkládání dat. Nejlépe tak vyhovuje operacím vkládání, aktualizace a projekce celých záznamů (příkaz *SELECT **). Na druhou stranu zde není zavedena žádná komprese dat. L2-delta úložiště slouží jako prostředník mezi L1-delta a hlavním úložištěm. Na rozdíl od L1-delta již používá sloupcově orientované datové schéma a také slovníkové kódování sloužící ke kompresi dat, které je detailně popsáno v kapitole 2.2.3. Z výkonnostních důvodů jsou slovníky ponechány neseřazené, aby se snížila režie při vkládání a aktualizacích záznamů. Při prováděné dávkových vkládání je možné přeskočit L1-delta úložiště a vložit data přímo do prostoru L2-delta. Třetím úložným prostorem je hlavní úložiště tvořící jádro databáze. Opět využívá sloupcově orientované uspořádání dat, ale v tomto případě je komprese dat zajištěna na nejvyšší možné úrovni. Kromě slovníkového kódování se seřazenými slovníky jsou použity další komprimační metody jako např. prefixové kódování a run-length kódování pro co nejmenší spotřebu paměti vzhledem k ceně paměťových čipů. Tímto způsobem je hlavní úložiště optimalizováno pro časté čtení a analytické dotazy nad databází ve stylu OLAP.



Obrázek 3 Unifikovaná tabulka (Zdroj: Sikka aj., 2012)

Co se týče množství dat, úložiště L1-delta obvykle obsahuje 10 000 až 100 000 záznamů a L2-delta až 10 milionů záznamů. Hlavní úložiště je pak omezeno pouze dostupnou kapacitou operační paměti, přičemž v něm musí být uložen zbytek databáze.

Proces pohybu každého záznamu mezi zmiňovanými úložišti během svého životního cyklu je prováděn asynchronně tak, aby nezasahoval do běhu transakcí aktuálně spuštěných v databázi. Předávání dat mezi L1-delta, L2-delta a hlavním úložištěm je prováděno slučovacím procesem.

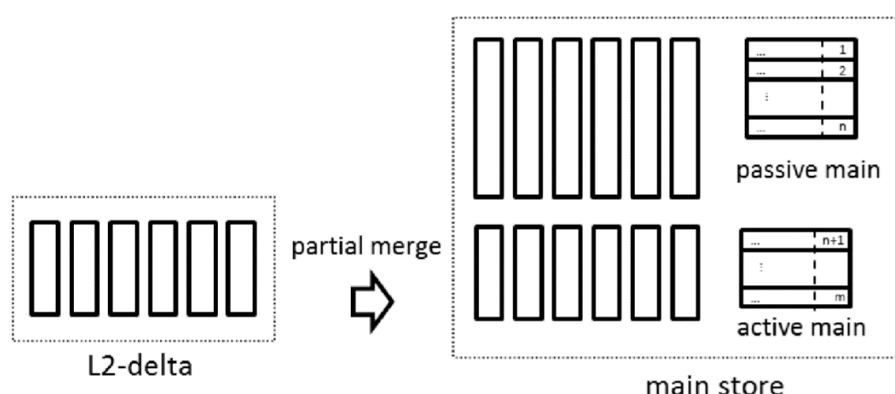
Slučovací proces z L1-delta do L2-delta spočívá v transformaci dat z řádkově orientovaného uspořádání na sloupcově orientované schéma uložení dat. Každý atribut řádku je uložen zvlášť do sloupců. Když je to nutné (přesouvání nově vložených dat), slovníky v L2-delta prostoru jsou aktualizovány. Oba tyto kroky mohou probíhat paralelně. Přesun dat z L1-delta do L2-delta končí odstraněním těchto dat z prvně jmenovaného úložiště. V tomto případě mluvíme o inkrementálním slučování.

Slučovací proces z L2-delta do hlavního úložiště je složitější. Je invazivnější z hlediska zásahu do právě probíhajících transakcí a také spotřebovává více zdrojů. Proto je zde ještě důležitější správná optimalizace a efektivní rozvržení úloh, aby se zabránilo jakýmkoli konfliktům. Na začátku procesu je úložiště L2-delta dočasně zablokováno pro všechny aktualizace a je vytvořen nový prostor L2-delta, který je vzat z posledního sloučení L1-delta a L2-delta a může obsloužit příchozí aktualizace. Poté je provedeno částečné nebo úplné sloučení. Díky novému prostoru L2-delta není v případě chyby ohrožen provoz databáze, a navíc je slučovací nezávislý na perzistentní vrstvě a úlohách logování a obnovy dat.

Ve slučovací procesu mezi L2-delta a hlavním úložištěm můžeme rozlišit mezi klasickým slučováním, slučováním s opětovným seřazením a částečným slučováním. Klasické slučování spojuje neseřazený slovník z L2-delta prostoru se seřazeným slovníkem z hlavního úložiště v jeden seřazený slovník, ale odstraňuje hodnoty, které pocházejí ze smazaných záznamů starého hlavního úložiště. To samo o sobě vyžaduje vytvoření mapování odstraněných hodnot (přesněji identifikátorů hodnot ve vektoru atributu, detailně popsáno v kapitole 2.2.3) na nové hodnoty nově vytvořeného slovníku. Slučování s opětovným seřazením doplňuje předchozí metodu o kompletní reorganizaci celé tabulky tak, aby hodnoty atributu, tedy jednoho sloupce, byly v paměti uloženy vedle

sebe, což zvyšuje míru komprese. Částečné slučování je technicky nejzajímavější. Snaží se snížit režii (výpočet nového slovníku a vektoru atributu) charakteristickou pro předchozí slučovací metody a využívá faktu, že v typických podnikových datech jsou hodnoty ve sloupcích brzy saturovány, tedy již jen zřídka kdy se použije zcela nová hodnota.

Koncept částečného slučování je vyobrazen na Obrázek 4. Hlavní úložiště je zde rozděleno na dvě části, pasivní prostor a aktivní prostor. Pasivní prostor obsahuje sloupce, u kterých je předpoklad, že již nebudou v budoucnosti aktualizovány. Jedná se tedy o stabilní část úložného prostoru. Na druhou stranu aktivní prostor obsahuje sloupce, u kterých ještě bude pravděpodobně často docházet ke změnám. Tato část se právě účastní procesu slučování s L2-delta úložištěm. Na začátku slučování, které probíhá v pravidelných intervalech, je aktivní prostor prázdný a dochází k přenesení dat z L2-delta sem. Pasivní prostor není do tohoto procesu zahrnut. Na rozdíl od klasického slučování se uchovávají dvě lokální kopie slovníkových struktur, jedna v pasivním úložišti a druhá, rozšířená o zcela nové hodnoty, pak v aktivním (Sikka, 2012).



Obrázek 4 Částečné slučování (Zdroj: Sikka aj., 2012)

Při představě distribuce dat mezi L1-delta, L2-delta a hlavním prostorem, a jeho aktivní a pasivní částí, může vyvstat otázka, jak je řešeno zpracování dotazů, které vyžadují data z několika těchto míst zároveň. Té ještě nahrává skutečnost, že agregace jsou prováděny na vyžádání, a tedy intenzita analytických dotazů je ještě vyšší. V in-memory databázích je tento problém řešen mechanismem podobným vyrovnávacím pamětím, který kombinuje uložené výsledky předchozích dotazů v hlavním úložišti s nově přidanými

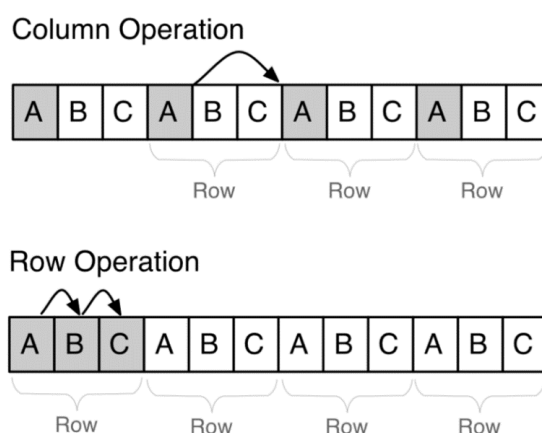
záznamy z L1-delta a L2-delta úložišť. Existuje několik strategií tohoto mechanismu, přičemž každá se zaměřuje na různé databázové operace jako např. spojení, aktualizace nebo odstranění. Údržba těchto postupů probíhá zároveň s procesy slučování (Müller aj., 2015).

2.2.2 Uspořádání dat

Ačkoliv jsou z pohledu konceptu databázové tabulky dvojrozměrné, jejich obsah musí být v paměti uložen lineárně v souladu s adresním uspořádáním. I tak má volba způsobu organizace dat v databázích velký vliv na výkon různých databázových operací. Jelikož je tato problematika jedním ze stěžejních témat této práce, bude detailněji popsána v této kapitole.

2.2.2.1 Řádkové uspořádání dat

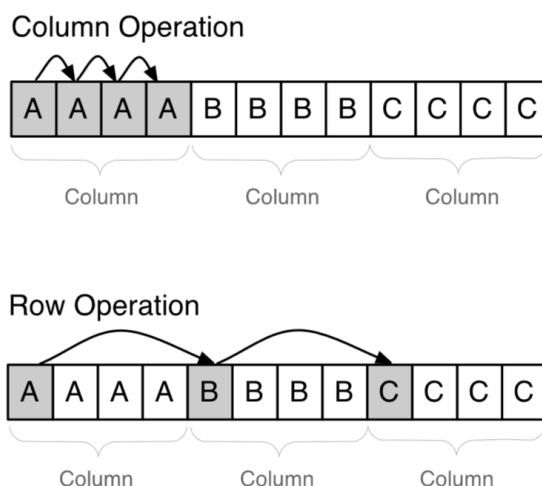
Řádkově orientované uspořádání dat je nejběžnějším způsobem organizace dat a je dlouhou dobu používáno ve všech typech databází. Všechny n-tice (řádky) jsou v paměti uloženy jedna za druhou. To je výhodné pro rekonstrukci celých n-tic, tedy vrácení všech atributů záznamu, ale nevhodné pro sekvenční procházení atributů a agregace (Plattner, 2013, s. 29). Řádkové uspořádání dat v paměti spolu s přístupem k řádkům a sloupcům lze vidět na Obrázek 5. Při procházení sloupců (column operation), tedy atributů, a agregačních operacích je vždy potřeba přeskočit zbytek atributů daného záznamu. Naopak při rekonstrukci celých n-tic záznamů jsou všechny atributy daného záznamu hned za sebou.



Obrázek 5 Řádkové uspořádání dat v paměti (Zdroj: Plattner, 2013, s. 60)

2.2.2.2 Sloupcové uspořádání dat

Sloupcově orientované uspořádání dat je již s oblibou používáno v některých OLAP systémech, jak bylo zmíněno v kapitole 2.1.3.1. V tomto případě jsou sloupce tabulky uloženy za sebou v sousedních paměťových blocích. Logicky by to mělo znamenat výhodnější podmínky pro sekvenční čtení hodnot z jednoho sloupce, konkrétně rychlé sekvenční procházení atributů umožňující počítání agregace na vyžádání v reálném čase. To znamená žádné předem vypočítané agregace, méně redundance a složitosti systému. Analogicky k řádkovému uspořádání, Obrázek 6 ukazuje postup při procházení sloupců a rekonstrukci celých n-tic záznamů. V prvním případě jsou hodnoty sloupce hned za sebou a výpočet tak může být velmi rychlý. Během přístupu k celým řádkům jsou však atributy záznamů od sebe vzdáleny a dochází tak ke zpomalení.



Obrázek 6 Sloupcové uspořádání dat v paměti (Zdroj: Plattner, 2013, s. 60)

Ukládání dat podle sloupců má několik důležitých důsledků. Kromě zrychlení, které se projevuje v sekvenčním procházení sloupců a výpočtu agregací, má přínos také během zpracování transakčních operací v případě výběru pouze několika atributů záznamu (tedy nepoužívání příkazu *SELECT **). Jak bude zmíněno v kapitole 2.2.4.2, v typických podnikových databázích je pravidelně čtena pouze menší část atributů záznamu, tudíž je tento přínos opravdu relevantní. Jedinou podmínkou je, aby se programátoři obecně vyhýbali automatickému výběru všech atributů (příkaz *SELECT **), protože rekonstrukce celých záznamů by ve sloupcově orientovaném schématu mohla být velmi drahá. Navíc odpadá potřeba mít zaveden mechanismus indexů, protože každý atribut je v tomto

uspořádání automaticky indexem sám sobě. Dále toto schéma přináší techniku slovníkového kódování (kapitola 2.2.3) a dalších metod komprese (kapitola 2.2.4). Některé z těchto metod lze využít právě jen ve sloupcovém uspořádání a mohou tak zužít dobrou lokalitu dat. Kromě nižší spotřeby paměti je komprese také výhodná v tom, že uchovává co největší množství dat v operační paměti a snižuje tak objem dat, který je nutno přenášet mezi hlavní paměti a nevolatilním úložištěm. Obecně je pro podniková data často výhodnější použít sloupcově orientované uspořádání (Plattner, 2013, s. 61). Krueger aj. (2011) dále prezentují výsledky své analýzy, která zkoumala pracovní zátěž v podnikové databázi. Bylo zjištěno, že jasně převažuje zpracování jednotlivých atributů a celkově čtecích operací, což je další fakt podporující využití sloupcového schématu.

Sloupcové uspořádání je však komplikovanější při operacích vkládání záznamů do databáze nebo jejich aktualizaci, protože n-tice musí být uloženy jedna za druhou. Z tohoto důvodu je v in-memory databázi zavedeno ještě jedno úložiště, tzv. rozdílová vyrovnávací paměť (differential buffer), které je optimalizováno pro častý zápis záznamů. Veškeré změny jsou pak v pravidelných intervalech převáděny do hlavního prostoru, jehož mechanismus byl popsán v kapitole 2.2.1.3.

Abadi aj. (2008) detailně zkoumali rozdíly mezi sloupcově a řádkově orientovaným úložištěm. Vycházeli z vědomosti, že sloupcově orientovaný překonává řádkový zejména v analytickém provozu (datové sklady, BI, podpora rozhodování) a snažili se zjistit, zda je možné emulovat výhodné vlastnosti sloupcového uspořádání v tom řádkovém. Taková emulace by byla teoreticky dosažitelná pomocí metody vertikálního partitioningu (bude popsán v kapitole 2.2.5.1), případně indexováním každého sloupce pro získání samostatného přístupu k atributům řádku. Z tohoto předpokladu autoři výzkumu vyšli a zkusili upravit fyzický návrh tradičního řádkového uspořádání dat, aby fungovalo jako sloupcové v situacích pro toto uspořádání výhodných. Vertikální partitioning umožnil rozdělení tabulek na sady menší tabulek o dvou sloupcích. Prvním atributem byl vždy primární klíč záznamu a druhým hodnota atributu. Indexování sloupců zase pomohlo k velmi rychlému přístupu k atributům požadovaných analytickými dotazy. Nakonec autoři vyzkoušeli i materializované pohledy obsahující jednotlivé sloupce, které byly předmětem dotazů. Závěr výzkumu však říká, že i když bylo v řádkovém schématu úspěšně dosaženo některých předností toho sloupcového, ani použitím zmíněných

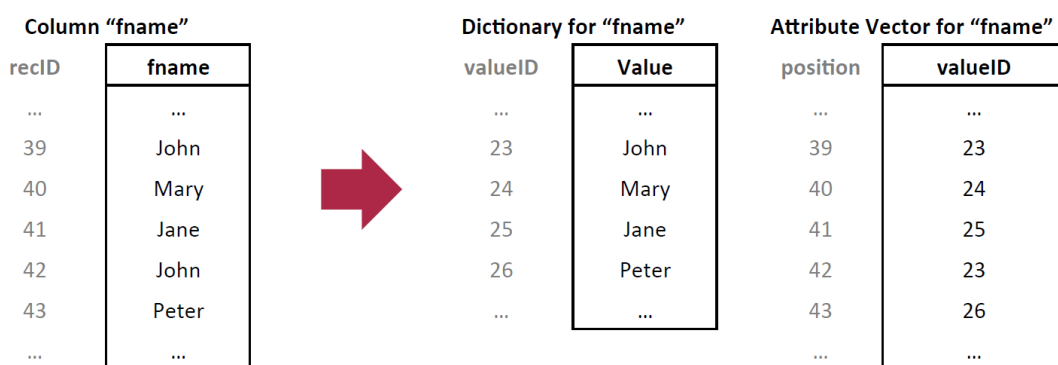
heuristik se nepodařilo emulovat všechny výhody bez radikálního zásahu do vrstvy úložiště, a především do vrstvy, která zpracovává a vykonává dotazy nad databází. Jsou zde zmíněny také pravděpodobné příčiny neúspěšné emulace, např. požadavky na zdroje pro rekonstrukci záznamů a dílčí režie vyžadována každou n-ticí. Sloupcově orientované schéma navíc nabízí další heuristiky jako je pozdní materializace (kapitola 2.2.6.4), vícenásobné předávání hodnot atributů mezi operátory, anebo další kompresní metody (kapitola 2.2.4).

