Školy Březová – střední odborná škola, základní škola a mateřská škola, Březová

Systém řízení báze dat pro použití v koncově uživatelských programech

Lukáš Snížek

Vedoucí práce: Martin Křížan, DiS.

Maturitní práce
Praha 2025

Prohlašuji, že jsem maturitní práci vypracoval/a samostatně a na základě literatury a pramenů uvedených v Seznamu použité literatury. Prohlašuji, že odevzdaná verze dokumentace maturitní práce a verze elektronická jsou totožné.

V Praze dne 31. 3. 2025

.....

podpis

Vedoucí maturitní práce: Martin Křížan, DiS.

V ..... dne ....

podpis

Děkuji svému vedoucímu maturitní práce, Martinu Křížanovi, DiS., za cenné rady a připomínky při konzultacích a tvorbě této práce.

Abstrakt

Tato maturitní práce se zabývá systémy řízení bází dat (SŘBD) určených pro použití koncově uživatelskými programy. V teoretické části práce je rozebráno, co vlastně systém řízení báze dat je, jeho části, datastruktury, které umožnují jeho činnost a již existující SŘBD.

V praktické části je jeden takový systém, „matdb“, implementován. Je použito programovacího jazyku Rust. SŘBD obsahuje dotazový optimizátor, který je založen na e-grafové knihovně egg. SŘBD podporuje dotazy napříč časem (verzemi databáze), k čemuž využívá persistentní B-stromové úložiště. Pro zjednodušení správy je toto B--stromové úložiště, stejně jako veškerá data, zcela obsaženo v jedinném souboru.

Matdb je možné použít jako knihovnu v jiném programu, či jako samostatný program na příkazovém řádku.

Obsah

Úvod

Po většinu jejich historie, byly systémy řízení bází dat používány výhradně ve velkých, objemných, průmyslových softwarových nasazeních. Toto se změnilo s příchodem SQLite v roce 2000. Na rozdíl od předešlých SŘBD, SQLite není samostatným programem, který komunikuje s uživatelským programem přes síť. Naopak je vestavěný přímo do uživatelských programů, jako jakákoliv jiná knihovna, pomocí linkeru.

Díky schopnosti efektivně dotazovat a psát data na disku, společně s možností vestavění do programu, spolehlivostí a štíhlostí, bylo SQLite velice rychle adoptováno. SQLite je dnes nejvíce používaný SŘBD podle počtu databází v aktivním nasazení: podle autorů SQLite, „zdá se pravděpodobné, že přes bilion (10^{12})“. Je všudepřítomné do té míry, že pokud tuto práci čtete v jednom z velkých webových prohlížečů (Chrome, Firefox, Safari, a prohlížečů na nich založených), pak jste nepřímým uživatelem SQLite.

V této práci, následujíc ve stopách SQLite 25 let po jeho první verzi, se snažíme vytvořit SŘBD podobné SQLite s některými novými vymoženostmi (dotazy napříč časem, grafové dotazy, apod.).

TEORETICKÁ ČÁST

1 Co je to systém řízení báze dat

Podle Edgara Franka Codda by plný SŘBD měl poskytovat následující vymoženosti:

1. ukládání, vyhledávání, a aktualizování dat,
2. poskytování katalogu popisující uložená data,
3. podporu transakcí, aby bylo zajištěné, že všechny ze řad operací jsou vykonány, nebo žádné nejsou,
4. systémy obnovení v případě selhání programu, operačního systému, či hardwaru,
5. systémy kontroly souběžnosti, které zajišťují, že souběžné transakce se chovají stejně, jako kdyby byly provedené samostatně,
6. systémy autorizace,
7. integrace s komunikačními systémy,
8. a systémy pro zajištění platných závislostí a stavů dat (Codd, 1982).

Podle Thomase Connollyho a Carolyn Beggové dále také:

9. systémy naopak pro podporu nezávislosti dat,
10. a obslužné programy pro například import dat či monitorování (Connolly a Begg, 2015).

SŘBD s těmito vymoženostmi se většinou dají rozdělit do dvou hlavních vrstev: datovou vrstvu a dotazovací vrstvu. Datová vrstva povětšinou zajišťuje vymoženosti 1, 3, a 4. Dotazovací vrstva pak zajišťuje zbytek vymožeností.

1.1 Ukládání, vyhledávání, a aktualizování dat

Ukládání, vyhledávání a aktualizování dat, často známo pod zkratkou CRUD (Create, Read, Update, Delete), patří mezi základní vymoženosti datové vrstvy a je popsáno blíže v kapitole 2.

V SQL těmto vymoženostem odpovídají příkazy `INSERT`, `SELECT`, `UPDATE` a `DELETE`. V SQL databázích tyto příkazy však nejsou prostým příkazem pro datovou vrstvu. Naopak jsou značně obohaceny dotazovací vrstvou.

Příkaz `SELECT` vyhledává a transformuje data v databázi podle nejrůznějších podmínek a vztahů. Příkazy `UPDATE` a `DELETE` využívají vyhledávací vymoženosti příkazu `SELECT` na aktualizování a mazání dat. Příkaz `INSERT` může přímo ukládat výsledky příkazu `SELECT`, ale také může ukládat data přímo zadaná.