Plattner (2014) ve svém novějším článku zkoumá dopad používání sloupcového uspořádání dat v in-memory databázi na podnikových systémech. Z větší části se zaměřuje na odstranění předem vypočítaných agregací, který jsou velmi náročné na zdroje a musí být neustále udržovány. Pro připomenutí, agregace hrají důležitou roli v procesech typických pro OLAP systémy, tj. reportingu, analýze a plánování. V dnešní době jsou ale požadavky na provádění reportingu v reálném čase, a to s aktuálními transakčními daty. Sloupcově orientované in-memory databáze, které kombinují OLTP a OLAP systémy do jednoho, se mohou zbavit údržby předem vypočítaných agregací a spočítat je za běhu až na vyžádání od uživatele. Tato schopnost přináší své výhody z pohledu vyšší flexibility, nižší složitosti, nižší spotřeby operační paměti a snížení redundance. Také má za následek nižší nároky na vkládání dat (žádná údržba agregací, jednodušší izolace transakcí při zachycování aktuálního stavu databáze). Počítání agregací za běhu však vyžaduje efektivní mechanismus vyrovnávací paměti. Pokud jde o možné pomalé vkládání záznamů do sloupcově orientované paměti, výsledky výzkumu ukazují, že odstranění nutnosti aktualizace agregací má mnohem větší dopad než zmíněná nevýhoda při vkládání do sloupců. Další analýza od Krueger aj. (2011) ukazuje, že poměr mezi četností čtecích a zápisových operací v podnikových systémech neustále narůstá ve prospěch prvně zmíněných. Operace čtení z databáze tedy dominují čím dál více. Příčiny tohoto trendu jsou následující: méně vkládání a aktualizací díky absenci údržby agregací, častější přímý přístup k transakčním datům (vynechání materializovaných pohledů a připravených agregací) a intenzivnější používání interaktivních aplikací, které se stávají dostupnými díky rychlým výpočtům za běhu.

2.2.3 Slovníkové kódování

Poté, co jsou pevné disky nahrazeny operační pamětí, se právě ona stává novým úzkým hrdlem systému. Je proto žádoucí minimalizovat množství dat v ní uložených, a tedy zároveň snížit objem dat, který je potřeba z paměti přenést. S cílem snížit množství obsazené paměti a zrychlit přístupu k datům je důležité implementovat speciální typ organizace dat. V databázi SAP HANA k tomuto účelu slouží tzv. slovníkové kódování (Plattner, 2013, s. 37). To je také základem pro další metody komprese data, které jsou popsány v příští kapitole 2.2.4.

Slovníkové kódování zužtkovává výhodu typickou pro velké podnikové databáze, kterou je nízká kardinalita hodnot. Nejdůležitějším přínosem je ale reprezentace složitých a dlouhých hodnot (např. dlouhé řetězce) pomocí obyčejných celých čísel. Můžeme předpokládat, že ve sloupcově orientované databázi obsahující pouze celá čísla bude dosaženo vynikajících rychlostí při zpracování agregačních operací a sekvenčním procházení sloupců.



Obrázek 7 Slovníkové kódování (Zdroj: Plattner, 2013, s. 38)

Každý sloupec je ve slovníkovém kódování zakódován zvlášť. Princip mechanismu slovníkového kódování můžeme vidět na Obrázek 7, kde je uveden příklad na tabulce světové populace, která obsahuje všechny lidi na světě, zhruba tedy 8 miliard záznamů). Sloupec *fname* (křestní jméno) v této tabulce je rozdělen na slovník a vektor atributu (v jiných zdrojích také nazýván jako index hodnot). Slovník každé unikátní hodnotě v našem sloupci přiřazuje celé číslo. Tato celočíselná hodnota se nazývá identifikátor hodnoty (*valueID*) nebo jednoduše ID. Například jméno *Jane* má ID 25. Vektor atributu

tak nahrazuje dlouhé textové hodnoty krátkými celočíselnými hodnotami podle odpovídající pozice ve sloupci. Míra komprese slovníkového kódování je závislá na míře opakování jednotlivých hodnot v daném sloupci. V našem případě jsou v tabulce dva výskyty jména *John* (pozice 39 a 42). Slovník však tuto hodnotu ukládá pouze jednou, a ne dvakrát jako v případě bez kódování. Protože podle odhadů na světě existuje 5 milionů unikátních, potřebujeme 23 bitů pro ID každé hodnoty ($2^{22} < 5 \text{ milionů} < 2^{23}$). Dále pokud uvažujeme velikost každé textové hodnoty 49 bajtů a počet záznamů v tabulce 8 miliard, dostáváme následující rovnici pro výpočet dosahované míry komprese:

$$\frac{\text{nekomprimovaná data}}{\text{komprimovaná data}} = \frac{8 \text{ miliard} \times 49 \text{ bajtů}}{(8 \text{ miliard} \times 23 \text{ bitů}) + (5 \text{ milionů} \times 49 \text{ bajtů})} \approx 17$$

Míra komprese je tedy v našem případě 17. To znamená, že sloupec zakódovaný slovníkovým kódováním zabírá pouze 6 % původního prostoru. Můžeme jednoduše vyvodit, že čím více je v tabulce opakujeících se hodnot, tím větší míry komprese bude dosaženo.

Jak bylo zmíněno v kapitole 2.2.1.3 o řízení životního cyklu záznamu, a L1-delta, L2-delta a hlavním úložišti, slovník může být seřazen. Se seřazeným slovníkem je možné nalézt identifikátor hodnoty s logaritmickou časovou složitostí, kdežto u neseřazených slovníků je to s lineární časovou složitostí. Je tu tedy velký rozdíl v rychlosti nalezení požadovaného ID. Na druhou stranu je uchovávání slovníku seřazeného poměrně drahé, protože řazení musí být provedeno pokaždé, když je vkládána zcela nová hodnota. Navíc se kromě seřazení slovníku musí též aktualizovat pozice ve vektoru atributu (více v kapitole 2.2.6.2).

Protože je procesor (CPU) mnohem vhodnější pro zpracování čísel než znaků, implementace celočíselného vektoru atributu znamená výrazné zrychlení všech operací, které pracují s uloženými daty. I když mohou vyvstat obavy, že režie způsobená dodatečnými datovými strukturami a z toho plynoucí vyšší zátěž procesoru při organizaci těchto struktur nebude odpovídajícím způsobem vyvážena výhodami, bylo zjištěno, že vysoká míra komprese tuto režii snadno kompenzuje. Navíc tím, že úzkým hrdlem systému je v in-memory databázi operační paměť, ne procesor, jeho vyšší zatížení vlastně není nijak na škodu (Plattner, 2013, s. 37-41).

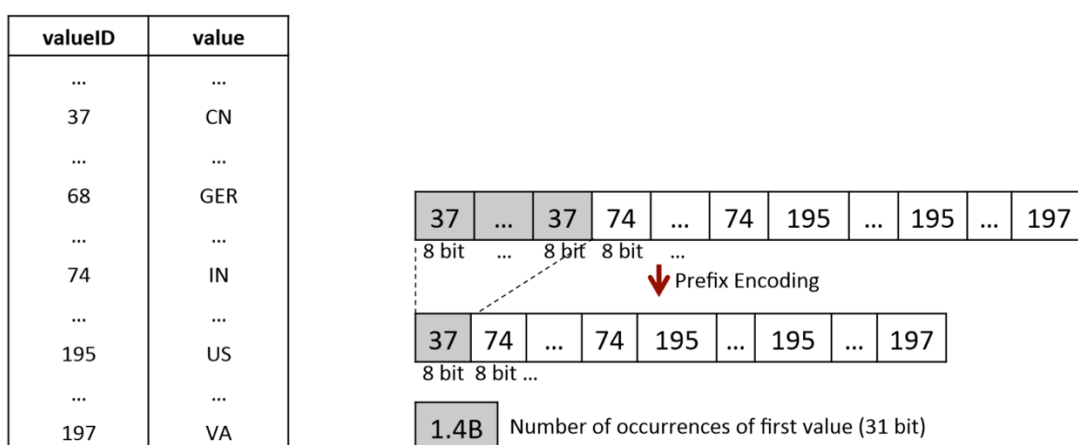
2.2.4 Další metody komprese dat

Protože je prostor v operační paměti kvůli její ceně vždy značně limitován, je důležité dosáhnout co možná největší míry komprese uložených dat. Nad rámec metody slovníkového kódování můžeme dále snížit spotřebu paměti použitím dodatečných kompresních technik, které jsou vhodné pro sloupcově orientované uspořádání dat. Metody komprese dat popsané v této části jsou poměrně jednoduché a nabízejí rozumný poměr mezi mírou komprese a nároky na výkon procesoru (Plattner, 2013, s. 43).

Potenciální efektivita další komprese je podpořena skutečností, že v podnikových databázích, které obsahují velké množství atributů, je mnoho z těchto atributů nevyužívaných, zhruba 55 %. Navíc má mnoho sloupců nízkou entropii, obsahující hodně prázdných hodnot (hodnot NULL) a hodnoty v nich mají nízkou kardinalitu. Toto všechno přispívá k větší efektivitě kompresních metod (Plattner, 2013, s. 17).

2.2.4.1 Prefixové kódování

Prefixové kódování je vhodné pro situace, kdy ve sloupci existuje jedna dominantní hodnota, která svou četností zřetelně převyšuje výskyt ostatních hodnot. Přímě se tak nabízí tuto dominantní hodnotu komprimovat a neukládat ji zbytečně tolikrát, kolikrát se vyskytuje v daném sloupci. Podmínkou pro aplikaci prefixového kódování je seřazený vektor atributu podle identifikátoru hodnot a umístění dominantní hodnoty na začátku tohoto vektoru.



Obrázek 8 Prefixové kódování (Zdroj: Plattner, 2013, s. 44)

Příklad prefixového kódování je vidět na Obrázek 8, přičemž opět je použita tabulka celé světové populace. Sloupec (atribut) reprezentující zemi původu člověka obsahuje jednu dominantní hodnotu *CN* (Čína, nejlidnatější země), která má ID 37. V nekomprimovaném vektoru atributu je tento identifikátor uložen zhruba 1,4 miliardkrát (populace Číny). Ve vektoru atributu, na který je aplikováno prefixové kódování, je identifikátor Číny uloženo pouze jednou a počet výskytů této dominantní hodnoty je uchován hned vedle. V tomto případě díky komprimaci ušetříme 17 % prostoru v paměti (Plattner, 2013, s. 43-45).

2.2.4.2 Ostatní kompresní metody

Mezi další kompresní metody patří tzv. run-length kódování, které je nejúčinnější pro sloupce obsahující pouze několik unikátních hodnot, které mají vysoký počet výskytů. Identifikátor každé hodnoty je uložen ve vektoru atributu pouze jednou a je vytvořen dodatečný vektor obsahující první pozice (offset) všech ID za sebou.

Clusterové kódování rozděluje vektor atributu do bloků konstantní velikosti, nejčastěji o 1024 hodnotách. Pokud je celý blok zaplněn jednou hodnotou, je následně zkomprimován do jednoho elementu. Jeho velikost tak není 1024, ale pouze 1. Aby systém věděl, které bloky byly zkomprimovány, je vytvořen přídatný vektor s příznaky komprimace jednotlivých bloků.

Nepřímé kódování je docela podobné právě zmíněnému clusterovému. Kromě manipulace s bloky vytváří dodatečné lokální slovníky pro každý blok, který obsahuje pouze pár unikátních hodnot. Lokální slovník se skládá jen z hodnot přítomných v daném bloku. Díky tomu identifikátory (ID) hodnot nedosahují příliš vysokých čísel, což znamená ušetření bitů potřebných pro reprezentaci identifikátorů v lokálním slovníku. Přístup k hodnotám se tak stává nepřímým, i když možnost přímého přístupu zůstává zachována.

Jako poslední zmíníme tzv. delta kódování, které se oproti předešlým metodám zaměřuje na slovníkovou kompresi. Pokud je slovník seřazen abecedně, sousední hodnoty často sdílejí stejný prefix. Delta kódování opět vytváří bloky o stejné velikosti. Takový komprimovaný slovníkový blok obsahuje počet znaků první hodnoty, hodnotu samotnou, a poté pro každou další hodnotu je zde počet sdílených znaků s první hodnotou (prefix), počet zbývajících znaků a samotné zbývající znaky (Plattner, 2013, s. 43-51).

Abadi aj. (2008) shrnuli vliv komprese na sloupcově orientované uspořádání dat. Zjistili, že průměrně snižuje spotřebu paměti o 50 %, ale v případě seřazených slovníků je úspora řádově vyšší.

2.2.5 Partitioning

Partitioning databází spočívá v jejich rozdělení na několik objektů, které jsou pak provozovány nezávisle. Tato nezávislost je velmi užitečná pro dosažení vysoké míry paralelismu při použití vícejádrových procesorů. I když je databáze z fyzického hlediska rozdělená, z pohledu uživatele musí zůstat transparentní, rozdělení tedy nesmí mít žádný vliv na používání. Existují dva typy partitioningu, vertikální a horizontální (Plattner, 2013, s. 63).

2.2.5.1 Vertikální partitioning

Vertikální partitioning dělí databázi do objektů podle sloupců. Vytváří skupiny atributů, kde v každé skupině je kopie primárního klíče. Pokud je rozdělení provedeno tak, aby ve skupinách byly atributy, které budou pravděpodobně v databázových operacích zpracovávány spolu, což může zvýšit výkon operací spojení (join). Vertikální partitioning může být také využit v řádkově orientovaném uspořádání dat za účelem emulace sloupcového schématu, jak již bylo zmíněno v kapitole 2.2.2.2.

2.2.5.2 Horizontální partitioning

V tradičních řádkově orientovaných databázích je běžným přístupem horizontální partitioning. Na rozdíl od vertikálního databázi rozděluje do navzájem nezávislých skupin n-tic podle řádků. Rozdělení může být provedeno podle několika kritérií. Těmi jsou např. rozsah hodnot, vzájemné přiřazení hodnot, nebo jejich sémantika, avšak popis těchto kritérií je již mimo rozsah této práce (Plattner, 2013, s. 63-65).

2.2.6 Operace v in-memory databázi

To, že in-memory databáze SAP HANA používá sloupcově orientované uspořádání dat a slovníkové kódování, vyžaduje určité změny v implementaci různých databázových operací a funkcí. Například jsou ovlivněny operace odstranění (*DELETE*), vkládání

(*INSERT*) nebo čtení (*SELECT*). Aby in-memory databáze poskytla stejnou podobu výstupu, na jakou jsou uživatelé zvyklí z tradiční řádkově orientované databáze, je potřeba zavést několik druhů strategií materializace. Jelikož je tato práce včetně praktické části hodně zaměřena na analytické operace a agregační funkce s nimi spojené, budou v následujících podkapitolách uvedena specifika jejich implementace.

2.2.6.1 Odstranění

Operace odstranění principiálně zneplatňuje daný záznam v databázi. Existují dva druhy odstranění záznamu, fyzické a logické. Prvně jmenované fyzicky vymaže n-tici z databáze, a ta je tudíž navždy odstraněna. Na druhou stranu logické odstranění pouze ruší platnost odpovídajícího záznamu. K tomu se používá např. příznak validity nebo časová známka. Záznam tak zůstává v databázi stále dostupný pro dotazy směřující do minulých stavů, což je důležitý nástroj pro analýzu historie (Plattner, 2009).

Nyní se podrobněji podíváme na metodu fyzického odstranění záznamu, protože je zajímavá hlavně z pohledu slovníkového kódování sloupců v databázi SAP HANA. Když dochází k odstranění hodnoty ze sloupce, nejprve musí být ve slovníku určeném pro tento sloupec nalezeno její ID. Poté se prochází vektor atributu a hledají se v něm všechny výskyty tohoto identifikátoru. Nalezené výskyty jsou následně smazány z vektoru atributu, což také znamená, že následující hodnoty musí být posunuty, aby došlo k odstranění vzniklých mezer ve vektoru. V samotném slovníku však hodnota zůstává pro možné budoucí využití. Protože je celá operace odstranění velmi drahá, může být rozhodnuto, že se budou záznamy pouze do databáze vkládat – fyzické odstraňování je nahrazeno logickým (Plattner, 2013, s. 71-72).

2.2.6.2 Vkládání

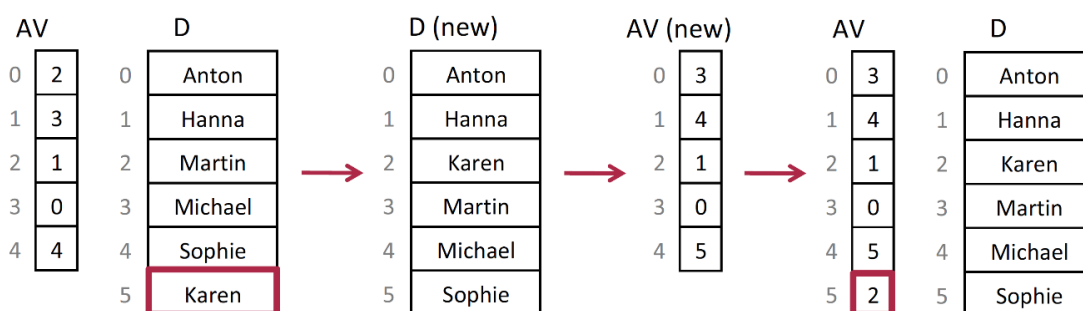
V případě řádkově orientované databáze jsou nové záznamy jednoduše vloženy za poslední n-tici tabulky. Situace se ale stává komplikovanější při přítomnosti sloupcově orientovaného a slovníkově kódovaného úložiště. Existují tři různé scénáře.

Prvním a nejjednodušším případem je, pokud vkládaná hodnota je již obsažena ve slovníku. Následně se pouze přečte její identifikátor ze slovníku a vloží se na konec vektoru atributu, čímž je vytvořena nová pozice.

Druhou možností je, když se nová hodnota ve slovníku ještě nenachází. Musí být tedy do něj přidána. Protože je slovník z důvodu rychlejšího hledání seřazen, budeme v tomto případě předpokládat, že nová hodnota je abecedně nejvyšší, a tudíž může být uložena na konec slovníku. Zbytek operace je pak analogický jako v první scénáři.

Třetí a nejkomplicovanější situace nastává, pokud musí být slovník po vložení nové hodnoty znovu seřazen. Tím pádem se bude muset přepočítat ID pro mnoho hodnot a následně se musí odpovídajícím způsobem upravit i vektor atributu. Do něj se nakonec vloží i identifikátor nové hodnoty. Je zřejmé, že tento proces je velmi nákladný.

Posledně zmíněný scénář je ilustrován na Obrázek 9. Nová hodnota *Karen* je vložena do slovníku. Jelikož je tím ve slovníku porušeno abecední pořadí, musí být slovník znovu seřazen. Výsledkem ale je, že některé hodnoty získají nové ID. Například hodnotě *Michael* se změní ID ze 3 na 4. Tyto změny pak musí reflektovat i vektor atributu. Za jeho poslední položku se na závěr vloží identifikátor nové hodnoty *Karen*.



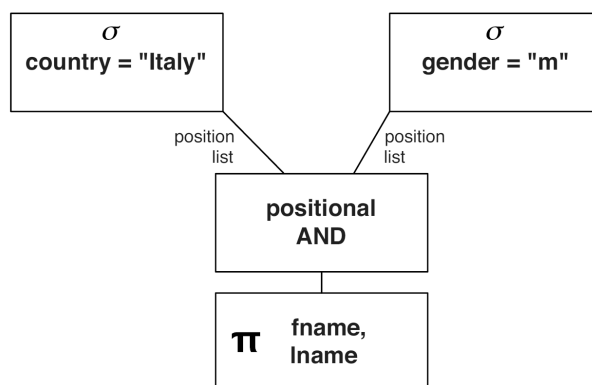
Obrázek 9 Operace vkládání (Zdroj: Plattner, 2013, s. 78-79)

Můžeme vidět, že vkládání nových hodnot je docela drahá operace. Pro zmírnění této negativní vlastnosti je v in-memory databázi implementována rozdílová vyrovnávací paměť, která již byla zmíněna v kapitole 2.2.1.3. Obecně dopad tohoto problému záleží zejména na kardinalitě sloupce, kde vyšší kardinalita způsobuje častější nutnost reorganizace slovníku a vektoru atributu (Plattner, 2013, s. 75-80).

2.2.6.3 Čtení

Databázový příkaz *SELECT* je nezbytným nástrojem pro čtení dat z databáze a je používán velmi často. Je na něm de facto postavena také praktická část této práce. Součástí tohoto příkazu je selekce (značena σ), projekce (značena π), dále relační operátory a kartézský součin.

Pro každý zadaný dotaz se vždy generuje tzv. exekuční plán, který obsahuje přesné kroky k vyzvednutí požadovaných dat z databáze. Může existovat několik exekučních plánů dávajících stejný výsledek, ale pomocí optimalizačních technik je vybrán ten nejefektivnější. Hlavním cílem v každém kroku je minimalizace dílčího objemu dat. To znamená provedení nejvíce omezujících selekcí a spojení nejmenších tabulek co nejdříve.



Obrázek 10 Exekuční plán operace čtení (Zdroj: Plattner, 2013, s. 101)

Příklad exekučního plánu zpracování dotazu při operaci čtení je na Obrázek 10. Opět uvažujeme tabulku celé světové populace a chceme získat křestní jména a příjmení všech mužů z Itálie. Na zobrazeném plánu můžeme vidět, že selekce obou atributů může být provedena paralelně. Nejprve se hledá ID hodnoty *Italy* ve sloupci země a ID hodnoty *m* ve sloupci pohlaví. Dále se prochází vektor atributu obou sloupců a pro každý z nich se vygeneruje seznam pozic, na kterých jsou zmíněné identifikátory. Následně je provedena logická operace s poziční operátorem *AND* a oběma seznamy jako operandy. Tím získáme jeden seznam pozic záznamů, které vyhovují oběma podmínkám selekce. Na závěr se s pomocí obdrženého seznamu pozic získávají n-tice, v tomto případě obsahující pouze atributy křestního jména a příjmení, a jsou ve formě řádků předávány na výstup (Plattner, 2013, s. 99-101).

2.2.6.4 Materializace

Proces materializace ve sloupcově orientovaných databázích slouží k poskytnutí výstupu v takové formě, na kterou jsou uživatelé zvyklí. To znamená řádkově orientované uživatelské rozhraní. Z hlediska typu provedení této operace rozlišujeme paralelní a řetězový typ materializace. Řetězová používá operátory, které si předávají dílčí výsledky a jsou tak na sobě vzájemně závislé. V této kapitole se ale budeme věnovat

paralelnímu typu materializace. Z časového pohledu existují dvě základní materializační strategie, raná a pozdní. Volba té či oné strategie závisí na organizaci databázového úložiště. Musí se tedy přihlídnout k tomu, zda je implementováno slovníkové kódování, případně další komprimační metody (Abadi aj., 2007).

Strategie rané materializace spočívá v co nejčasnějším dekodování dat. Zbytek požadované operace je pak prováděn se skutečnými hodnotami. Například celočíselné identifikátory hodnot jsou nahrazeny jejich reálnými textovými hodnotami. Řádkově orientovaný výstup je tak přítomen již od prvních kroků operace.

Nyní budou popsány veškeré kroky strategie rané materializace. Nejprve jsou materializovány všechny sloupce neboli atributy, které se v dotazu vyskytují jako predikáty, nebo jsou součástí projekce. Pro každý takový sloupec se sekvenčně prochází vektor atributu na základě identifikátorů hodnot přítomných v projekci dotazu a s pomocí získaných pozic z vektoru atributu jsou identifikátory nahrazeny skutečnými hodnotami ze slovníku. Výsledkem je tedy vektor skutečných hodnot. Tento dílčí vektor je následně zredukován na základě predikátů dotazu. Podle typu a složitosti dotazu jsou sloupce nakonec zkombinovány, seskupeny nebo seřazeny, a je vyprodukován finální výsledek.

Pozdní materializace, na rozdíl od rané materializace, uchovává slovníkově zakódované celočíselné hodnoty z vektoru atributu co nejdéle v procesu. V ideálním případě je tento typ materializace proveden jako poslední krok před vrácením finálního výsledku dotazu.

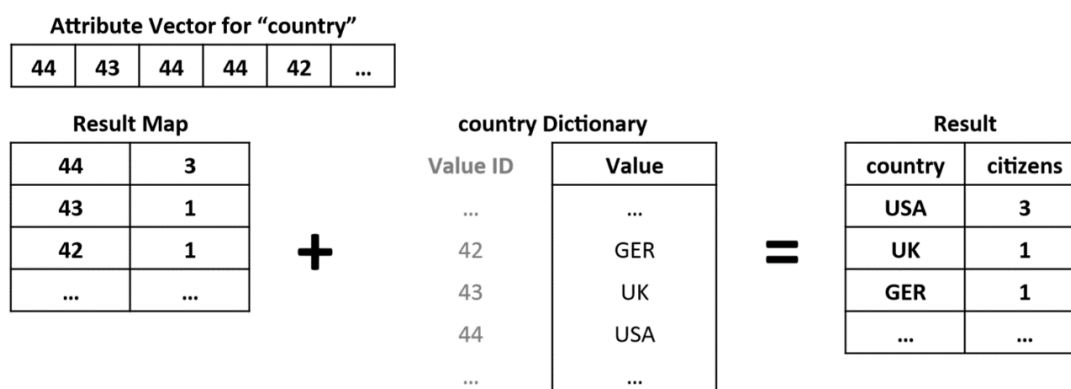
Proces se zařazenou pozdní materializací začíná hledáním požadovaných sloupců ve slovníku. Pro každý sloupec je výsledkem dílčí vektor obsahující ID všech hodnot, které splňují podmínky predikátů dotazu. Tato operace může být provedena paralelně, čímž se celý proces urychlí. Následně jsou procházeny vektory atributu a nalezené pozice hodnot jsou přidány do dílčích vektorů z minulého kroku. Pro každý sloupec tak máme vektor obsahující identifikátor hodnoty a její pozice ve sloupci. Poté jsou tyto vektory podle parametrů dotazu sloučeny, seskupeny nebo seřazeny, a konečně je provedena samotná materializace, tedy vyzvednutí skutečných hodnot ze slovníku na základě jejich ID (Plattner, 2013, s. 105-110).

Abadi aj. (2008) přišli se závěrem své analýzy, že pozdní materializace trojnásobně zrychluje zpracování dotazu, což podporuje rozhodnutí ji upřednostnit před ranou materializací.

2.2.6.5 Agregční funkce

Agregční funkce hrají zásadní roli v dotazech typických pro OLAP databáze. Jelikož jsou i v rámci praktické části této práce hojně využity, tato kapitola se bude věnovat specifikům implementace agregčních funkcí v databázi SAP HANA.

Agregční funkce obvykle pracují na velkém objemu dat a často seskupují data podle jednotlivých atributů. Mezi nejpoužívanější agregční funkce patří *COUNT* (počet), *SUM* (součet), *AVG* (aritmetický průměr), *MIN* (minimum) a *MAX* (maximum). Některé systémy pak nabízejí další speciální funkce, např. specifické funkce OLAP systému.



Obrázek 11 Agregční funkce COUNT (Zdroj: Plattner, 2013, s. 142)

Jako příklad implementace agregční funkce v in-memory databázi SAP HANA bude uvedena funkce *COUNT*. Opět použijeme tabulku světové populace. Dotaz v tomto případě směřuje na počet občanů každé země. Proces zpracování tohoto dotazu můžeme pozorovat na Obrázek 11. Nejprve se prochází vektor atributu sloužícího pro ukládání země a počítá se četnost výskytů jednotlivých identifikátorů. Výsledkem je mapa obsahující nalezené hodnoty a počet jejich výskytů. V této mapě jsou nakonec nahrazeny identifikátory hodnot jejich skutečnými hodnotami – zkratkami zemí, které se nacházejí v příslušném slovníku. Způsob implementace ostatních agregčních funkcí je založen na podobném principu (Plattner, 2013, s. 141-142).

3 ANALÝZA SOUČASNÉHO STAVU

3.1 Shrnutí teoretických poznatků

V této části budou stručně shrnuty teoretické poznatky z minulé kapitoly jako příprava půdy pro analýzu současného stavu a dnešních postupů v oblasti databázových systémů.

Již dlouho jsou databázové systémy založeny na tradičním principu. Systém řízení báze dat (SŘBD) operuje na řádkově orientovaném schématu uspořádání dat a veškeré záznamy jsou uloženy na pevných discích. Ve větších společnostech je běžně databáze rozdělena na OLTP a OLAP systémy. OLTP systém slouží k hlavnímu transakčnímu provozu a obsahuje aktuální data. Je pro něj typické velké množství krátkých operací. Na druhou stranu OLAP systém je určen ke speciálním analytickým operacím, které zpracovávají velké množství záznamů zároveň. Případem užití OLAP systémů jsou manažerské informační systémy (MIS), systémy pro podporu rozhodování (DSS) nebo exekutivní informační systémy (EIS). Nevýhodou je, že OLAP systém neobsahuje aktuální data, protože k jejich replikaci z OLTP systému dochází pouze jednou za čas. Tento proces přenosu dat do OLAP systému se nazývá ETL a skládá se z extrakce, transformace a nahrání dat. ETL proces je navíc poměrně náročný na výpočetní zdroje.

Zpracování analytických dotazů může v dnešních systémech často trvat velmi dlouho v závislosti na objemu zpracovávaných dat. Protože je dnes velmi důležité rychlé rozhodování, vzniká od velkých společností požadavek na vyšší výkon databázových systémů, aby tak mohli rychleji vypočítávat uskutečňující se trendy a lépe předvídat budoucí události. Požadovaný vyšší výkon se odvíjí od schopností nabízeného hardwaru. Jelikož úzkým hrdlem databázových systémů jsou jednoznačně pevné disky, přišel profesor Hasso Plattner, spoluzakladatel firmy SAP, s nápadem ukládat veškerá data přímo v operační paměti a pevné disky odstavit na druhou kolej. Dosavadní překážkou realizace tohoto přístupu byly nízké dostupné kapacity, a hlavně cena paměťových čipů. V posledních letech se ale tento problém podstatně zmírnil a operační paměti jsou dostupnější. Uložení dat přímo v paměti znamená též méně čekacích cyklů procesoru, a tedy lepší souhru obou komponent.

Vycházejíce ze zmíněné ideje, profesor Hasso Plattner a jeho kolegové představili novou in-memory databázi SAP HANA. Umístění veškerých dat přímo do operační paměti však

zdaleka není jedinou rozdílovou vlastností in-memory databáze. V některých výzkumech bylo zjištěno, že OLTP a OLAP systémy se od sebe z hlediska typu provozu neliší tak, jak je obecně uvažováno. Důležitým znakem in-memory databáze je tedy spojení OLTP a OLAP systému do jednoho unifikovaného systému. Pro analýzu jsou tak vždy k dispozici aktuální data, a navíc zcela odpadá potřeba ETL procesu. Díky dramaticky zvýšené rychlosti databáze již také není nutná přítomnost předem vypočítaných agregací a materializovaných pohledů. Ty jsou tak vypočítány až na vyžádání za běhu, a to na aktuální datech.

Co se architektury nového systému týče, SAP HANA je rozdělena do několika vrstev. Ty jsou uspořádány do dvou fází, kompilační a běhové. V kompilační fázi je dotaz skrze výpočetní graf přeložen do fyzického plánu a běhová část pak vygenerovaný plán provádí a manipuluje přímo s úložištěm dat. Důležité je zde řízení životního cyklu záznamu, kde každý záznam během své životnosti prochází několika typy úložných prostorů. První úložiště (L1-delta) je optimalizováno pro vkládání nových záznamů a ty poté putují do dalších prostorů, které jsou již optimalizovány pro čtení a analýzu (L2-delta, hlavní úložiště). K přesunu dat mezi těmito prostory slouží proces slučování.

Další stěžejní vlastností SAP HANA je sloupcově orientované uspořádání dat. To znamená, že hodnoty jednoho sloupce, atributu, jsou v paměti uloženy za sebou. Ne tedy po řádcích, tj. jednotlivých záznamech, jako v tradičním řádkově orientovaném úložišti. Sloupcové schéma je vhodné právě pro analytické operace, kde je často vyžadováno jen několik málo atributů, zejména pro agregační funkce.

Jelikož je kapacita operační paměti stále poměrně omezená, používá se v in-memory databázi tzv. slovníkové kódování. To kromě razantní komprese dat přináší také zrychlení odezvy na některé typy dotazů. Funguje na principu slovníku, který je vytvořen pro každý sloupec, obsahuje všechny unikátní hodnoty daného sloupce a přiřazuje jim celočíselný identifikátor. Identifikátory hodnot je pak naplněn vektor atributu, který již reflektuje původní uspořádání hodnot ve sloupci. Slovníkové kódování je opět vhodnější pro operace čtení, jelikož vkládání nových záznamů přináší komplikace ve smyslu změny slovníku, přemapování identifikátorů hodnot a reorganizace vektoru atributu. Proto jsou v rámci řízení životního cyklu záznamu používána tři různá úložiště. Řádkově orientované L1-delta optimalizované pro zápis, sloupcově orientované L2-delta se

slovníkovým kódováním, a hlavní úložiště vhodné pro čtení dat, které navíc udržuje slovníky seřazené. Kromě slovníkového kódování mohou být rovněž využity další kompresní metody, jako je např. prefixové kódování.

Za účelem dosažení co nejvyššího stupně paralelismu je v in-memory databázi používán tzv. partitioning. Jedná se o rozdělení databáze do několika objektů. Vertikální partitioning dělí tabulky podle sloupců, horizontální podle řádků. Navenek ovšem databáze musí zůstat pro uživatele transparentní, jako by nebyla rozdělena.

Vlivem uspořádání dat ve sloupcovém schématu a použití slovníkového kódování pro kompresi dat je nutné příslušným způsobem upravit zpracování základních databázových operací. U odstranění a vkládání záznamu je často nutná reorganizace slovníku a vektoru atributů, zatímco operace čtení může těžit z paralelismu a práce s celočíselnými identifikátory hodnot atributů. Při implementaci databázových operací pracujících na sloupcově orientovaném úložišti dat je též potřeba provést materializaci, aby byl výstup organizován tradičně podle řádků. Raná materializace provádí tuto transformaci na prvním dostupném mezivýsledku, kdežto pozdní je aplikována až na finální výsledek. Důležité je zmínit agregační funkce, které v in-memory databázi těží z celočíselné reprezentace hodnot a rychlého procházení atributů ve sloupcově orientovaném schématu.

3.2 Business Intelligence

3.2.1 Základní informace o BI

S analytickými databázovými systémy úzce souvisí pojem Business Intelligence (BI). Ten zahrnuje celou množinu činností, úloh a technologií, které jsou součástí podnikových procesů a zejména podnikových informačních systémů. Produkty BI jsou založeny na multidimenzionálním uložení a zpracování dat, které vešlo ve známost již v 80. letech minulého století, a bylo využíváno zejména v exekutivních informačních systémech (EIS). Dále se začaly v multidimenzionálních technologiích prosazovat i další trendy jako datové sklady nebo datová tržiště. Jelikož objem zpracovávaných a ukládaných dat se postupem času rapidně zvětšuje, dostávají se ke slovu i nástroje k dolování dat, které sestávají z vysoce sofistikovaných analytických postupů s využitím matematických a statistických metod.

Kvůli stále tvrdšímu konkurenčnímu prostředí se manažeři firem musí rozhodovat rychle a správně. Aby toho byli schopni, využívají právě produktů Business Intelligence. V multidimenzionálním náhledu na data potřebují pružně měnit kritéria a přistupovat k agregovaným datům podle nejrozličnějších úrovní. Jelikož je klasický ERP systém většinou zahlcen transakčními operacemi, nedokáže již poskytnout potřebný výkon ke zpracování složitých analytických operací. Proto se kromě OLTP databáze začala vytvářet také oddělená OLAP databáze, sloužící výhradně pro analýzu dat, trendů, vzorců a predikci budoucího vývoje.