1.2 Katalog popisující uložená data

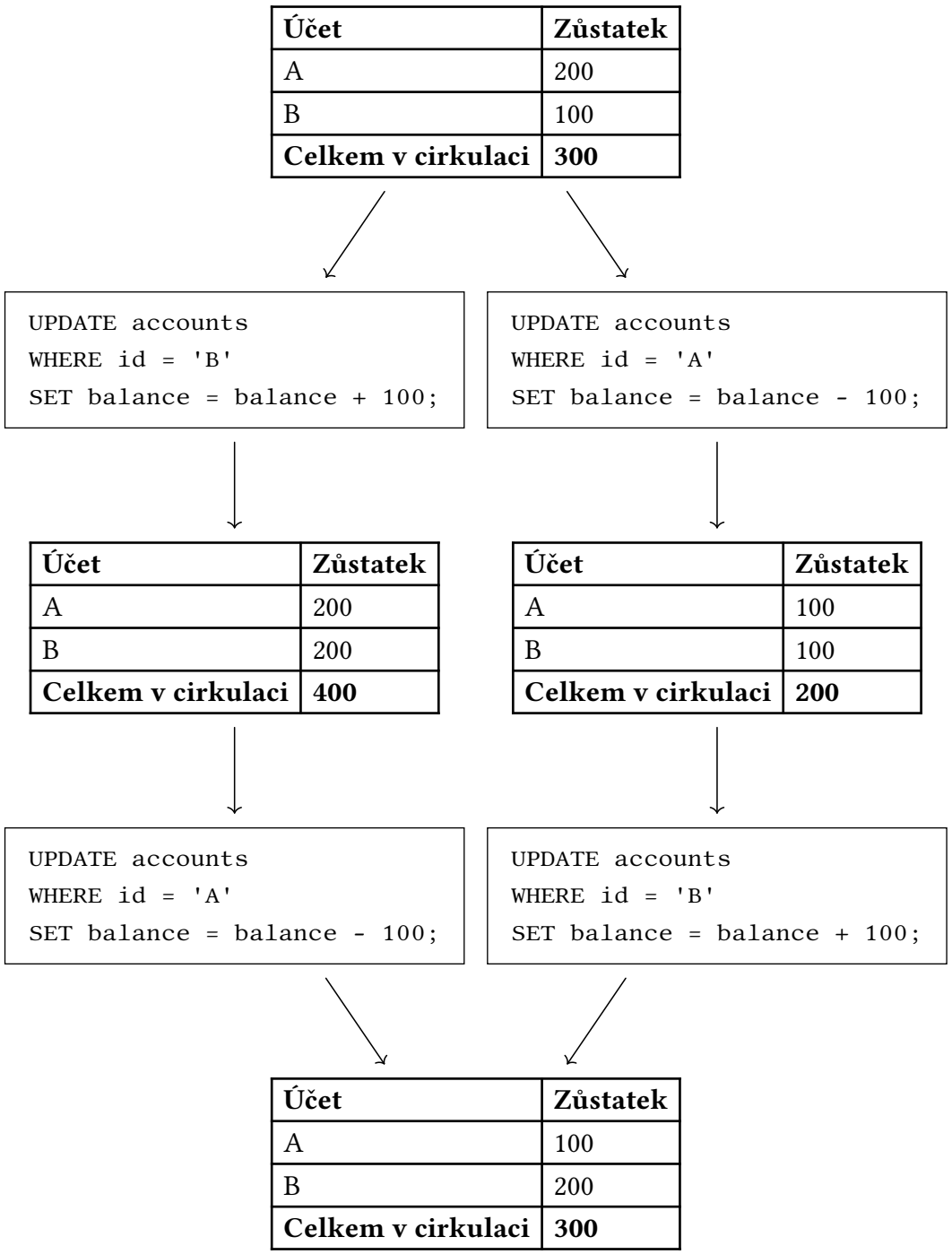
Katalog poskytuje dva typy informací o uložených datech: schéma popisující pravidla, která platí pro každé dané dato, a statistické informace o datech samotných.

V případě SQL jde o schémata jednotlivých tabulek. Schéma SQL tabulky zahrnuje jméno tabulky, počet sloupců, jména sloupců, datové typy sloupců, nulovatelnost, indexování, a vztahy k ostatním tabulkám ve formě cizích klíčů.

Mezi statistické informace, které jsou mimo jiné relevantní pro optimalizaci dotazů, patří například počet řádků dané tabulky, a histogram hodnot jednotlivých sloupců.

1.3 Transakce, souběžnost, obnovení

Typický příklad je příklad dvou bankovních účtů (viz Obrázek 1 na další straně). Z bankovního účtu A je poslána částka N na bankovní účet B. Během této operace je N odečteno ze zůstatku účtu A a přičteno na zůstatek účtu B. Pokud by tato operace selhala v jakémkoliv bodě, pak v závislosti na pořadí operací buď vznikly nové peníze na účtu B, nebo by peníze na účtu A zcela zanikly.



Obrázek 1: Příklad poslání částky mezi dvěma bankovními účty. Pokud operace selže mezi kterýmikoliv příkazy, dojde k nekonzistentnímu stavu. Toto platí pro každé pořadí operací. Obrázek autora

Řešením tohoto problému jsou takzvané transakce. Transakce je atomický (ve smyslu nerozdělitelný) shluk příkazů, které jsou prováděny ve stylu „všechno nebo nic“, to jest, buď jsou všechny příkazy úspěšně dokončeny nebo žádné z nich neplatí. V

SQL se transakcí týkají především příkazy `START TRANSACTION` pro započetí transakce, `COMMIT` pro úspěšné dokončení transakce, a `ROLLBACK` pro zrušení transakce, což je identické s transakčním selháním.

Důvody selhání transakcí se dají rozdělit do dvou skupin: logické chyby, a systémové selhání. Logická chyba by v tomto příkladu mohla být například, že jeden z účtů neexistuje. Systémové selhání by byla například chyba v programování SŘBD, nebo výpadek proudu.

Obnovení ze selhaných transakcí patří mezi vymoženosti datové vrstvy, a je blíže popsáno v kapitole 2.3.

V každém případě platí, že pokud nebylo uživateli oznámeno úspěšné dokončení transakce, má se za to, že transakce nikdy neproběhla/nikdy nebyla zadána a naopak, pokud bylo uživateli oznámeno úspěšné dokončení transakce, pak databáze musí být v odpovídajícím stavu při dalším dotazu.