Business Intelligence zahrnuje široký komplex nástrojů a aplikací, které jsou postaveny na architektuře IS/ICT. Musí také komunikovat s dalšími aplikacemi IS/ICT, ze kterých buďto data čerpá nebo je do nich posílá zpět. Z toho můžeme usoudit, že BI je vysoce závislá na kvalitě dat a databází, na kterých ostatní IS/ICT aplikace pracují.

Architektura multidimenzionálních databází je navržena pro ukládání a interaktivní prohlížení multidimenzionálních dat. Je založena na využívání tabulek, které jsou schopny pojmout několik dimenzí. Vzniká tak datová struktura podobná Rubikově kostce. První dimenzí je vždy čas a dalšími pak jednotlivé ekonomické ukazatele. Je zde zajištěna také agregace hodnot na různých úrovních, přičemž pohyb mezi nimi je založen na principu drill-down a drill-up. Drill-down zpřístupňuje data pro nižší úroveň agregace a drill-up funguje opačným směrem.

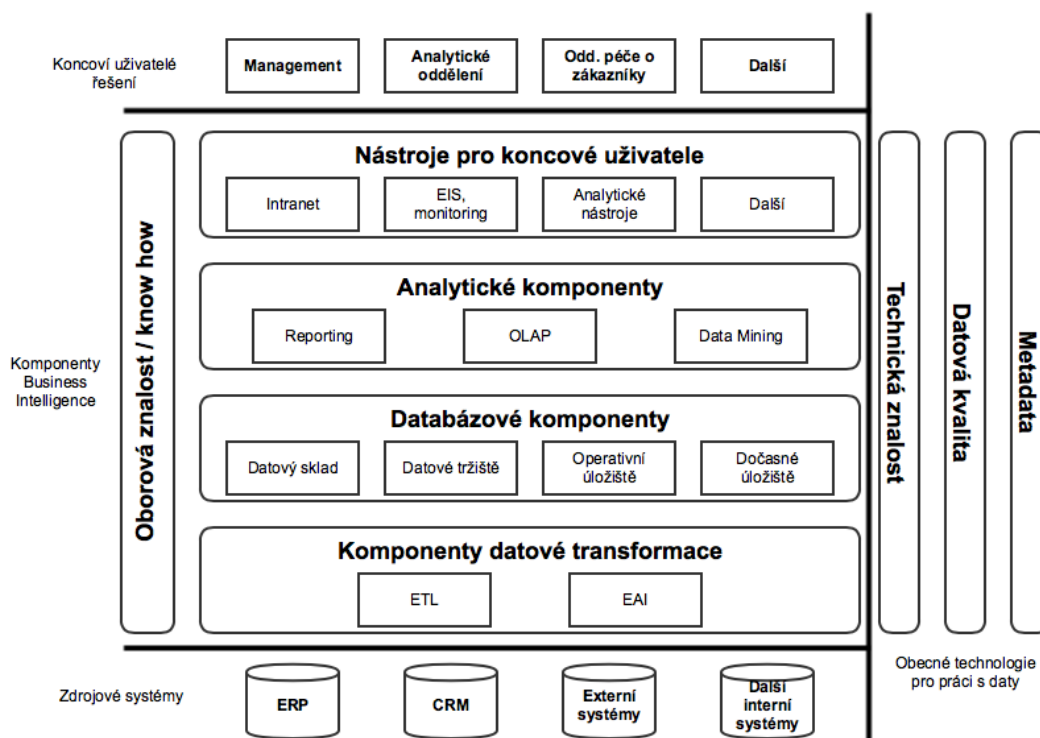
Na úrovni relační databáze mohou být tabulky v BI provázány za pomoci různých typů datových schémat. Jedná se o schéma hvězdy a schéma sněhové vločky. Centrum schématu vždy tvoří tabulka faktů obsahující sledované ukazatele, které jsou identifikované klíči, složených z klíčů multidimenzionálních tabulek. Ty tedy tabulku faktů obklopují.

Na základě rozdílů mezi OLTP a analytickými systémy popsaných v kapitole 2.1.3.1 je možno uvést základní principy, ze kterých vycházejí aplikace Business Intelligence. Prvním principem je orientace výhradně na analytický provoz a plánovací potřeby uživatelů, nikoliv na transakční provoz. Navíc zdrojem dat analytických systémů jsou právě transakční systémy. Další charakteristikou je ukládání dat společně s časovým rozlišením do hierarchické struktury dimenzí, čímž vzniká multidimenzionální databáze.

Výsledkem jsou automaticky agregovaná data, na které lze pohlížet z různých dimenzí (Novotný aj., 2005).

3.2.2 Architektura BI

Koncepce architektury Business Intelligence se může v různých případech lišit v závislosti na potřebách podniku nebo zákazníka. Přesto zde pro lepší představu bude uveden obecný model architektury BI tak, jak je aktuálně vnímána, viz Obrázek 12. Architekturu BI lze rozdělit do několika vrstev. V té nejnížší jsou komponenty datové transformace, které zajišťují sběr dat ze zdrojových systémů do vyšších vrstev pro ukládání dat. Těmito komponentami jsou buď ETL systémy, které slouží pro extrakci, transformaci a nahrávání dat (ETL proces), nebo EAI (Enterprise Application Integration) systémy pro integraci aplikací.



Obrázek 12 Obecná koncepce architektury BI (Zdroj: Vlastní zpracování dle: Novotný aj., 2005)

Ve druhé vrstvě jsou databázové komponenty využívané k ukládání dat, jejich aktualizaci a správě. Jsou to datové sklady (základní BI databázová komponenta), datová tržiště

(nastavba datových skladů), operativní datová úložiště (podpůrné analytické databáze) a dočasná úložiště dat (přechodná úložiště před zpracování dat do ostatních komponent).

Do třetí vrstvy architektury BI patří analytické komponenty sloužící k vlastnímu zpracování dat a jejich analýze. K těmto komponentám patří reporting, OLAP, a dolování dat (Data Mining). Reporting je zaměřen na standardní a ad hoc dotazovací procesy nad databází. Již mnohokrát zmiňovaný OLAP pak na pokročilé a dynamické analytické úlohy. Nakonec komponenta dolování dat je určena na sofistikovanou analýzu velkého množství dat.

Ve čtvrté vrstvě jsou nástroje pro koncové uživatele, které zabezpečují komunikaci mezi uživatelem a ostatními komponentami BI systému. Tato vrstva se také nazývá prezentační. Dochází zde ke sběru požadavků na analytické operace a k následné prezentaci výsledných dat. Součástí prezentační vrstvy jsou portálové webové aplikace (intranet), exekutivní informační systémy (EIS) a další analytické nástroje.

Poslední vrstvou architektury Business Intelligence je vrstva oborové znalosti (know how), která čerpá ze všech předchozích vrstev. Zahrnuje oborovou znalost a osvědčené postupy (best practises) v nasazování řešení BI.

Stranou všech vrstev jsou pak komponenty pro správu a manipulaci s daty. Jedná se o správu metadat (popis a dokumentace systému a procesů), nástroje pro zajištění kvality a komponentu technické znalosti obsahující schopnosti implementačního týmu (programovací a technologické).

Nejspodnější vrstva datové transformace komunikuje přímo se zdrojovými systémy (ERP, CRM, externí a interní systémy) a nejvyšší prezentační vrstva pak přímo s koncovými uživateli BI řešení. Tím je specifikováno komunikační okolí architektury Business Intelligence (Novotný aj., 2005).

3.2.3 Trendy vývoje v BI

Segment Business Intelligence aplikací se vyvíjí velmi rychle. Když vezmeme v úvahu celý IS/ICT sektor, transakční systémy jsou dnes již dostatečně vyspělé a evolučně stabilní. Naproti tomu analytické systémy zažívají v poslední dekádě překotný vývoj. Zatímco dříve byly k dispozici jednoduché analytické vizualizační nástroje, dnes je nabízeno mnoho řešení BI, která jsou již poměrně složitá, ale zároveň velmi mocná. Tato

řešení zahrnují komponenty, které byly blíže popsány v předchozí kapitole pojednávající o architektuře Business Intelligence.

Novotný aj. (2005) rozlišují čtyři sektory na trhu BI. Je to databázový sektor, sektor ETL, sektor pokročilých analytických aplikací a sektor technologií dolování dat. Každý z těchto sektorů má své typické dodavatele. Někteří dodavatelé se navíc úspěšně pohybují v několika sektorech zároveň, což lze považovat za současný trend. Mezi hlavní velké dodavatele se řadí IBM, Microsoft, Oracle a SAP.

Dalším trendem v oblasti BI je konvergence nabídky služeb na jednotnou technologickou platformu, konvergence řešení z oblastí zdrojových systémů a BI, a též konvergence reportingových řešení a řešení pro pokročilou analýzu dat. Zde velkou roli hraje právě firma SAP se svou nabídkou ERP systémů. Dále je patrné slučování aktivit na trhu ETL nástrojů a konvergence dat obsažených v BI produktech. To znamená například schopnost zpracovávat nestrukturovaná data, jak bylo popsáno v kapitole 2.1.1.2.

Na celosvětovém trhu je pozorován posun k dodavatelskému způsobu vývoje řešení BI a také dochází ke standardizaci jednotlivých komponent a rozhraní BI produktů za účelem vyšší flexibility a znovupoužitelnosti. V neposlední řadě se společnosti zaměřují na data o zákaznících a analyzují jejich chování.

V poslední době dochází k využívání Business Intelligence řešení nejen ve vrcholném managementu firem, ale také v ostatních vrstvách podnikové hierarchie. S BI aplikacemi tak pracují všechna oddělení a používají je rovněž k analýze a podpoře rozhodování. Konkrétně můžeme jmenovat oddělení pro řízení rizik či útvar podniku pro komunikaci s dodavateli a zákazníky. Můžeme též pozorovat rychlý rozvoj technologií dolování dat a datových skladů z hlediska zpracování úloh v reálném čase (Novotný aj., 2005).

3.3 SAP MaxDB

SAP MaxDB je tradiční relační databáze, která v této práci zastává důležitou roli. Reportovací aplikace (popsaná v kapitole 4.1) bude spuštěna jak na této databázi, tak na in-memory databázi SAP HANA a obdržené výsledky budou porovnány a diskutovány. Výkon MaxDB je tak výchozím referenčním bodem pro srovnání.

SAP MaxDB je kompletní systém řízení báze dat a obsahuje mnoho vestavěných administrátorských a vývojářských funkcí. Je dostupný na všech běžně používaných

operačních systémech. Jeho historie sahá až na konec 70. let 20. století, kdy byl vyvíjen pod mnoha různými názvy, například Adabas D. V roce 2003 byl systém přejmenován na MaxDB a od té doby byl hlavní databázovou platformou pro ERP produkty firmy SAP.

S rozmachem objektově orientovaného programování byl systém MaxDB rozšířen o technologii SAP liveCache, která již umožňuje práci s objekty. Obsahuje datové struktury a metody pro rychlé plánování produkce a zpracování plánovacích algoritmů ATP, které byly popsány v kapitole 2.1.1.3 (Malik, 2018).

Databázový systém MaxDB sestává z jádra databáze, nástrojů a rozhraní. Mezi nástroje patří instalační správce pro instalaci a upgrade, databázové studio pro správu databáze, analyzátor pro sledování výkonu databáze a nástroje příkazového řádku. Mezi podporovaná rozhraní patří JDBC, ODBC a SQLDBC (Malik, 2018).

3.4 SAP S/4 HANA

Zatímco databáze SAP HANA byla představena v roce 2010, ucelené ERP řešení postavené výhradně na této in-memory databázi je dostupné od roku 2015 jako SAP S/4 HANA. Je to de facto úplně nová generace produktů SAP, protože komponenty byly přepracovány jak zevnitř pro vysoký výkon postavený na moderním hardwaru a in-memory databázi, tak zvenku díky novému konceptu uživatelskému rozhraní SAP Fiori. Obsahuje všechny požadované podnikové komponenty v rámci ERP systémů, viz Obrázek 13. Je to správa financí, lidských zdrojů (HR), získávání a zadávání veřejných zakázek, správa výroby, marketingu, dodavatelského řetězce, obchodu, aktiv, údržby a výzkumu a vývoje. Další komponenty je možno přikoupit. SAP S/4 HANA lze provozovat buď přímo u zákazníka nebo v cloudu, což je dnešní velký trend.



Obrázek 13 Komponenty SAP S/4 HANA (Zdroj: Vlastní zpracování dle: Nascimento, 2016)

SAP S/4 HANA obsahuje aplikace, které byly kompletně přepsány takovým způsobem, aby co nejvíce využily možností in-memory databáze. Produkt tak funguje výhradně na databázi SAP HANA, není možné jej instalovat na databáze třetích stran. Avšak díky této nativní implementaci představuje výkonnou databázovou platformu, která zahrnuje prediktivní analýzu, pokročilé dolování textů a podporu rozhodování v reálném čase. Uživatelské rozhraní SAP Fiori je lehce přizpůsobitelné pro zobrazení na různých zařízeních, takže kromě stolních počítačů a notebooků lze k systému pohodlně přistupovat i z chytrých telefonů a tabletů.

Další vlastnosti produktu SAP S/4 HANA vycházejí přímo ze schopností in-memory databáze, popsanych v teoretické části této práce. Je to například zjednodušený datový model s nízkou redundancí a vysokou flexibilitou (OLTP a OLAP spojeny dohromady), provádění operací ve stylu OLAP v reálném čase na vyžádání nebo volné textové vyhledávání v rámci celé databáze (Nascimento, 2016).

3.5 Vývojové prostředí

V této kapitole se již budeme věnovat přípravě půdy pro praktickou část této práce. Jelikož cílem je vytvořit reportovací aplikaci, která je schopna běhu jak na tradiční MaxDB tak i na in-memory databázi HANA, bylo potřeba zajistit přístup k odpovídajícím prostředkům a nástrojům pro účely vývoje.

3.5.1 Systémy

Pro vývoj aplikace bylo zvoleno prostředí systému SAP R/3, což je předchůdce řešení SAP S/4 HANA. Autor této práce pracuje ve společnosti SAP, proto nebyl problém získat přístup do vhodných systémů. Podmínkou bylo, aby jeden z nich běžel na MaxDB databázi a druhý na in-memory databázi HANA. Na základě doporučení byly zvoleny systémy s označením ZKU a ZPU, oba na klientech 400. ZKU je testovací systém postavený na databázi MaxDB a jeho parametry jsou následující:

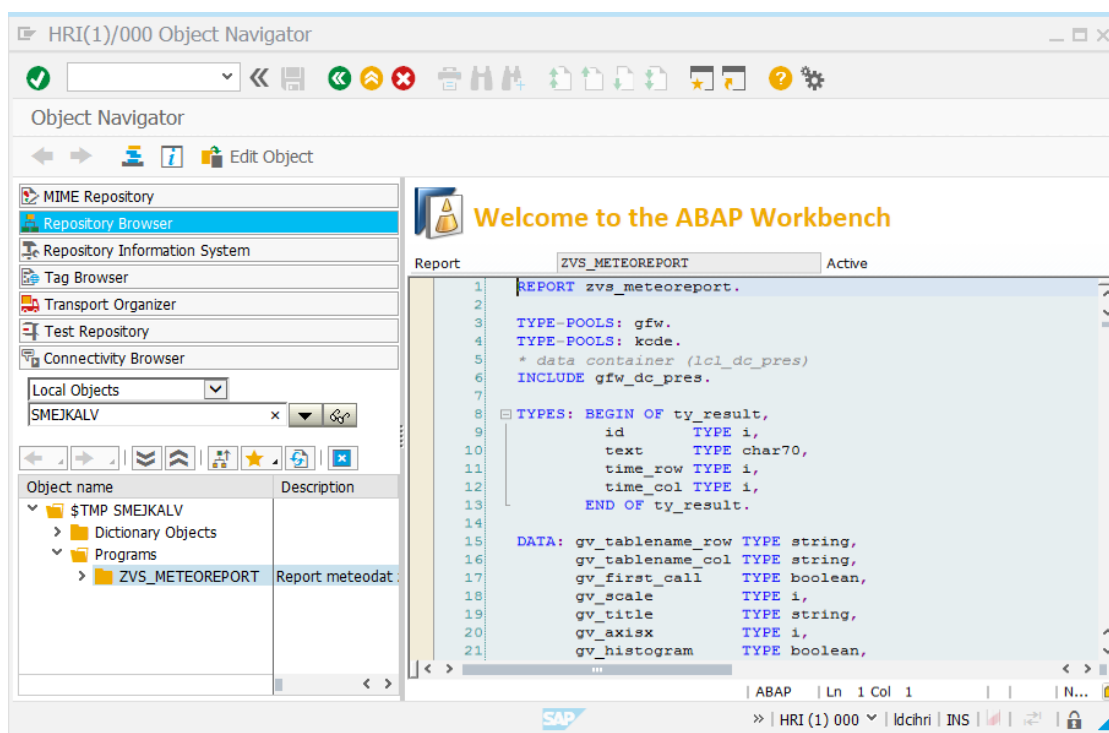
- Operační systém: Linux, 64 bitová architektura
- Nainstalovaná verze produktu: EHP7 pro SAP ERP 6.0
- Procesor: 16 procesorů Intel Xeon E7-8870 s 2,4 GHz
- Operační paměť: 65,552 GB

Druhý systém ZPU běží na in-memory databázi SAP HANA a má tyto parametry:

- Operační systém: Linux, 64 bitová architektura
- Nainstalovaná verze produktu: EHP7 pro SAP ERP 6.0
- Procesor: 8 procesorů Intel Xeon E7-4830 s 2,13 GHz
- Operační paměť: 261,120 GB

3.5.2 ABAP

Kód aplikací ve vývojovém prostředí ABAP Workbench je psán v programovacím jazyce ABAP (Advanced Business Application Programming), což je proprietární jazyk vyvinutý přímo firmou SAP. Jedná se o strukturovaný, imperativní, objektově orientovaný a typovaný jazyk, který je nezávislý na platformě a dopředně kompatibilní. ABAP sdílí některé vlastnosti s ostatními programovacími jazyky, jako např. SQL nebo COBOL, používaný pro podnikové a databázové aplikace. Standard Open SQL je ve skutečnosti přímo vestavěn do jazyka ABAP, takže v jeho kódu je možné používat standardní příkazy pro manipulaci s databázemi.



Obrázek 14 Vývojové prostředí ABAP Workbench

V programovacím jazyce ABAP je možno vytvořit několik typů programů. Buď jsou to spustitelné programy jako reporty a tzv. module pooly (složitější reporty s pokročilou funkcionalitou obrazovek), nebo pomocné programy, které bývají využity v reportech. Jsou to subrutiny (lokální funkce v reportu), funkční skupiny obsahující funkční moduly, includey (rozšíření reportů) nebo type pooly (soubor definicí datových typů).

Aplikace jsou v SAP systému vyvíjeny pomocí nástroje ABAP Workbench, viz Obrázek 14. Jedná se o vývojové prostředí, které nabízí funkce k editování kódu aplikace a ladění (debugging). ABAP Editor je textový editor kódu aplikace a Object Navigator slouží k procházení všech druhů vývojových objektů. Dále jsou k dispozici komponenty pro editaci ostatních částí aplikace. ABAP dictionary obsahuje definici tabulek a datových elementů, Screen Painter je grafický editor pro konfiguraci obrazovek, Menu Painter je nástroj pro návrh uživatelského rozhraní, Function Builder pro správu funkčních modulů (jedná se o globální funkce využitelné v různých aplikacích) a Class Builder pro vytváření a editaci globálních tříd a rozhraní (Kühnhauser, 2009).

3.6 Testovací data

Aby mohlo dojít k řádnému otestování a porovnání výkonu mezi klasickou databází MaxDB a in-memory databází HANA, bylo potřeba obstarat dostatečný objem dat, na kterých by testování proběhlo. Ve velkých podnikových databázích jsou často miliony a někdy až miliardy záznamů v tabulkách. Ve zvolených SAP systémech, které byly popsány v předchozí kapitole, bohužel nebyly žádné testovací sady dat k dispozici. Proto byla snaha nalézt veřejně dostupný dataset, který by povahou a objemem dat vyhovoval potřebám reportovací aplikace. Nakonec byla zvolena sada dat, která obsahuje hodnoty z měření meteostanice brněnské hvězdárny za posledních pět let (únor 2012 až březen 2017). Dataset je zveřejněn na webu Datahub, což je platforma pro otevřenou správu dat, založenou na systému CKAN, který má za cíl poskytnout prostor pro zveřejňování dat vládními organizacemi, státními podniky a soukromými firmami. Jedná se o open source systém (Datahub, 2017).

3.6.1 Schéma databáze

Stažená data jsou ve formátu CSV. Aby reportovací aplikace byla schopna řádně zpracovat všechny atributy a vložit je do tabulky, některé z nich jsou upraveny takovým způsobem, aby svou formou lépe odpovídaly datovým typům používaným v jazyce ABAP. Například atribut identifikující datum a čas byl rozdělen a také byly přidány další atributy pro měsíc v roce a rok, to všechno za účelem flexibilnějšího dotazování nad databází a zajímavějších agregací.

ZVSMETEO
DATUM
CAS
MESIC
ROK
VNITRNI_TEPLOTA
VENKOVNI_TEPLOTA
ROSNY_BOD
VNITRNI_VLHKOST
VENKOVNI_VLHKOST
RYCHLOST_VETRU
SMER_VETRU_NUM
SMER_VETRU_TEXT
POCITOVA_TEPLOTA
SRAZKY_1H
SRAZKY_24H
SRAZKY_CELKOVE
TLAK
TLAKOVA_TENDENCE
PREDPOVED

Obrázek 15 Schéma testovací databáze

Testovací data jsou charakterizována naměřenými meteorologickými veličinami. Všechny atributy jsou součástí jedné tabulky. Schéma testovací databáze je znázorněno na Obrázek 15. Jméno tabulky je *ZVSMETEO* a primární klíč tvoří dvojice atributů datum a čas. Pro lepší možnosti agregace tu jsou dále atributy měsíc (perioda včetně roku) a rok. Meteorologické hodnoty začínají vnitřní a venkovní teplotou ve °C, dalšími atributy jsou rosný bod (taktéž °C) a vnitřní a venkovní vlhkost v %. Následuje rychlost větru v m/s a směr větru. Ten je vyjádřen číselně (úhel ve °) a textově (např. N – sever). Je zde také pocitová teplota (°C) a srážky za poslední hodinu, den a celkové srážky od začátku měření v mm. Posledními atributy jsou tlak (v hPa), tlaková tendence (stoupající/klesající) a předpověď (slunečno/zataženo/srážky).

3.6.2 Uspořádání dat

Jelikož reportovací aplikace bude porovnávat rychlost vkládání záznamů a zpracování dotazů v řádkově a sloupcově orientovaném uspořádání dat, potřebujeme dva typy tabulky *ZVSMETEO*, jednu s řádkovým schématem a jednu se sloupcovým. Obě tabulky budou vytvořeny jako kopie základní tabulky *ZVSMETEO*.

3.6.3 Objem testovacích dat

Získaný dataset meteorologických měření má 592 975 záznamů. Jak již bylo zmíněno, ve větších podnikových databázích se běžně nachází miliony záznamů. Proto budou vytvořeny další dvě databázové tabulky, ve kterých budou data uměle namnožena na desetinásobek, tedy téměř 6 milionů záznamů. V konečném důsledku budeme mít čtyři různé tabulky. Dvě tabulky standardní velikosti a dvě s namnoženými daty. V těchto dvojicích se pak budou lišit schématem uspořádání dat (řádkový/sloupcový). Budeme tedy moci pozorovat rozdíly v rychlosti zpracování obou objemových úrovní. Jde nám hlavně o ověření schopnosti in-memory databáze SAP HANA zpracovávat analytické dotazy a agregace v reálném čase na vyžádání, tedy schůdnost odezvy např. v případě zadání dotazu na mobilním zařízení.

3.7 Modely testování

Cílem této práce je otestovat a porovnat rozdíly výkonnostní rozdíly mezi tradiční databází MaxDB a in-memory databází SAP HANA, a v rámci druhé zmíněné též rozlišit mezi řádkově a sloupcově orientovaným uspořádáním dat.

V praktické části práce bude měřeno a vyhodnocováno několik parametrů. Zaprvé se bude měřit čas vkládání všech dat do tabulek. Aby bylo dosaženo co možná nejpřesnějších výsledků, import záznamů bude proveden celkem 5krát a naměřené časy budou zprůměrovány. Reportovací aplikace bude naměřené hodnoty ukládat do speciálně vytvořené tabulky *ZVSINSTIMES*, takže k nim bude možné přistupovat i po ukončení programu.

Dále budeme pozorovat, kolik fyzického místa v paměti (na pevném disku v případě MaxDB, v operační paměti u HANA databáze) tabulky zabírají. Zde nás hlavně bude

zajímat, zda je pozorovatelný efekt slovníkového kódování v in-memory databázi. Důležité je si uvědomit, že ačkoliv jsou v MaxDB databázi také nakonfigurovány řádkově a sloupcově orientované tabulky, toto nastavení zde nemá žádný vliv a implicitně jsou všechny tabulky uloženy v řádkovém schématu. Předpokládáme, že v MaxDB by tedy neměl být žádný rozdíl mezi tabulkami s řádkovým a sloupcovým uspořádáním dat. Vliv datového uspořádání tedy budeme zkoumat pouze v HANA databázi.

Stěžejní částí modelu testování bude spouštění reportovací aplikace s různě nastavenými parametry podle požadované charakteristiky a složitosti dotazu. Budou vybrány takové dotazy, aby demonstrovaly vliv užití predikátů a agregačních funkcí na rychlost odezvy. Na základě informací z kapitoly 2.2.2.2 bude též zkoumáno zpracování příkazu *SELECT ** vyžadující rekonstrukci celého záznamu, což by mělo představovat nevýhodu pro sloupcově uspořádanou tabulku. Opět jako u měření rychlosti vkládání do tabulek budou dotazy zpracovány celkem 5krát a odezvy zprůměrovány. Aplikace bude spuštěna zvlášť na systému ZKU s databází MaxDB a systému ZPU s in-memory databází HANA.

4 VLASTNÍ NÁVRH ŘEŠENÍ

4.1 Reportovací aplikace

Praktická část této práce spočívala v návrhu a vývoji aplikace v prostředí SAP R/3, která je schopna reportingu na testovacím datasetu meteorologických dat a měření rychlosti zpracování analytických dotazů na řádkově i sloupcově orientované databázi, včetně importu dat do databáze ze zdrojových souborů. Aplikace musí být spustitelná na obou typech zkoumaných databází, tedy MaxDB i HANA.

4.1.1 Tabulky

Pro účely aplikaci byla vytvořena vzorová tabulka *ZVSMETEO*, která odpovídá schématu na Obrázek 15 z kapitoly 3.6.1. Definice této tabulky v SAP systému je znázorněna na Obrázek 16. Součástí definice jednotlivých atributů jsou datové elementy, což jsou pojmenované datové typy z ABAP Dictionary a jsou odvozeny od vestavěných datových typů jazyka ABAP.

Transparent Table

ZVSMETEO

Active

Short Description

Meteo data

Attributes

Delivery and Maintenance

Fields

Input Help/Check

Currency/Quantity Fields

</

Obrázek 16 Definice databázové tabulky ZVSMETEO

Protože pro účely reportovací aplikace potřebujeme dvě verze této tabulky, řádkově uspořádanou a sloupcově uspořádanou, byla tabulka *ZVSMETEO* použita jako vzor pro vytvoření tabulek *ZVSMETEO_ROW* a *ZVSMETEO_COL*. Prvně zmíněná je nastavena pro řádkově orientované uspořádání dat a druhá pro sloupcově orientované uspořádání dat. Tato vlastnost se dá nastavit přímo v konfiguraci tabulky.

Jak bylo naznačeno v kapitole 3.6.3 o objemu testovacích dat, abychom mohli pozorovat vliv objemu dat na zpracování databázových operací v MaxDB a HANA databázích, byly vytvořeny další dvě tabulky, *ZVSMETEOXL_ROW* a *ZVSMETEOXL_COL*, které obsahují desetinásobné množství záznamů oproti základním tabulkám. Pro každý originální záznam vzniklo 9 dalších záznamů, lišících se pouze primárním klíčem, konkrétně časem posunutým vždy o sekundu. Přehled databázových tabulek, které byly vytvořeny a naplněny testovacím daty, je znázorněn na Obrázek 17.

	Řádkové uspořádání	Sloupcové uspořádání
600 tisíc záznamů	ZVSMETEO_ROW	ZVSMETEO_COL
6 milionů záznamů	ZVSMETEOXL_ROW	ZVSMETEOXL_COL

Obrázek 17 Přehled databázových tabulek

Dále byla vytvořena pomocná tabulka *ZVSINSTIMES*, která slouží pro ukládání naměřených časů vkládání záznamů do tabulek, takže je přístup k těmto hodnotám možný i po skončení běhu programu. Tabulka obsahuje datum a čas vkládání, počet vkládaných záznamů, cílovou tabulku (standardní/velká) a naměřený čas pro řádkové uspořádání a sloupcové uspořádání.

4.1.2 Návrh a vývoj aplikace

Reportovací aplikace byla vyvinuta s pomocí nástroje ABAP Workbench jako spustitelný report. Reporty v prostředí SAP R/3 nabízejí uživateli jako první tzv. selekční obrazovku, kde je možno nastavit parametry běhu reportu, např. časovou periodu, za kterou se mají

výsledky spočítat. Jakmile jsou parametry nastaveny, report se spustí tlačítkem *Execute* a dodá požadovaný výstup.

Report může obsahovat několik obrazovek, mezi kterými uživatel v průběhu používání reportu prochází. Každá obrazovka má svoje identifikační číslo a je navržena pomocí nástroje Screen Painter. Také obsahuje dvě programovací části. Část *PROCESS BEFORE OUTPUT* slouží k nastavení stavů a inicializací prvků obrazovky před jejím zobrazením uživateli, *PROCES AFTER INPUT* pak ke zpracování vstupů od uživatele prostřednictvím tlačítek, textových polí a dalších elementů uživatelského rozhraní a předává příslušné informace vnitřní logice aplikace.

4.1.2.1 Selekční obrazovka

Selekční obrazovka reportovací aplikace je rozdělena na dvě záložky. První, viz Obrázek 18, je určena ke správě databáze, nabízí tedy možnost smazat obsah tabulky anebo do ní vložit nová data. Také jsou zde zobrazeny naměřené údaje o posledním vkládání dat a aktuální počet záznamů ve standardní a velké databázi. Uživatel může zvolit, s jakou databází bude report pracovat (standardní/velká). Veškeré nastavené operace jsou pak prováděny na zvolené tabulce.

Report meteorological data from meteorological station Brno

Databáze Konfigurace dotazů

Statistika DB

Počet záznamů v standardní DB:	592.975
Počet záznamů ve velké DB:	5.929.738

Výběr databáze a poslední statistika

☐ Standardní databáze

☒ Velká databáze

Počet vkládaných záznamů:	5.929.760
Čas vkládání do řádků:	3.647.113.973
Čas vkládání do sloupců:	4.925.103.506

Manipulace s vybranou DB

☒ Smazat aktuální obsah DB

☒ Vložit nová data

1. soubor

☐ Přidat soubor

Obrázek 18 Správa databáze

Při nahrávání nových dat do databáze musí uživatel poskytnout odpovídající zdrojové soubory. Pro otevření a konverzi dat ze souboru do interní tabulky programu je využita globální funkce *KCD_CSV_FILE_TO_INTERN_CONVERT*, která je však schopna na jedno spuštění zpracovat maximálně 100 000 záznamů (řádků). Z tohoto důvodu byl původní zdrojový soubor rozdělen na části o velikosti tohoto limitu. Na selekční obrazovce je pak zahrnuta možnost dynamicky přidávat další zdrojové soubory, aby bylo možné provést import dat do databáze v jedné dávce a jednom měření.

Obrázek 19 Konfigurace dotazů

Druhá záložka na selekční obrazovce slouží ke konfiguraci dotazů nad zvolenou databází. Je možné ji vidět na Obrázek 19. Uživatel si může nastavit rozsah časové periody, za kterou bude zvolená veličina reportována. Dále jsou již na výběr meteorologické veličiny. V prvním bloku je možno zvolit mezi teplotou (vnitřní a venkovní), vlhkostí (vnitřní a venkovní), rychlostí větru a tlakem. Pro vybranou veličinu se zde dá nastavit omezení odpovídající predikátu *WHERE*, např. chceme brát pouze záznamy s hodnotou vlhkosti

45–50 %. Nakonec uživatel volí požadovaný typ dotazu na sledovanou veličinu. K dispozici je klasický vývoj v čase (varianta *SELECT **, rekonstrukce celých záznamů), druhá varianta tohoto dotazu s výběrem pouze požadovaného atributu, a dále průměrné/maximální denní hodnoty, průměrné/maximální měsíční hodnoty a průměrné/maximální roční hodnoty. Dalšími nabízenými veličinami jsou celkové srážky (denní/měsíční/roční) a směr větru v číselné nebo textové variantě. V případě průměrných a maximálních hodnot se jedná o agregované dotazy s využitím predikátu *GROUP BY*. Je možné nastavit a spustit více dotazů současně.

4.1.2.2 Přehled provedených dotazů

Po spuštění reportu pomocí tlačítka *Execute* dojde k provedení nastavených akcí. V případě zvolení alespoň jednoho dotazu se následně zobrazí obrazovka s přehledem těchto dotazů a jejich časem zpracování (v mikrosekundách získaného pomocí příkazu *GET RUN TIME FIELD*) v obou typech uspořádání dat. Tato obrazovka je vidět na Obrázek 20.