Souběžnost je příbuzným problémem. Pokud by v našem příkladu během provádění transakce bylo zadáno poslání částky z účtu A na účet C, kde částka je vyšší něž zůstatek na účtu A po provedení předchozí transakce, pak by se mohlo stát, že bude tato transakce provedena před tím, než předchozí transakce odečte předchozí částku ze zůstatku, a databáze bude zase v nekonzistentním stavu.[1] Podobný případ nastane pokud úživatel zadá dotaz na zůstatky obou účtů: mohlo by se stát, že SŘBD odpoví se zdánlivě nekonzistentními daty.

Z tohoto důvodu je zapotřebí souběžnost kontrolovat. Obecně platí, že může souběžně probíhat jedna psací transakce, která mění nejnovější verzi dat, a libovolný počet čistě čtecích transakcí, které čtou z nejnovější verze dat, pokud neexistuje žádná souběžná psací transakce, v opačném případě čtou z předchozí verze dat. V určitých případech je možné, aby probíhalo více psacích transakcí souběžně.

1.4 Autorizace

Mnohé SŘBD mají mnoho uživatelů, ze kterých každý potřebuje jen nějakou podmnožinu pravomocí. Z tohoto důvodu mnoho SŘBD velkého měřítka disponují řízením přístupu a autorizací. Řízení přístupu může rozhodnout, zda daný uživatel může číst dané tabulky, zda může dané tabulky upravovat, apod.

[1] V souběžných systémech je tento problém nazývaný Double-spending.

Jelikož se tato práce zabývá koncově uživatelskými programy, které běží pod operačními systémy osobních zařízení, které sami řídí přístup uživatelů i programů, autorizace v SŘBD samotném není pro tuto práci zajímavá.

1.5 Integrace s komunikačními systémy

V praxi se často stává, že SŘBD běží na jiných počítačích, než které uživatelé používají, a tak musí mezi sebou nějakým způsobem komunikovat přes sítě.

Vzhledem k předmětu této práce, komunikační systémy také nejsou pro naše účely zajímavé. Jelikož SŘBD u koncově uživatelských programů povětšinou běží na stejném systému, není důvod, aby SŘBD bylo od programu oddělené komunikační vrstvou, naopak je nejužitečnější, pokud je vestavěné v samotných programech.

1.6 Zajištění platnosti dat

V rámci SQL je platnost dat zajištěna například podmínkami CHECK, které zajišťují, že daný řádek je platný, a cizími klíči, které zajišťují, že řádek na který daný řádek ukazuje skutečně existuje.

2 Datová vrstva

Datová vrstva je většinou v podstatě jednoduché takzvané klíč-hodnota úložiště (anglicky key-value store, v této práci také KV store), kde klíče a hodnoty jsou prosté bajtové pole. Samotné páry klíčů a hodnot jsou organizovány v podmnožinách. Tyto podmnožiny mají spoustu názvů: sloupcové rodiny, klíčové prostory, „kbelíky“, apod., v této práci je budeme nazývat „stromy“. Pro ideální fungování SŘBD, tento KV store musí disponovat jíž zmíněnými vymoženostmi:

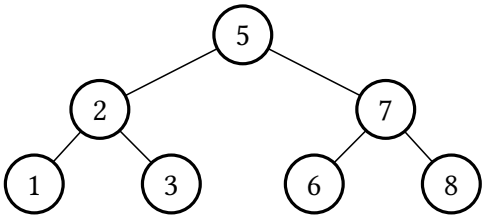
- ukládání, čtení, aktualizace, a mazání dat (CRUD operace),
- podpora atomických transakcí,
- a obnovení v případě systémového selhání.

Jelikož SŘBD vetšinou musí ukládat data o velikosti větší, než je velikost operační paměti systému, KV store musí být schopen efektivně načítat data po částech z disku. Z tohoto důvodu je tradičně implementován data strukturou nazývanou B-strom, i když v poslední době jsou také používané jiné data struktury jako jsou LSM stromy.

2.1 B-stromy

Pro pochopení B-stromů by mohlo být užitečné vrátit se zpět k prostým polím. Pokud jsou prvky v prostém poli seřazeny, pak je možné najít libovolný prvek v logaritmickém čase binárním vyhledáváním.

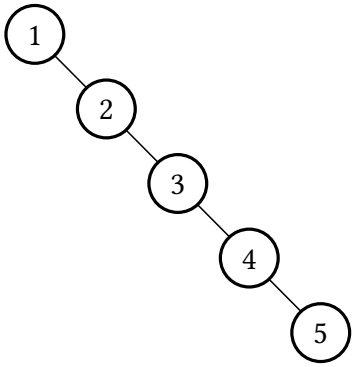
Problém je však ten, že vložení nového prvku do seřazeného pole v nejhorším případě trvá lineárně dlouho. Tento problém je (z části) řešen binárními vyhledávacími stromy, které se skládají z vrcholů, u kterých je každý levý potomek menší než jeho předek a pravý naopak větší. Vyhledávání (téměř) zůstává logaritmické a vkládání je nyní také (téměř) logaritmické, jelikož jen stačí najít nebližší předkový vrchol.



Obrázek 2: Binární vyhledávací strom obsahujíc číselnou řadu 1 až 7 včetně. Idealizovaný příklad. Obrázek autora

Problém binárních vyhledávacích stromů je, že je možné, ba v mnoha případech i velice pravděpodobné, že strom začne být nebalancovaný. V minulém paragrafu

jsme zmínili, že vyhledávání zůstává logaritmické, ale tomu tak je jen, pokud je strom balancovaný, to znamená, že pro každý vrchol (přibližně) platí, že levý podstrom je stejně veliký jako pravý podstrom. Pokud je pořadí vložených prvků skutečně neštastné (například pořadí čísel 1, 2, 3, 4, 5, viz Obrázek 3), pak se může stát, že prvek bude vždy v pravém, nebo levém vrcholu, v tomto případě by vyhledávání bylo lineární (platí i pro vlkádání). Řešením jsou takzvané samo-balancující-se vyhledávací stromy, které se sami balancují. Příkladem jsou červeno-černé stromy.