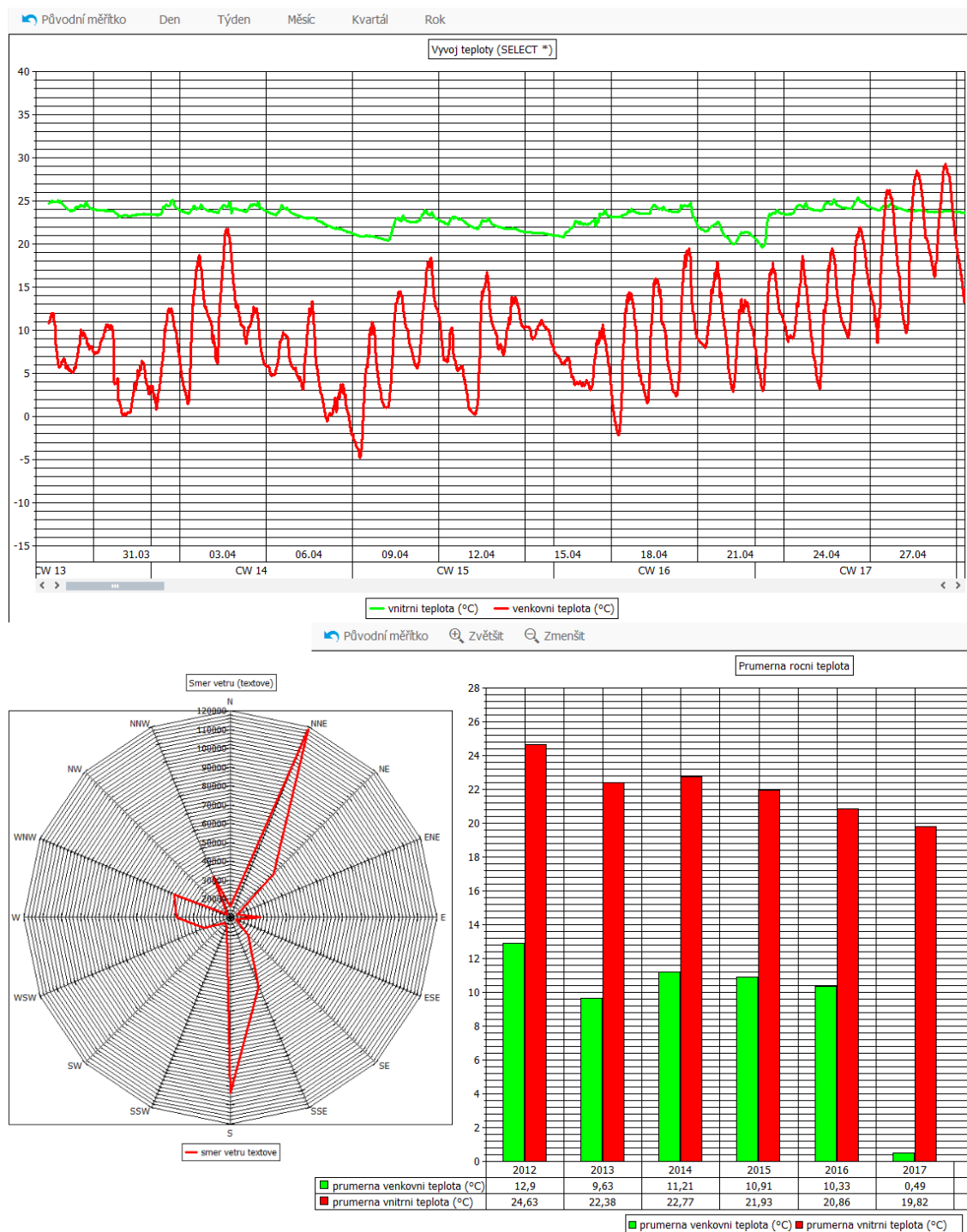
Report meteodat z meteorologické stanice Hvězdárna Brno				
ID	TEXT	TIME ROW	TIME COL	
1	Vývoj teploty (SELECT *)	16.037.865	23.393.790	 Graf
2	Vývoj teploty (výběr atributů)	4.277.014	7.909.229	 Graf
3	Průměrná denní teplota	153.512	318.779	 Graf
4	Maximální denní teplota	116.145	28.306	 Graf
5	Průměrná měsíční teplota	152.808	49.348	 Graf
6	Maximální měsíční teplota	116.938	24.807	 Graf
7	Průměrná roční teplota	147.754	45.824	 Graf
8	Maximální roční teplota	113.160	23.019	 Graf
25	Směr větru (číselně)	95.238	18.218	 Graf
26	Směr větru (textově)	122.002	20.730	 Graf
27	Celkové denní srážky	123.426	38.874	 Graf
28	Celkové měsíční srážky	110.803	21.994	 Graf
29	Celkové roční srážky	102.993	18.237	 Graf

Obrázek 20 Přehled provedených dotazů

4.1.2.3 Vykreslení grafů

U každého zpracovaného dotazu je pak možné vykreslit příslušný graf. V případě vývoje veličiny v čase je to spojnicový graf, u agregací sloupcový graf a pro znázornění směru větru je využit pavučinový (paprskový) graf. Měřítko spojnicového grafu lze přizpůsobit pro pohled na jeden den, týden, měsíc, kvartál nebo rok. Sloupcový graf je možné podle

potřeby přibližovat a oddalovat. Vždy se lze vrátit na původní měřítko stisknutím tlačítka *Původní měřítko*. Všechny tři typy grafů lze vidět na Obrázek 21.



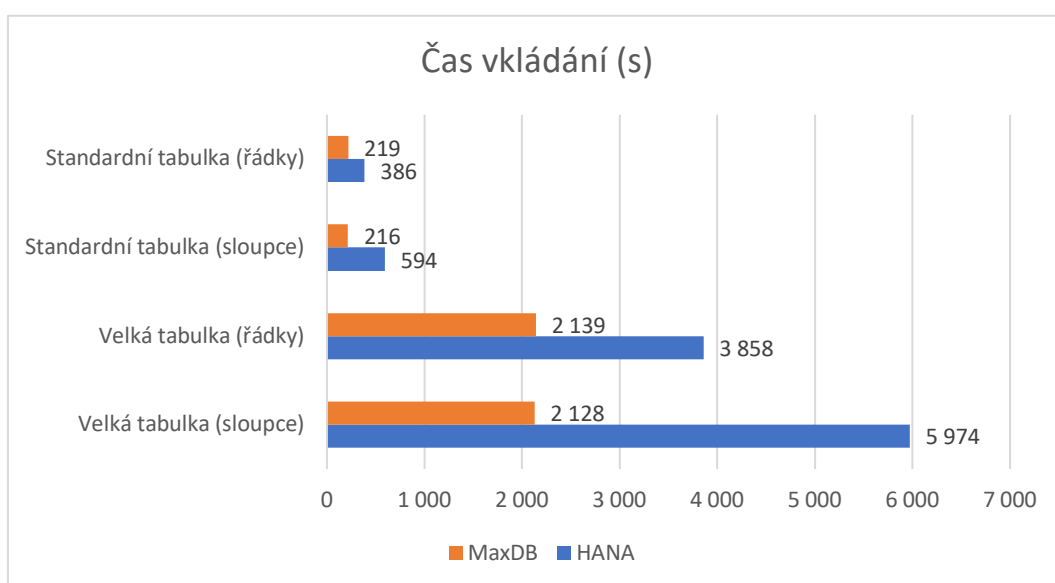
Obrázek 21 Vykreslené grafy na základě dotazů

4.2 Výsledky měření

Způsob měření vycházel z modelu testování popsaného v kapitole 3.7. Nejprve bylo změřeno, jak dlouho trvá vkládání dat do tabulek ze zdrojových souborů datasetu. Poté bylo využitím administračního nástroje zjištěno, kolik fyzického místa na disku (MaxDB) nebo v operační paměti (HANA) tabulky zabírají. Následně již proběhlo testování odezvy databází na vybrané dotazy, které budou podrobněji popsány. Operace byly provedeny na všech vytvořených tabulkách: *ZVSMETEO_ROW*, *ZVSMETEO_COL*, *ZVSMETEOXL_ROW* a *ZVSMETEOXL_COL*. Stejně tak reportovací aplikace byla spuštěna na obou databázích. Všechny výsledky jsou aritmetickým průměrem 5 provedených měření.

4.2.1 Výsledky vkládání

Import dat ze zdrojových souborů do databáze byl časově nejnáročnější. Plnění velké tabulky s 6 miliony záznamů trvalo i více jak 2 hodiny. Naměřené výsledky byly uloženy do speciální tabulky *ZVSINSTIMES*. Zjištěné výsledky můžeme vidět v Graf 1. V první řadě je překvapující, že tradiční MaxDB zvládá import dat do tabulek za poloviční čas než databáze HANA. Podle předpokladů MaxDB databáze nerozlišuje mezi řádkově a sloupcově orientovanými tabulkami, všechny ukládá po řádcích, časy jsou tak prakticky stejné.



Graf 1 Čas vkládání

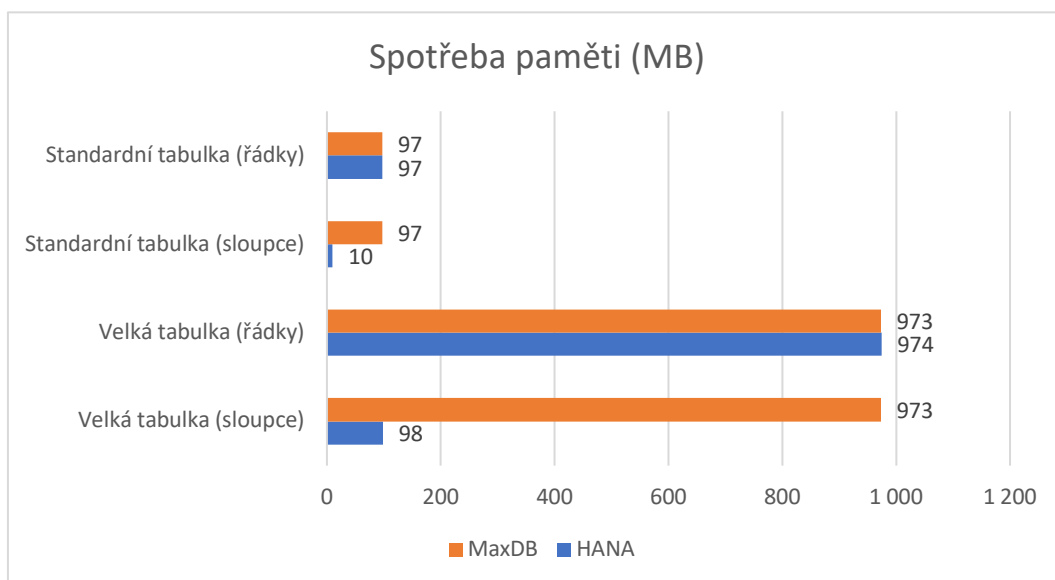
U databáze HANA můžeme pozorovat, že vkládání do řádkově orientovaných tabulek je průměrně o 35 % rychlejší než vkládání do sloupcových tabulek. Zřejmě je to z důvodu použití slovníkového kódování, které vyžaduje poměrně náročnou režii. Pro každý atribut musí být vytvořen slovník a vektor atributu.

Rozdíl jednoho řádu v objemu dat nemá žádný viditelný vliv na čas zpracování. Velké tabulky s 6 miliony záznamů tak trvá naplnit 10krát déle než standardní tabulky s 600 tisíci záznamů.

Tradiční MaxDB databáze je tedy pro vkládání záznamů výhodnější než in-memory databáze HANA. Otázkou je, jak velkou roli hrají mírně odlišný výkon HW, který byl specifikován v kapitole 3.5.1.

4.2.2 Spotřeba paměti

Poté, co byla do obou databází vložena všechna data, byly s pomocí administračního nástroje systému SAP zjištěny informace o skutečné velikosti jednotlivých tabulek. Jinými slovy, kolik fyzického místa na pevném disku v případě MaxDB a operační paměti u databáze HANA zabírají. Jde nám o to, abychom ověřili efektivitu komprimační techniky slovníkového kódování, aplikované na tabulky se sloupcově orientovaným uspořádáním dat.



Graf 2 Spotřeba paměti

V Graf 2 vidíme, že tabulky s řádkovým spořádáním zabírají stejně místa v tradiční i in-memory databázi. Avšak pokud by testovací data obsahovala záznamy, ve kterých by měly některé atributy prázdnou hodnotu (hodnota NULL), v in-memory databázi by tabulka zabírala méně prostoru než v tradiční databázi. MaxDB databáze totiž automaticky rezervuje místo pro celé záznamy včetně prázdných hodnot, což u in-memory databáze HANA neplatí. Tam jsou platné hodnoty uloženy přesně za sebou a prázdné hodnoty jsou úplně vynechány.

Dále můžeme pozorovat, že byl potvrzen předpoklad efektivní komprimace sloupcově uspořádaných tabulek pomocí slovníkového kódování. Komprimované tabulky tak spotřebovávají o zhruba 90 % méně místa v paměti. Slovníkové kódování zde těží ze skutečnosti, že v jednotlivých attributech existuje pouze omezený počet unikátních hodnot. Hodnoty tedy mají poměrně nízkou kardinalitu. Tabulky se sloupcově orientovaným uspořádáním dat tak ukazují ve spojení se slovníkovým kódováním svou silnou stránku z hlediska nízké spotřeby paměti.

Co se týče možné úspory paměti s narůstajícím objem dat, které jsou zakódované slovníkovým kódováním, v našem případě se ukázala jen malá úspora 2 MB u velké tabulky s 6 miliony záznamy oproti standardní s 600 tisíci záznamy. K dosažení větší úspory by byl zřejmě potřeba ještě daleko vyšší objem dat nebo nižší kardinalita hodnot.

4.2.3 Testovací dotazy

Závěrečnou a nejdůležitější částí měření výkonu obou databází bylo 7 vybraných analytických dotazů. První dva jsou zaměřeny na rekonstrukci celých záznamů (*příkaz SELECT **) a další dvojice pak na výběr pouze požadovaných atributů. Vždy je vyzkoušena varianta bez omezující podmínky a s její přítomností (predikát *WHERE*) z důvodu sledování vlivu míry restrikce na rychlost zpracování dotazu. Druhou část dotazů tvoří agregační funkce. V prvních dvou agregačních dotazech jsou použity funkce pro výpočet denního průměru a maximální hodnoty, liší se použitím omezující podmínky (predikát *HAVING*). Třetí agregační dotaz, a poslední celkově, je zaměřen na funkci počtu záznamů v jednotlivých agregovaných skupinách.

Předpokládáme, že ve všech dotazech by měla být rychlejší in-memory databáze HANA na úkor klasické databáze MaxDB, jelikož u prvně jmenované jsou všechna data uložena

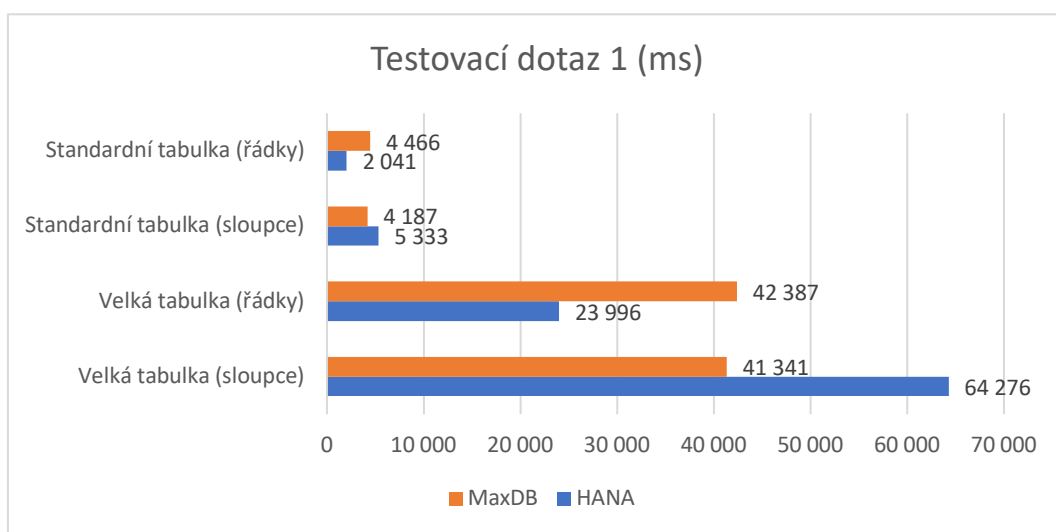
v operační paměti. Dále očekáváme, že v rámci in-memory databáze bude první dotaz s rekonstrukcí celých záznamů zpracován rychleji na tabulkách s řádkově orientovaným uspořádáním dat. U druhého dotazu s výběrem atributů (částečná rekonstrukce záznamu) je pak předpoklad nejasný. Pro řádkové uspořádání hraje rekonstrukce části záznamu, pro sloupcové zase manipulace jen s pár atributy. U agregačních dotazů pak v rychlosti zpracování předpokládáme jasnou převahu na straně sloupcového uspořádání dat v tabulce.

4.2.3.1 Testovací dotaz 1

První testovací dotaz zjišťuje vývoj vnitřní a venkovní teploty za celé dostupné časové období (zhruba 5 let). Aby byl otestován výkon databází při rekonstrukci celých záznamů, byl použit příkaz *SELECT **. Příkaz v jazyce SQL pro provedení tohoto testovacího dotazu vypadá následovně:

```
SELECT *
FROM zvsmeteo;
```

Výsledky naměřeného výkonu obou databází na všech vytvořených tabulkách jsou ukázány v Graf 3. Je překvapující, že databáze MaxDB vrací výsledek podstatně rychleji než databáze HANA pracující se sloupcově orientovanou tabulkou. MaxDB zde zcela stáhla rozdíl mezi rychlostí pevného disku a operační paměti. Na druhou stranu je prakticky o polovinu pomalejší ve srovnání s řádkově orientovanou tabulkou v databázi HANA.



Graf 3 Testovací dotaz 1 – rekonstrukce celých záznamů

Dále můžeme vidět, že při porovnání řádkově a sloupcově orientovaných tabulek v in-memory databázi HANA jasně lépe vychází užití řádkového schématu. Zpracování dotazu na tomto schématu je téměř 3krát rychlejší.

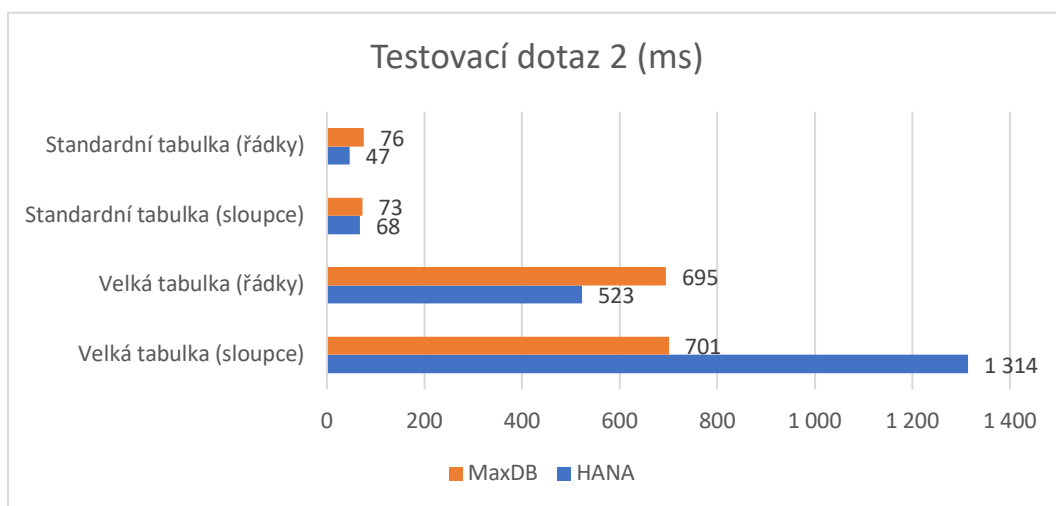
V případě rekonstrukce celých záznamů tedy tradiční databáze MaxDB obstála. V databázi HANA hraje velkou roli typ uspořádání dat, kde sloupcové je výrazně pomalejší než řádkové.