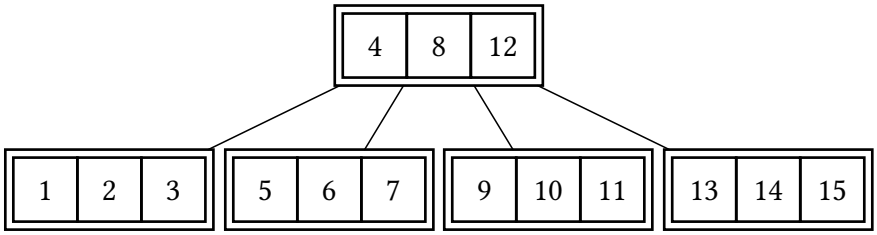


Obrázek 3: Nevybalancovaný binární vyhledávací strom obsahujíc číselnou řadu 1 až 5 včetně. Jelikož jsou pouze pravé vrcholy obsazeny, struktura je identická lineárnímu seznamu. Časová komplexita vyhledávání i vlkádání je linearní. Obrázek autora

Posledním naším problémem je, že samotné rozložení stromu v paměti je neefektivní. Při načítání dat z disku, disk data načítá po stránkách, které jsou obecně větší (na moderních zařízení vetšinou mnohem větší) než jeden vrchol našeho stromu. B-stromy řeší tento i předešlé problémy.

B-strom není binárním stromem (viz Obrázek 4 na další straně), ale je binárnímu stromu podobný. Zjednodušeně by se dalo říct, že B-strom zplošťuje několik úrovní binárního stromu do jedné. V B-stromu vrchol nereprezentuje prvek B-stromu, spíše se skládá ze svých potomků, které jsou všechny uloženy ve stejné diskové stránce. Kdežto vrchol v binárním stormu má nejvíce 2 přímé potomky, vrchol v B-stromu obecně má tolik potomků, kolik se vejde do diskové stránky, což představitelně může být až v řádu stovek.[2]

[2] $\frac{P-p}{k+v+p}$, kde P je velikost stránky (často 4096 bajtů), p je velikost ukazatele (vetšinou 64-bitový, tudíž 8 bajtů), a k a v jsou velikosti klíče a hodnoty (představitelně obě 8 bajtů), tudíž $\frac{4096-8}{8+8+8} \approx 170$.



Obrázek 4: B-strom o 3 potomcích obsahujíc číselnou řadu od 1 až po 15 včetně. Je zřejmá efektivita počtu úrovní stromu a vrcholů vůči počtu obsažených prvků. B--stromu o 3 potomcích je značně nerealistický, ale B-stromy s větším počtem potomků je složité zobrazit (kvůli počtu prvků/vrcholů), což mimo jiné svědčí o jejich efektivitě.

Obrázek autora

2.2 Porovnání hodnot

Hodnoty mohou být kódovány způsobem specificky uzpůsobeným, aby se hodnoty řadily podle lexikografického uspořádání i v kódované formě, když jsou bitově porovnávány.

Například klíče v datové vrstvě RocksDB pro MyRocks nebo SurrealDB, jsou řazeny čistě pomocí bitových porovnání bez dekódování (viz Obrázek 5). Tradičně jsou však hodnoty dekódovány a porovnávány v dekódované podobě. Dekódování umožnuje například řazení textu podle komplikovaných jazykových pravidel.

Křesní jméno										Příjmení										Rok	Název
S	t	e	i	n	b	e	c	k		J	o	h	n							1939	The Grapes of Wrath
53	74	65	69	6E	62	65	63	6B	00	4A	6F	68	6E	00						0793	
S	t	u	r	g	e	o	n			T	h	e	o	d	o	r	e			1953	More Than Human
53	74	75	72	67	65	6F	6E	00		54	68	65	6F	64	6F	72	65	00		07A1	
S	u	k	e	n	i	c	k			R	o	n	a	l	d					1968	Up
53	75	6B	65	6E	69	63	6B	00		52	6F	6E	61	6C	64	00				07B0	
T	h	e	r	o	u	x				A	l	e	x	a	n	d	e	r		1981	Darconville's Cat
54	68	65	72	6F	75	78	00			41	6C	65	78	61	6E	64	65	72	00	07BD	
T	o	o	m	e	r					J	e	a	n							1923	Cane
54	6F	6F	6D	65	72	00				4A	65	61	6E	00						0783	
V	o	l	l	m	a	n	n			W	i	l	l	i	a	m		T	.	1987	You Bright and Risen Angels
56	6F	6C	6C	6D	61	6E	6E	00		57	69	6C	6C	69	61	6D	20	54	2E	00	07C3
V	o	l	l	m	a	n	n			W	i	l	l	i	a	m		T	.	1994	The Rifles
56	6F	6C	6C	6D	61	6E	6E	00		57	69	6C	6C	69	61	6D	20	54	2E	00	07CA

Obrázek 5: Příklad bitového lexikografického řazení strukturovaných dat pomocí kódování storekey. Seznam knih je řazen podle příjmení autora, poté křesního jména autora, a nakonec roku vydání. Název knihy není v řazení zohledněn. První řádek je vždy lidsky čitelná reprezentace, druhý řádek jsou výsledné bajty v hexadecimální notaci. V datové vrstvě jsou bajtové řádky ve společném bajtovém poli, v tabulce jsou odděleny v zájmu přehlednosti. Obrázek autora

2.3 Obnovení

Na obnovení datové vrstvy se většinou používá takzvaný Write-Ahead-Log (WAL), nebo takzvané rollback deníky. Princip je jednoduchý, v případě Write-Ahead--Log, existuje vedle souboru databáze ještě jeden soubor, který je daný WAL. Při psaní do databáze je nejdříve psáno do souboru WAL, a poté při uspěšném dokončení jsou nové verze stránek překopírovány do původního databázového souboru a WAL je smazán. Tento proces je možný kdykoliv přerušit a rozumně ho poté pokračovat. Pokud je přerušen před úspěšným dokončením, pak se WAL jednoduše smaže. Pokud je přerušen po úspěšném dokončení, pak je možné dané stránky z WALu znovu překopírovat.

V případě rollback deníků platí obdobný proces, ale rollback deníky zaznamenávají původní stránky databáze a do databáze se píše přímo. Pokud není operace úspěšně dokončena, pak se kopírují stránky zpět, a pokud je úspěšně dokončena, pak se rollback deník maže.

3 Dotazovací vrstva

Uživatel (ať už fyzická osoba či jiný program) interaguje s datovou vrstvou přes takzvanou dotazovací vrstvu. Samotné dotazovací vrstvy existují na velikém spektru, často se od sebe mohou radikálně lišit. Od velice jednoduchých vrstev, které nesplňují všechny vymoženosti stanoveny v začátku této části, a které jsou víceméně jen systémem přikazování datové vrstvě (například Redis), přes dotazovací vrstvy odpovídající nějaké úzce vymezené doméně (například Elasticsearch, Meilisearch, Quickwit a další pro doménu textového vyhledávání), až po vysoce pokročilé dotazovací vrstvy, které sami dovedou řídit celé infrastruktury bez jakýchkoliv jiných programů (PostgreSQL).

3.1 Paradigmata

Dotazovací vrstvy jsou často (i když ne vždy) definované dotazovacími jazyky, které definují interakce s touto vrstvou. Dotazovací jazyky obecně přichází s vlastními paradigmaty. Nejčastější paradigmata pro obecné SŘBD jsou: relační, grafové a dokumentové.

3.1.1 Relační

V relačním paradigmatu jsou veškerá data ukládána v tabulkách. Vztahy mezi tabulkami A a B jsou reprezentovány takzvanými cizími klíči. V případě vztahu one-to-many (jeden-k-mnoha) je cizí klíč sloupec v tabulce B, v opačném případě vztahu many-to-one (mnoho-k-jednomu) naopak v tabulce A, a v případě vztahu many-to-many (mnoho-k-mnohu) je použita takzvaná hraniční tabulka C, která obsahuje cizí klíče ukazující na obě tabulky a tímto reprezentuje hranici (ve smyslu hranice mezi vrcholy grafu) mezi nimi (vztahy one-to-one (jeden-k-jednomu) jsou jednoduše ukládány v jedné tabulce v jednom řádku).

Tabulka zákazníků

ID	Jméno
1	Bill Smith
2	Jim Kenshaw

Tabulka obědnávek

ID	ID zákazníka	Datum	ID produktu	Počet
5	1	2020-07-03	536	1
6	1	2020-07-03	537	1

Obrázek 6: Příklad vztahu one-to-many a many-to-one. Jeden zákazník může mít mnoho obědnávek, a mnoho obědnávek může mít jednoho zákazníka. Obrázek autora

Relační paradigma je nejstarší z obecných paradigmat a bylo to toto paradigma, o kterém Edgar Frank Codd psal v jeho průkopnické práci (Codd, 1982). Relační paradigma ustálo zkoušku času a je stále nejčastějším paradigmatem (Red Gate Software Ltd, 2024).

3.1.2 Grafové

I přes veliký počet vymožeností relačního paradigmatu, nevyniká v reprezentaci komplikovaných, velice propojených dat. Pro tento účel vzniklo grafové paradigma, kde SŘBD ukládá vrcholy a hranice grafu, které poté mohou být „označené“ libovolnými daty.

Častá kritika grafového paradigma je, že převážná většina, pokud ne všechny, grafové SŘBD jsou výrazně pozadu co se týče technické vyspělosti (Boncz, 2022; Sikhinam, 2019). A vskutku, většina velikých průmyslových aplikací s grafovými daty, jako je Facebook či Twitter, používá relační SŘBD a grafové paradigma pouze emuluje (Marchukov, 2013; K., 2021; Hashemi, 2017).

Z těchto důvodu jsou grafové SŘBD používány hlavně ve „výzkumných“ aplikacích jako je analytika či odhalování podvodů (Webber a Robinson, 2017). I přes toto, některé grafové SŘBD mají zajímavé vymoženosti, jako Neo4j, která podle Johna Stegemana dokáže procházet grafy v konstantním čase, což by v tradičním relačním SŘBD muselo být v čase logaritmickém (Stegeman, 2023).

3.1.3 Dokumentové

Na druhé straně spektra, dokumentové paradigma tvrdí, že tabulky jsou příliš neohebné, omezující a nevhodné pro to, co moderní web většinou potřebuje: nekom-