4.2.3.2 Testovací dotaz 2

Další testovací dotaz je velmi podobný tomu prvnímu, ale navíc přidává omezení výsledných záznamů pomocí predikátu *WHERE*. Tento dotaz tedy zjišťuje vývoj vnitřní a venkovní teploty v omezeném časovém období 1. dubna 2012 až 30. dubna 2012. Příkaz *SELECT* má následující podobu:

```
SELECT *  
FROM zvsmeteo  
WHERE datum BETWEEN `20120401` AND `20120430`;
```

Graf 4 ukazuje naměřené časy zpracování tohoto dotazu. Poměrově jsou hodnoty podobné prvnímu testovacímu dotazu. MaxDB databáze je tedy výkonnostně mezi odezvami sloupcového a řádkového uspořádání v in-memory databázi HANA.



Graf 4 Testovací dotaz 2 – rekonstrukce celých záznamů s omezující podmínkou

Zpracování tabulky se sloupcovým schématem v in-memory je opět výrazně pomalejší než v případě řádkového schématu. Avšak vzhledem k dosavadním výsledkům je překvapující, že u tabulky standardní velikosti je sloupcově uspořádaná tabulka o trochu

rychlejší než standardní tabulka v MaxDB. Dostáváme tedy rozdílný výsledek v závislosti na objemu dat. Můžeme odvodit, že čím větší objem dat je zpracováván, tím víc je sloupcové uspořádání nevýhodné.

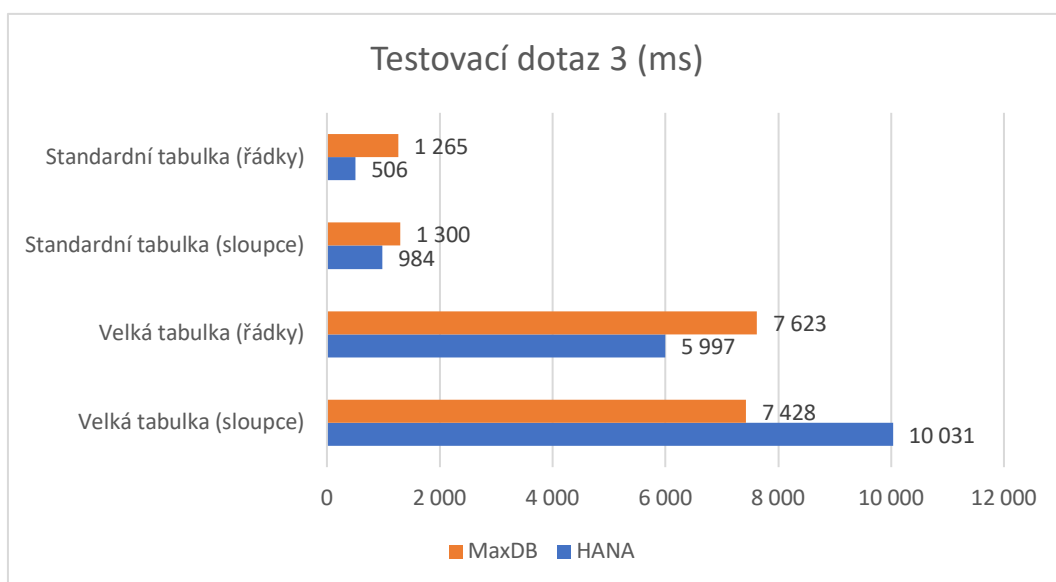
Zároveň pozorujeme, že s omezující podmínkou jsou všechny odezvy o dva řády rychlejší, protože systém prováděl rekonstrukci pouze zlomku záznamů ve srovnání s prvním testovacím dotazem. Použití predikátu *WHERE* má tedy obecně velký vliv na rychlost odezvy.

4.2.3.3 Testovací dotaz 3

U třetího testovacího dotazu jsou již v příkazu *SELECT* přímo specifikovány vyžadované atributy, takže databázový systém nemusí provádět rekonstrukci celého záznamu, ale pouze jeho malé části, což by mělo být výhodné pro sloupcově orientované uspořádání dat. Tentokrát dotaz zjišťuje vývoj naměřené hodnoty tlaku za celé dostupné období. Příkaz *SELECT* vypadá následovně:

```
SELECT datum cas tlak  
FROM zvsmeteo;
```

Naměřené časy zpracování tohoto dotazu je možné vidět v Graf 5. I když se zdálo, že práce jen s pár atributy bude příhodná pro sloupcové schéma, výsledky jsou velice podobné předchozím dotazům.



Graf 5 Testovací dotaz 3 – částečná rekonstrukce záznamů

Opět platí, že čím větší objem dat je zpracováván, tím je použití sloupcového schématu pomalejší. To zde platí také pro in-memory databázi HANA, jelikož na standardní tabulce s 600 tisíci záznamy je o polovinu rychlejší než MaxDB, ale v případě dotazu nad velkou tabulkou s 6 miliony záznamy už rozdíl není tak velký. Navíc jak již bylo zmíněno, sloupcové schéma má u velké tabulky nejpomalejší odezvu.

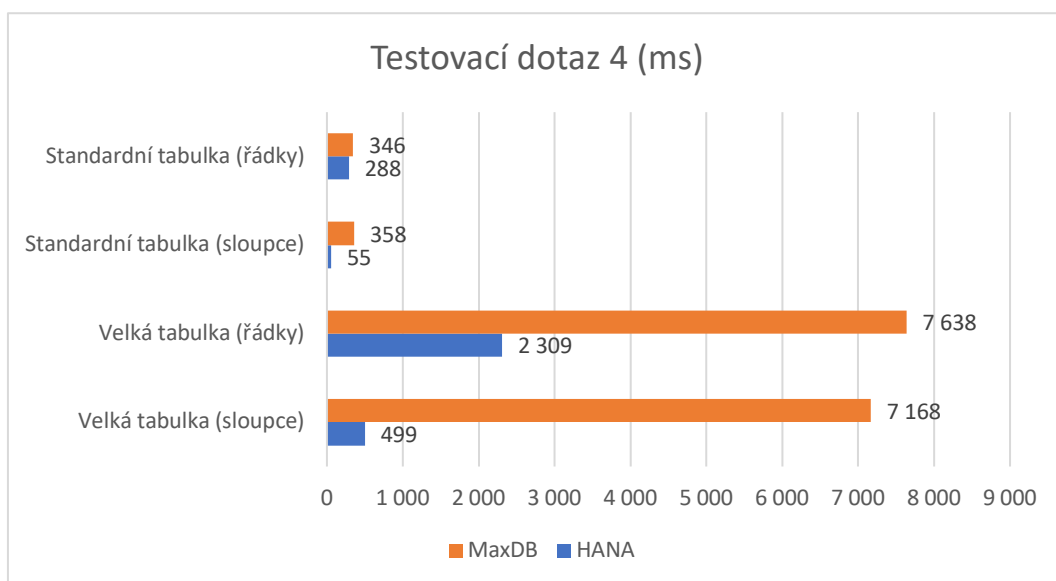
Klasická databáze MaxDB má tedy velmi srovnatelný výkon s in-memory databází, pokud jde o rekonstrukci celých nebo částí záznamů. To je překvapivé zjištění.

4.2.3.4 Testovací dotaz 4

Čtvrtý testovací dotaz kromě pouze některých atributů přidává stejně jako ve druhém dotazu omezující podmínku predikátu *WHERE*. Opět tak bude docházet k rekonstrukci částí záznamů, proto nás bude zajímat, zda bude sloupcově orientované uspořádání dat používané v in-memory databázi nadále vykazovat překvapivě pomalou odezvu, a jestli výkon MaxDB databáze bude stále srovnatelný s databází HANA.

Cílem dotazu je zjistit vývoj hodnoty tlaku za celé dostupné časové období, ale vybrány budou pouze ty záznamy, které mají hodnotu tlaku mezi 1000 a 1005 hPa. Tomuto odpovídá následující příkaz *SELECT*:

```
SELECT datum cas tlak
FROM zvsmeteo
WHERE tlak BETWEEN 1000 AND 1005;
```



Graf 6 Testovací dotaz 4 – částečná rekonstrukce záznamů s omezující podmínkou

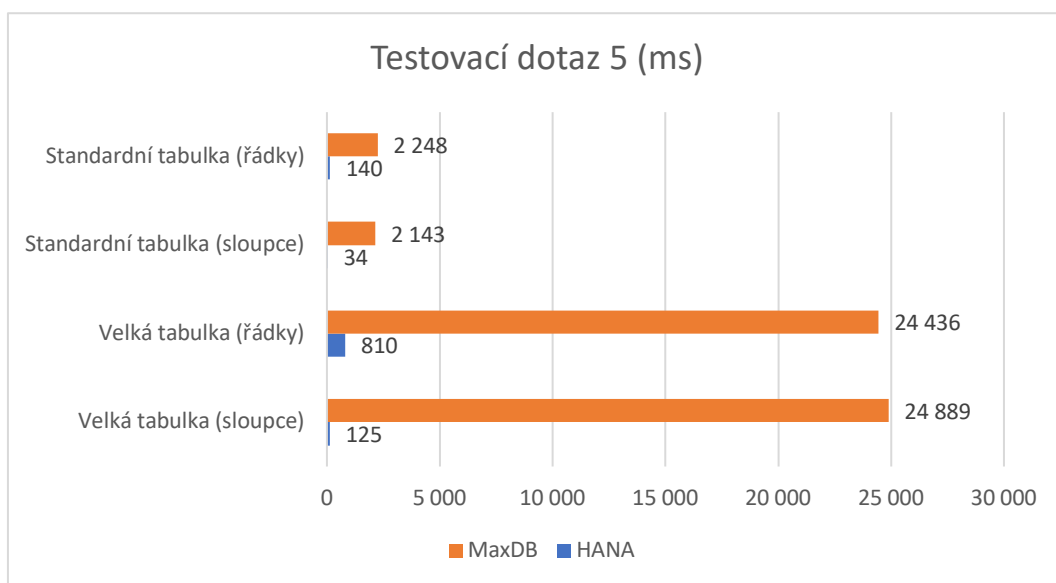
V Graf 6 můžeme vidět, že tentokrát jsou výsledky zcela rozdílné. MaxDB databáze je zřetelně pomalejší než oba dva typy uspořádání dat v in-memory databázi HANA. U tabulky standardní velikosti je vidět znatelné zrychlení zejména u sloupcového schématu, ale největší rozdíly jsou pak dosaženy v případě velkých tabulek s miliony záznamy. Odezva v in-memory databázi se sloupcově uloženými daty trvá pouze 7 % času potřebného ke zpracování stejného dotazu v MaxDB. Pouhé použití omezující podmínky a současného výběru jen některých atributů má tedy za následek mnohem rychlejší odezvu in-memory databáze.

4.2.3.5 Testovací dotaz 5

Nyní se již dostáváme k dotazům, které v sobě obsahují agregační funkce. Jak již bylo několikrát zmíněno v teoretické části této práce, sloupcově orientované uspořádání dat by mělo poskytnout nejlepší odezvu právě na agregačních operacích a celkově u in-memory databáze HANA by se měly projevit veškeré výhody této technologie.

Pátý testovací dotaz používá agregační funkci *AVG*, tedy aritmetický průměr. Jeho úkolem je zjistit průměrnou denní rychlost větru za celé dostupné časové období. Příkaz *SELECT* vypadá následovně:

```
SELECT datum AVG(rychlost_vetru)
FROM zvsmeteo
GROUP BY datum;
```



Graf 7 Testovací dotaz 5 – agregační funkce AVG

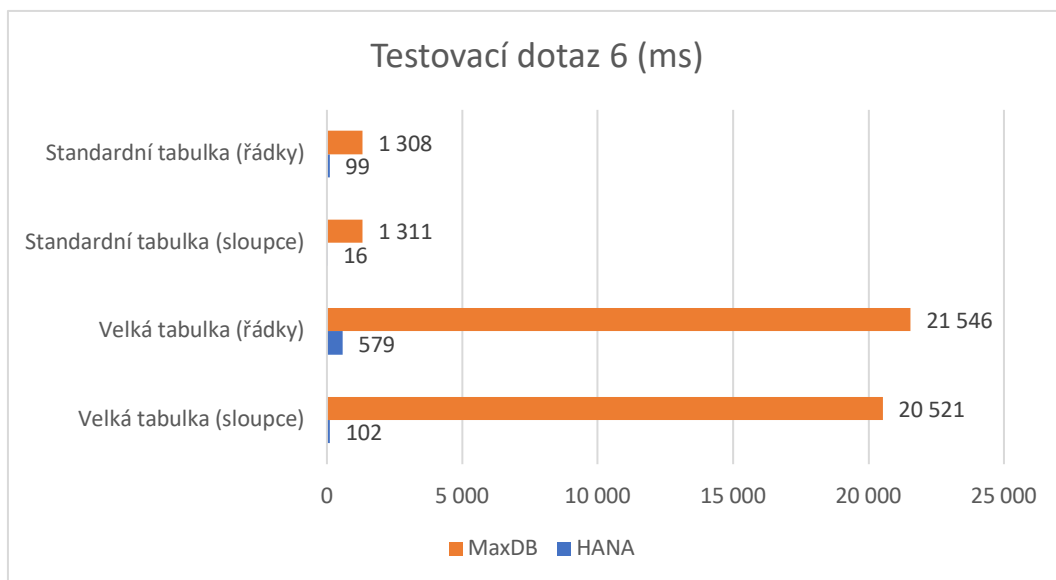
Při pohledu na výsledky měření, viz Graf 7, je zřejmé, že v případě agregačních dotazů je in-memory databáze HANA mnohonásobně rychlejší než MaxDB. Opět platí, že s rostoucím objemem zpracovávaných dat roste výhoda in-memory databáze. V případě velké tabulky s miliony záznamů a sloupcovým uspořádáním dat trvá odezva databáze HANA asi 0,5 % času, který na provedení stejné operace potřebuje databáze MaxDB.

Z hlediska srovnání obou typů uspořádání dat již jednoznačně vítězí sloupcové schéma, které je v uvedené agregační funkci několikanásobně rychlejší.

4.2.3.6 Testovací dotaz 6

Předposlední testovací dotaz přidává omezující podmínku. Ta je u agregačních funkcí specifikována pomocí predikátu *HAVING*. V tomto případě je použita agregační funkce *MAX*, tedy maximální hodnota zvoleného atributu seskupených záznamů. Testovací dotaz 6 zjišťuje maximální denní rychlost větru za celé dostupné období, s omezením na rychlost větru 0 až 2 m/s. Příkaz *SELECT* je následující:

```
SELECT datum MAX(rychlost_vetru)
FROM zvsmeteo
GROUP BY datum
HAVING MAX(rychlost_vetru) BETWEEN 0 AND 2;
```



Graf 8 Testovací dotaz 6 – agregační funkce MAX s omezující podmínkou

Z naměřených hodnot v Graf 8 můžeme vyčíst, že přidání omezující podmínky k agregační funkci má jen minimální vliv na rychlost zpracování v databázi. Ve všech

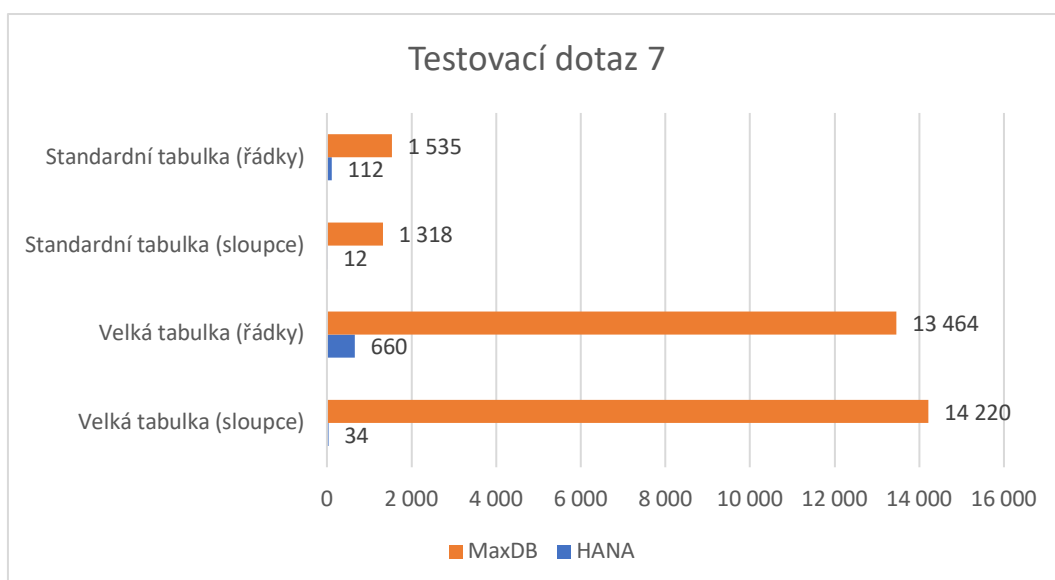
případech došlo pouze k mírnému zrychlení, a to zřejmě díky menšímu počtu výsledných záznamů.