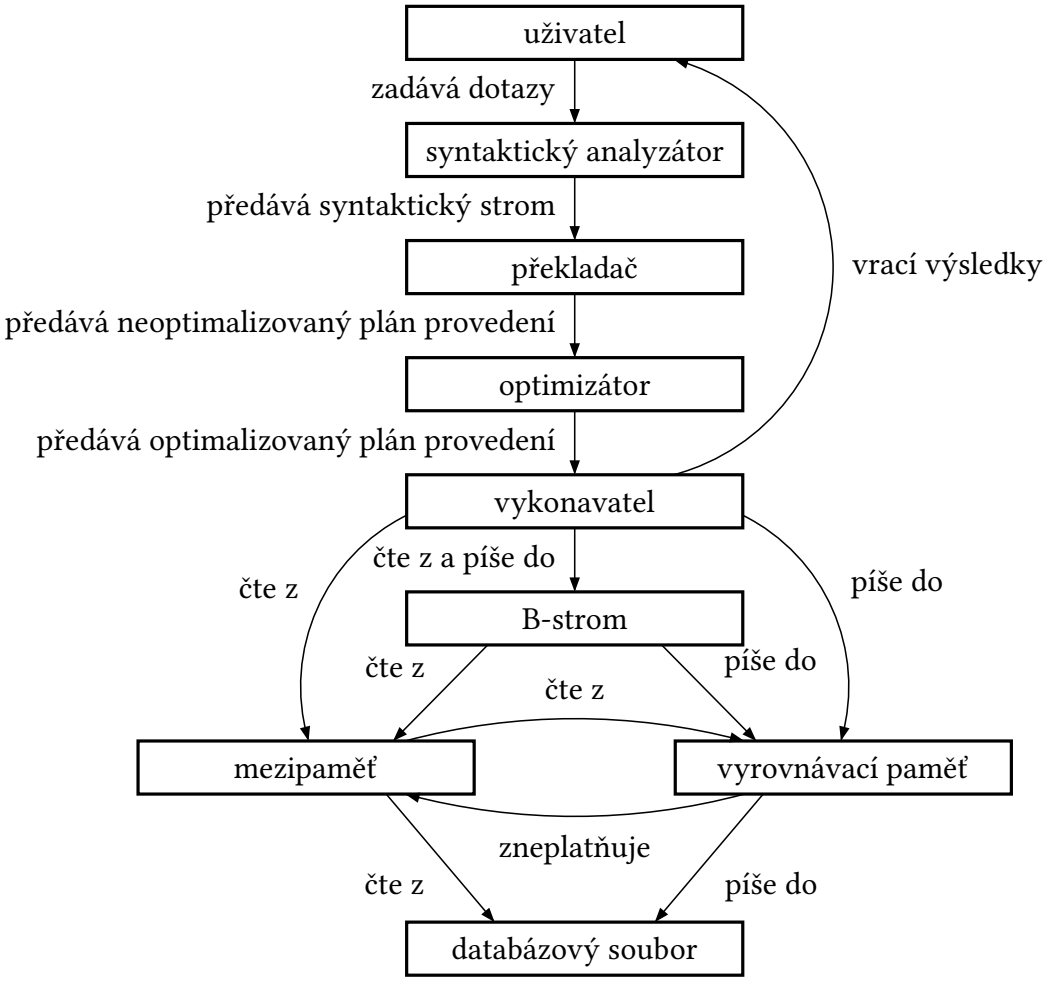
plikované ukládání a čtení JSONových objektů (neboli dokumentů). Takové SŘBD obecně dovoluje ukládání libovolných JSON dokumentů bez jakéhokoliv schématu. Hlavním příkladem dokumentových SŘBD je MongoDB (Red Gate Software Ltd, 2024).

3.1.4 Kombinované

V poslední době se objevily SŘBD kombinovaných paradigmat. Dva takovéto systémy jsou například SurrealDB a ArangoDB (SurrealDB Ltd., 2024; ArangoDB, 2024).

PRAKTICKÁ ČÁST

4 Architektura



Obrázek 7: Diagram architektury matdb. Obrázek autora

SŘBD je programovaný v jazyce Rust s pracovním názvem „matdb.“ Zdrojový kód je k dispozici v repozitáři https://gitlab.com/vizdun/matdb. V obrázku 7 je možné vidět diagram architektury. Aby bylo předejito zahlcením informacemi, bude v práci architektura nejprve popsána z dola nahoru, a poté budou informace doplněny zkráceným popsáním v opačném směru.

Matdb nepoužívá SQL, ale používá dotazovací jazyk podobný SQL, proto jej ve zbytku práce budeme také nazývat SQL, i když nenásleduje žádnou SQL normu.

5 Formát databázového souboru

Tento formát je do velké míry inspirován formátem databázových souborů SQLite (Hipp, 2024). Databázový soubor je rozdělen do stránek předem dané velikosti. Tyto stránky jsou rozděleny podle účelu do 6 typů:

- záhlavní stránky,
- zápatní stránky,
- B-stromové stránky:
    - vnitřní,
    - a listové,
- přetékací stránky,
- a volno-seznamové stránky.

Typ stránky je vždy značen prvním bajtem. Odpovídající hodnoty prvního bajtu jednotlivých typů stránek jsou vidět v obrázku 8.

Velikost stránky databázového souboru (to znamená, velikost každé stránky v databázovém souboru) bude v této kapitole označena písmenem P.

Všechny integery jsou kodované podle způsobu Big-endian.

Každá stránka je unikátně označena jejím číslem (někdy také zváno ukazatel). Toto číslo je 64-bitový integer, který představuje pořadové číslo této dané stránky v databázovém souboru, kde první (záhlavní) stránka má pořadové číslo 0, další pak 1, atd. Typicky nemá smysl ukazovat na záhlavní stránku, a tak ukazatel s nulovou hodnotou může sloužit jako reprezentace chybějící hodnoty. Když je v této práci řečeno „stránka," často je myšleno její číslo (ukazatel), nikoliv její bitová reprezentace.

Volná stránka se rozumí stránka, která se již nachází v databázovém souboru, a která byla v minulé činnosti SŘBD použita, ale která je nyní volná k přepsání.

Typ stránky	Hodnota
záhlavní	0x4D
zápatní	0x4E
vnitřní B-stromová	0x01
listová B-stromová	0x02
přetékací	0x53
volno-seznamová	0x54

Obrázek 8: Hodnoty prvního bajtu podle typu stránky. Obrázek autora

5.1 Záhlavní stránky

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x4D	volný prostor														
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															

Obrázek 9: Struktura záhlavní stránky o velikosti 256 bajtů. Obrázek autora

V platném databázovém souboru je vždy právě jedna záhlavní stránka, je to vždy první stránka databázového souboru. Zároveň je nejjednodušším typem stránky, protože v současnosti neobsahuje žádná data kromě prvního bajtu, který má hodnotu 0x4D, čímž jí označuje jako záhlavní.