Dále tedy platí, že in-memory databáze HANA podává v agregačních operacích mnohem vyšší výkon než tradiční databáze MaxDB a sloupcově orientované uspořádání dat potvrzuje svou výhodu při operaci s jednotlivými atributy. Zatímco v databázi MaxDB trvá zpracování agregačního dotazu na velké tabulce kolem 20 sekund, v in-memory databázi je to otázka desetin sekundy. I při vyšších objemech dat (desítky či stovky milionů) je tak možné dosáhnout výpočtu agregací v reálném čase, např. v interaktivních aplikacích.

4.2.3.7 Testovací dotaz 7

Poslední testovací dotaz je zaměřen na agregační funkci *COUNT*, která vrací počet záznamů v každé agregované skupině. Tento dotaz má za cíl zjistit četnost jednotlivých směrů, kterými vanul vítr v celém dostupném časovém období. Celkem je rozlišováno 16 různých směrů včetně základních světových stran. Příkaz *SELECT* má následující podobu:

```
SELECT smer_vetru_num smer_vetru_text COUNT(*)  
FROM zvsmeteo  
GROUP BY smer_vetru_num smer_vetru_text;
```



Graf 9 Testovací dotaz 7 – agregační funkce COUNT

V Graf 9 můžeme pozorovat, že u agregační funkce *COUNT* je rozdíl mezi MaxDB a HANA databázemi ještě propastnější. In-memory databáze dokáže zpracovat stejný agregační dotaz za zlomek procenta času, který jinak potřebuje tradiční databáze.

Zajímavostí je nepoměr mezi časem zpracování standardních a velkých tabulek v in-memory databázi. I když má velká tabulka 10krát více záznamů než standardní, odezva je u velké tabulky 6krát pomalejší v případě řádkového schématu, a dokonce pouze 3krát pomalejší u sloupcového schématu uložení dat. Naopak u databáze MaxDB je zachován rozdíl jednoho řádu.

V rámci in-memory databáze se zde projevuje síla sloupcově orientovaného uspořádání dat. I když výkon řádkového uspořádání je rovněž velmi slušný, u databázi velkých firem, které mohou obsahovat i miliardy záznamů, ob stojí už pouze sloupcové schéma.

4.3 Zhodnocení výsledků

Na základě naměřených výsledků můžeme říci, že vkládání dat do tradiční databáze MaxDB je překvapivě rychlejší než u nové in-memory technologie. Je to zřejmě způsobeno tím, že in-memory databáze HANA se snaží ušetřit co nejvíce místa v poměrně stále drahé operační paměti. U řádkového uspořádání hledá prázdné hodnoty, pro které nerezervuje v paměti místo. U sloupcového pak používá komprimační metodu slovníkového kódování, které představuje při vkládání nových záznamů docela velkou režii. Proto dochází ke zpomalení oproti řádkovému uspořádání. Mezi objemem dat a rychlostí jejich vkládání do tabulek je zachována přímá úměra. Otázkou je, jakou měrou zde hraje roli mírně rozdílná hardwarová konfigurace obou používaných systémů.

Z hlediska spotřeby paměti si pak stojí velmi podobně databáze MaxDB a řádkově uspořádaná databáze HANA. V obou případech jsou záznamy uloženy za sebou po řádcích. Rozdíl by nastal, jen pokud testovací data obsahovala prázdné hodnoty, z čehož by těžila in-memory databáze. Na rozdíl od rychlosti vkládání zde sloupcové schéma velmi těží ze slovníkového kódování. Tabulky s tímto uspořádáním zabírají pouze desetinu prostoru, než kdyby byly uloženy po řádcích. Velkou roli v úspoře paměti hraje kardinalita hodnot jednotlivých atributů. Při nízké kardinalitě bychom s narůstajícím objemem dat mohli dosáhnout ještě vyšší míry komprese.

U dotazů, které vyžadují rekonstrukci celých záznamů příkazem *SELECT **, je výkon databáze MaxDB ve srovnání s in-memory databází HANA někde uprostřed. Jasně nejpomalejší je sloupcové uspořádání dat, které s přibývajícím objemem dat dále zvýrazňuje svou nevýhodu při tomto typu operace. Tímto se tedy potvrzuje doporučení zmíněné v kapitole 2.2.2.2 ohledně vyhnutí se použití příkazu *SELECT ** ve sloupcové in-memory databázi. MaxDB patrně dosahuje takto dobrých výsledků díky efektivnímu mechanismu vyrovnávacích pamětí.

Nejvyrovnanější výsledky byly obdrženy při zpracování dotazu, který požadoval jen několik málo atributů. Tento výsledek úplně neodpovídá předpokladům, že in-memory databáze, a především sloupcové uspořádání dat naplno ukáží svou výhodu. Situace je však zcela odlišná při použití omezující podmínky. V tomto případě je již in-memory databáze několikanásobně rychlejší než MaxDB, a zpracování dotazu na sloupcově uspořádané tabulce je dokonce více jak 10krát rychlejší.

U agregačních dotazů se naplno projevuje síla in-memory databáze SAP HANA, která ve srovnání s databází MaxDB dosahuje odezvy ve zlomcích procenta. Dále platí, že čím větší objem dat je zpracováván, tím vyšší je rozdíl v rychlosti obou databází u agregačních operací. Jelikož je tento typ operací velmi často používáný v analytickém provozu ve stylu OLAP, velké firmy s databázemi o milionech až miliardách záznamů mohou z výkonu in-memory databáze značně profitovat. Výsledky analytických operací jsou dostupné v reálném čase na vyžádání, a nemusí tak být předem vypočítávány na neaktuálních datech.

ZÁVĚR

Až dosud jsou ve velkých firmách používány tradiční relační databáze, založené na řádkově orientovaném uspořádání dat. Zároveň je často databáze rozdělena na dva systémy, OLTP a OLAP, které jsou optimalizovány a slouží výhradně pro transakční, respektive analytické operace. Data jsou z OLTP systému replikována do OLAP systému pomocí procesu ETL, který je poměrně náročný. Mezi tradiční databáze patří i MaxDB od firmy SAP.

Vývoj hardwaru v posledním desetiletí inspiroval profesora Hasso Plattnera k představení in-memory databáze SAP HANA, která ukládá všechna data přímo v rychlé operační paměti. Ta již není tak drahá jako v minulosti a nabízí větší kapacity. Navíc spojuje OLTP a OLAP systémy v jeden unifikovaný systém, což přináší vyšší flexibilitu a analýzu dat vždy na aktuálních datech. ETL proces tak již není potřeba.

In-memory databáze SAP HANA primárně používá sloupcově orientované uspořádání dat, takže záznamy nejsou uloženy po řádcích, ale po jednotlivých sloupcích – attributech. Zároveň na data aplikuje komprimační metodu slovníkového kódování, která snižuje spotřebu omezené operační paměti a vytváří celočíselnou reprezentaci všech hodnot, takže v některých případech může výrazně urychlit zpracování analytických dotazů, především agregačních funkcí. Tento typ uspořádání dat je tak dobře optimalizován pro operace čtení, ale představuje zpomalení při vkládání nových hodnot, aktualizaci nebo smazání záznamů. Databáze HANA slibuje přínos zejména v analytickém provozu, který je charakteristický pro řešení Business Intelligence.

Za účelem ověření slibovaných výhod in-memory databáze nad tradiční databází byla v rámci této práce vytvořena reportovací aplikace v prostředí SAP R/3, která je schopna provozu na obou databázích a nabízí reporting meteorologických veličin, změřených za posledních 5 let na meteorologické stanici brněnské hvězdárny. Při běhu této aplikace jsme tak zjistili rychlost odezvy jednak při vkládání dat do databáze a také při zpracování vybraných typů dotazů.

Na základě naměřených výsledků v reportovací aplikaci bylo možné lépe porovnat databáze MaxDB a HANA. Bylo zjištěno, že rychlost vkládání dat do tabulek je srovnatelná v obou databázích. Jako nejpomalejší se ukázalo sloupcové schéma v in-

memory databázi, které však bylo na druhou stranu jasně nejúspornější z hlediska spotřeby paměti díky slovníkovému kódování. MaxDB velmi dobře obstála i u dotazů, které vyžadovaly rekonstrukci celých záznamů. Můžeme předpokládat, že v těchto případech dobře zafungoval mechanismus vyrovnávacích pamětí a in-memory databáze tak navzdory předpokladům nebyla rychlejší. Zásadní zlom přinesl dotaz s výběrem pouze několika atributů a omezující podmínkou. Databáze HANA již byla několikanásobně rychlejší, a to zejména v případě sloupcově orientovaného uspořádání dat. Ještě propastnější rozdíly v rychlosti odezvy pak nastaly u dotazů s agregačními funkcemi. Sloupcově orientovaná in-memory databáze zvládla stejnou operaci za zlomek procenta času, než který potřebovala databáze MaxDB. Zároveň byl zjištěn vztah mezi zvyšujícím se objemem dat a rychlostí zpracování. Čím větší databáze, tím větší výhodu sloupcově orientovaná in-memory databáze HANA získává nad tradiční MaxDB.

Můžeme dojít k závěru, že u velkých firem, které provozují databáze s tabulkami disponujícími miliony až miliardami záznamů, může in-memory databáze přinést velmi výrazné zrychlení analytických operací, např. v rámci řešení BI, a umožnit jejich zpracování v reálném čase na vyžádání. U menších databází je však přínos diskutabilní a zřejmě by nevykompenzoval vysoké náklady na zavedení in-memory databáze. Co se týče testovacích meteorologických dat použitých v reportovací aplikaci, mohly by z nasazení in-memory databáze čerpat např. zemědělské společnosti, a lépe tak plánovat a předvídat události spojené s vývojem počasí a jeho vlivem na zemědělskou půdu.

SEZNAM POUŽITÝCH ZDROJŮ

ABADI, D. J., D. S. Mayers, D. J. DeWitt and S. R. Madden, 2007. Materialization Strategies in a Column-Oriented DBMS. In: *2007 IEEE 23rd International Conference on Data Engineering* [online]. Istanbul: IEEE [cit. 2018-05-01]. ISSN 1063-6382. Dostupné z: databáze IEEE Xplore Digital Library.

ABADI, D. J., S. R. Madden and N. Hachem, 2008. Column-stores vs. row-stores: how different are they really?. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* [online]. New York: ACM. p. 967-980 [cit. 2018-05-01]. ISBN 978-1-60558-102-6. Dostupné z: databáze ACM Digital Library.

Datahub, 2017. *Hvězdárna – meteostanice* [online]. Brno: Statutární město Brno [cit. 2018-05-01]. Dostupné z: <https://old.datahub.io/dataset/hvezdarna-meteostanice>.

FÄRBER, F., S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg and W. Lehner, 2012. SAP HANA Database: Data Management for Modern Business Applications. *SIGMOD Rec.* [online]. vol. 40, no. 4, p. 45-51 [cit. 2018-05-01]. ISSN 0163-5808. Dostupné z: databáze ACM Digital Library.

FÄRBER, F., N. May, W. Lehner, P. Große, I. Müller, H. Rauhe and J. Dees, 2012. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.* [online]. vol. 35, no. 1, p. 28-33 [cit. 2018-05-01]. Dostupné z: <http://sites.computer.org/debull/A12mar/hana.pdf>.

KIFER, M., A. Bernstein and P. M. Lewis, 2005. *An application-oriented approach: Introductory version* [online]. Boston: Pearson/Addison Wesley [cit. 2018-05-01]. Dostupné z: <http://wps.aw.com/wps/media/objects/1230/1260320/PracticeProblemsandSolutions.pdf>

KRUEGER, J., C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey and A. Zeier, 2011. Fast Updates on Read-optimized Databases Using Multi-core CPUs. *Proceedings of the VLDB Endowment* [online]. vol. 5, no. 1, p. 61-72 [cit. 2018-05-01]. ISSN 2150-8097. Dostupné z: databáze ACM Digital Library.

KÜHNHAUSER, K.-H., 2009. *ABAP: výukový kurz*. 1. vyd. Brno: Computer Press. 368 s. ISBN 978-80-251-2117-7.

LABERGE, R. *Datové sklady – Agilní metody a business intelligence*. Praha: Computer Press, 2012. 352 s. ISBN 978-80-251-3729-1.

LEE, J., M. Muehle, N. May, F. Färber, V. Sikka, H. Plattner, J. Krueger and M. Grund, 2013. High-Performance Transaction Processing in SAP HANA. *IEEE Data Eng. Bull.* [online]. vol. 36, no. 2, p. 28-33 [cit. 2018-05-01]. Dostupné z: <ftp://131.107.65.22/pub/debull/A13june/A13Jun-CD.pdf#page=30>.

MALIK, B., 2018. SAP Community WIKI. In: *About SAP MaxDB* [online]. Waldorf: SAP [cit. 2018-05-01]. Dostupné z: <https://wiki.scn.sap.com/wiki/display/MaxDB/About+SAP+MaxDB>

MALIK, B., 2018. SAP Community WIKI. In: *SAP MaxDB Components* [online]. Waldorf: SAP [cit. 2018-05-01]. Dostupné z: <https://wiki.scn.sap.com/wiki/display/MaxDB/SAP+MaxDB+Components>

MÜLLER, S. and H. Plattner, 2015. Aggregates caching in columnar in-memory databases. In: *In Memory Data Management and Analysis* [online] Potsdam: Springer. p. 69-81 [cit. 2018-05-01]. ISBN 978-3-319-13960-9. Dostupné z: databáze Springerlink.

NASCIMENTO, L., 2016. SAP Blogs. In: *Introduction to SAP Business Suite 4 SAP HANA* [online]. Waldorf: SAP [cit. 2018-05-01]. Dostupné z: <https://blogs.sap.com/2016/07/04/introduction-to-sap-business-suite-4-sap-hana/>

NOVOTNÝ, O., J. POUR a D. SLÁNSKÝ, 2005. *Business Intelligence: jak využít bohatství ve vašich datech*. 1. vyd. Praha: Grada. 256 s. ISBN 80-247-1094-3.

PLATTNER, H., 2009. A common database approach for OLTP and OLAP using an in-memory column database. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data* [online]. Providence: ACM. p. 1-2 [cit. 2018-05-01]. ISBN 978-1-60558-551-2. Dostupné z: databáze ACM Digital Library.

PLATTNER, H., 2013. *A Course in In-Memory Data Management*. 1st ed. Potsdam: Springer. 297 p. ISBN 978-3-642-44314-5.

PLATTNER, H., 2014. The impact of columnar in-memory databases on enterprise systems: implications of eliminating transaction-maintained aggregates. *Proceedings of the VLDB Endowment* [online]. vol. 7, no. 13, p. 1722-1729 [cit. 2018-05-01]. ISSN 2150-8097. Dostupné z: databáze ACM Digital Library.

SIKKA, V., F. Färber, W. Lehner, S. K. Cha, T. Peh and C. Bornhövd, 2012. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* [online]. New York: ACM. p. 731-742 [cit. 2018-05-01]. ISBN 978-1-4503-1247-9. Dostupné z: databáze ACM Digital Library.

SEZNAM POUŽITÝCH OBRÁZKŮ

Obrázek 1 Hierarchie paměti (Zdroj: Plattner, 2013, s. 24)	19
Obrázek 2 Vícevrstvá architektura databáze SAP HANA (Zdroj: Sikka aj., 2012).....	24
Obrázek 3 Unifikovaná tabulka (Zdroj: Sikka aj., 2012)	27
Obrázek 4 Částečné slučování (Zdroj: Sikka aj., 2012).....	29
Obrázek 5 Řádkové uspořádání dat v paměti (Zdroj: Plattner, 2013, s. 60)	30
Obrázek 6 Sloupcové uspořádání dat v paměti (Zdroj: Plattner, 2013, s. 60).....	31
Obrázek 7 Slovníkové kódování (Zdroj: Plattner, 2013, s. 38).....	34
Obrázek 8 Prefixové kódování (Zdroj: Plattner, 2013, s. 44).....	36
Obrázek 9 Operace vkládání (Zdroj: Plattner, 2013, s. 78-79).....	40
Obrázek 10 Exekuční plán operace čtení (Zdroj: Plattner, 2013, s. 101)	41
Obrázek 11 Agregční funkce COUNT (Zdroj: Plattner, 2013, s. 142)	43
Obrázek 12 Obecná koncepce architektury BI (Zdroj: Vlastní zpracování dle: Novotný aj., 2005).....	48
Obrázek 13 Komponenty SAP S/4 HANA (Zdroj: Vlastní zpracování dle: Nascimento, 2016)	51
Obrázek 14 Vývojové prostředí ABAP Workbench	53
Obrázek 15 Schéma testovací databáze	55
Obrázek 16 Definice databázové tabulky ZVSMETEO	58
Obrázek 17 Přehled databázových tabulek.....	59
Obrázek 18 Správa databáze.....	60
Obrázek 19 Konfigurace dotazů	61
Obrázek 20 Přehled provedených dotazů.....	62
Obrázek 21 Vykreslené grafy na základě dotazů.....	63

SEZNAM POUŽITÝCH GRAFŮ

Graf 1 Čas vkládání.....	64
Graf 2 Spotřeba paměti	65
Graf 3 Testovací dotaz 1 – rekonstrukce celých záznamů	67
Graf 4 Testovací dotaz 2 – rekonstrukce celých záznamů s omezující podmínkou	68
Graf 5 Testovací dotaz 3 – částečná rekonstrukce záznamů	69
Graf 6 Testovací dotaz 4 – částečná rekonstrukce záznamů s omezující podmínkou	70
Graf 7 Testovací dotaz 5 – agregační funkce AVG.....	71
Graf 8 Testovací dotaz 6 – agregační funkce MAX s omezující podmínkou	72
Graf 9 Testovací dotaz 7 – agregační funkce COUNT	73