5.2 Zápatní stránky

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x4E	ukazatel na kořen hlavního stromu								číslo transakce						
-	ukazatel na následující volno-seznam								volné stránky						
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-									volný prostor						

Obrázek 10: Struktura zápatní stránky o velikosti 256 bajtů. Obrázek autora

Zápatní stránky se vždy nacházejí na samotném konci platného databázového souboru. Kdykoliv během operace SŘBD je možné načíst první zápatní stránku od konce souboru a ocitnout se tak v plně platném stavu.

Zápatní stránka je ozančena prvním bajtem o hodnotě 0x4E. Po tomto bajtu následuje ukazatel na B-stromovou stránku, která je kořenem hlavního stromu. Dále 64-bitový integer představující číslo transakce, kterou tato zápatní stránka zapečeťuje. Následuje ukazatel na volno-seznamovou stránku obsahujíc ukazatele na volné stránky, které následují volné stránky, na které jsou ukazatele obsažené v této zápatní stránce. Pokud je hodnota tohoto ukazatele nulová, pak další volno-seznamová stránka neexistuje. Dále následuje až $\frac{P-1-8-8-8}{8}$ ukazatelů na volné stránky. První ukazatel s nulovou hodnotu představuje předčasný konec seznamu.

5.3 B-stromové stránky

Matdb používá specifickou variantu B-stromu. Tato varianta se nazývá B+ strom. B+ strom je v podstatě stejný jako B-strom popsaný v kapitole 2.1, ale B+ strom na rozdíl od B-stromu obsahuje páry klíčů a hodnot jen v listových vrcholech, ostatní

vrcholy jsou čistě navigační. Toto zjednodušuje a zefektivňuje některé operace. Aby byly dotazy napříč časem možné, je tento B-strom také persistentní.

5.3.1 Buňky

Společným rysem vnitřních a listových B-stromových stránek je koncept takzvané „buňky." Tato buňka slouží pro uložení bajtového pole o libovolné velikosti. Jelikož je každá B-stromová stránka velikostně omezena na P bajtů, a protože je ideální, aby B-stromové stránky obsahovaly co nejvíce hodnot, jsou bajtové pole po dané velikosti C bajtů rozděleny, kdy část pole zůstává v domovské B-stromové stránce a část takzvaně „přeteče" na přetékací stránky. Od tohoto se odvíjí dvě varianty buněk:

- celá, kdy celé bajtové pole je obsaženo v B-stromové stránce,
- a částečná, kdy je v B-stromové stránce jen část pole.

5.3.1.1 Buňka celá

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
velikost	pole						

Obrázek 11: Struktura buňky celé. Obrázek autora

První bajt buňky celé tvoří 8-bitový integer představující počet bajtů v bajtovém poli této buňky. Tento počet je vždy striktně menší než C. Následuje bajtové pole o právě tomto počtu bajtů.

5.3.1.2 Buňka částečná

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
C	pole						
ukazatel na první přetékací stránku							
zbývající délka							

Obrázek 12: Struktura buňky částečné. Obrázek autora

V buňce částečné má první bajt vždy hodnotu C. Následuje bajtové pole o $C - 16$ bajtů. Poté následuje ukazatel na první přetékací stránku, ve kterém bajtové pole pokračuje. Dále následuje 64-bitový integer představujc zbývající délku bajtového pole, tedy délku té části bajtového pole, která se nachází v přetékacích stránkách, tedy $L - (C - 16)$, kde L je celková délka uloženého bajtového pole.

5.3.2 Vnitřní B-stromové stránky

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x01	číslo transakce								počet párů						
-	nejlevější ukazatel								páry buněk a ukazatelů						
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															

Obrázek 13: Struktura vnitřní B-stromové stránky o velikosti 256 bajtů. Obrázek autora

Vnitřní B-stromová stránka obsahuje klíče a ukazatele na potomkové stránky. Kromě nejlevějšího ukazatele jsou tyto klíče a ukazatele organizovány v párech, kdy první z páru je buňka představující klíč, a druhý z páru je ukazatel. Páry jsou seřazeny podle klíčů: vždy platí, že klíč nalevo od jiného klíče je striktně menší, a naopak. Ukazatelé jsou také seřazeny podle obsahu potomkových stránek: vždy platí, že ukazatel nalevo od daného klíče ukazuje na stránku obsahující klíče buď menší nebo rovno tomuto klíči. Naopak ukazatel napravo od daného klíče vždy ukazuje na stránku obsahující klíče striktně větší než tento klíč.

První bajt má hodnotu 0x01, čímž tuto stránku označuje jakožto vnitřní B--stromovou. Dále následuje 64-bitový integer představující číslo transakce, ve které byla tato stránka napsána. Poté následuje 64-bitový integer představující počet párů tvořených za prvé buňkou, která představuje klíč, a poté ukazatelem na potomkovou stranu. Následuje nejlevější ukazatel a poté páry buněk a ukazatelů o již zmíněném počtu.

5.3.3 Listové B-stromové stránky

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x02	číslo transakce								počet párů buněk						
-	buňkové páry														
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															

Obrázek 14: Struktura listové B-stromové stránky o velikosti 256 bajtů. Obrázek autora

Listová B-stromová stránka obsahuje klíče a jejich korespondující hodnoty. Klíč a jeho korespondující hodnota je reprezentována párem buněk. Tyto páry buněk jsou ve stránce seřazeny podle klíčů: vždy platí, že pár buněk na levo od jiného páru buněk má striktně menší klíč a naopak.

Po prvním bajtu o hodnotě 0x02, který tuto stránku označuje jako listovou B--stromovou, následuje 64-bitový integer představující číslo transakce, ve které byla tato stránka napsána. Dále následuje 64-bitový integer představující počet párů buňěk obsažených v této stránce. Poté následuje právě tento počet párů buňek, kdy první buňka z páru je klíč, a druhá buňka z páru je hodnota.

5.4 Přetékací stránky

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x53	číslo transakce								ukazatel na další stránku						
-	bajtové pole														
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															

Obrázek 15: Struktura přetékací stránky o velikosti 256 bajtů. Obrázek autora

Přetékací stránky obsahují přetečený zbytek bajtového pole dané buňky. Fungují stylem spojeného seznamu.

Po typově-identifikačním prvním bajtu s hodnotou 0x53, následuje číslo transakce, a poté ukazatel na další přetékací stránku, která bajtové pole pokračuje. Zbytek stránky je část bajtového pole buňky. Pokud je hodnota ukazatele nulová, pak je tato stránka poslední v seznamu. V tomto případě přesný konec bajtového pole lze zjistit z celkového počtu bajtů, který je zapsaný v buňce (viz Kapitola 5.3.1.2).

5.5 Volno-seznamové stránky

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0F
0x54	ukazatel na další volno-seznam								volné stránky						
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-															
-									volný prostor						

Obrázek 16: Struktura volno-seznamové stránky o velikosti 256 bajtů. Obrázek autora

Volno-seznamové stránky obsahují ukazatele na volné stránky. Fungují podobně jako přetékací stránky ve stylu spojeného seznamu.

Po prvním bajtu s hodnotou 0x54, následuje ukazatel na další volno-seznamovou stránku, která seznam pokračuje, poté následuje až $\frac{P-1-8}{8}$ ukazatelů na volné stránky. Podobně jako v přetékacích stránkách, pokud je ukazatel na další stránku nulový, pak je tato stránka poslední. A podobně jako u zápatních stránek, pokud je jeden z ukazatelů na volné stránky nulový, pak značí předčasný konec seznamu.

6 Mezipaměť a vyrovnávací paměť

Aby se předešlo zbytečnému čtení z disku, pokaždé, když SŘBD přečte danou stránku ze souboru, je tato stránka umístěna do mezipaměti. Při dalších čteních této stránky bude tato stránka přečtena nejdříve z mezipaměti, a jen poté, pokud se v mezipaměti nevyskytuje, bude přečtena ze souboru na disku (a znovu umístěna do mezipaměti). Mezipaměť má předem definovanou velikost. Každá stránka v mezipaměti má časový údaj (v praxi prostý vstoupající integer), a to, kdy byla naposledy přečtena. V případě že je načtena nová strana a mezipaměť je plná, bude strana v mezipaměti s nejstarším časovým údajem (v praxi nejnižším integerem) smazána a nová strana bude načtena do mezipaměti.

```
1  pokud je stránka p v mezipaměti
2      načti stránku p z mezipaměti
3      vrať stránku p
4  jinak
5      pokud je stránka p ve vyrovnávací paměti
6          načti stránku p z vyrovnávací paměti
7      jinak
8          načti stránku p z databázového souboru
9      konec podmínky
10     pokud je mezipaměť plná
11         najdi stránku s nejmenším časovým údajem a vymaž jí z mezipaměti
12     konec podmínky
13     ulož stránku p do mezipaměti
14     vrať stránku p
15 konec podmínky
```

Algoritmus 1: Čtení stránky p

Z podobného důvodu existuje vyrovnávací paměť. Aby se předešlo zbytečnému (a neorganizovanému) psaní na disk, jsou veškeré vypsané stránky nejdříve ukládány do vyrovnávací paměti. Tato paměť má také omezenou velikost, ale liší se činností, kterou provádí při jejím naplnění. Když se má vypsat nová stránka a vyrovnávací pamět je plná, pak jsou všechny stránky ve vyrovnávací paměti vypsané do souboru najednou a je do ní uložena nová stránka. Pokud se stránka, která byly uložena do vyrovnávací paměti, také nachází v mezipaměti, pak je její verze v mezipaměti zneplatňena a je z mezipaměti vymazána (ale nová verze není do mezipaměti

zapsána). Pokud by stránka, která se v té době nachází ve vyrovnávací paměti, měla být přečtena, pak je přečtena přednostně z vyrovnávací paměti, nikoliv z disku.

```
1 pokud je vyrovnávací pamět plná
2 |  vypiš všechny stránky ve vyrovnávací paměti
3 konec podmínky
4 pokud je stránka p v mezipaměti
5 |  vymaž stránku p z mezipaměti
6 konec podmínky
7 ulož stránku p do vyrovnávací paměti
```

Algoritmus 2: Psaní stránky p

7 Alokace a uvolňování stránek

V nejjednodušším případě jsou nové stránky přidány na konec souboru. Zápatní stránky jsou vždy přidány na konec souboru. Ale aby se předešlo plýtvání s prostorem, všechny ostatní typy stránek mohou místo toho přepsat původní stránky, pokud tyto stránky byly uvolněny v předešlých operacích.

```
1  pokud je parametr p definován, a stránka p může být uvolněna (viz Algoritmus 4)
2  | alokuj novou stránku na místo stránky p
3  jinak pokud seznam volných stránek v paměti není prázdný
4  | alokuj novou stránku na místo stránky ze seznamu volných stránek v paměti
5  jinak pokud je ukazatel posledního volno-seznamu nenulový
6  | načti poslední volno-seznam do paměti
7  | uvolni poslední volno-seznam (spustí Algoritmus 4)
8  | alokuj novou stránku na místo stránky ze seznamu volných stránek v paměti
9  jinak
10 | alokuj novou stránku na konci souboru
11 konec podmínky
```

Algoritmus 3: Alokace nové stránky, s volitelným parametrem stránky k přepsání p

SŘBD udržuje v paměti až $\frac{P-1-8-8-8}{8} + \frac{P-1-8}{8}$ volných stránek (počet ukazatelů na volné stránky, které se vejdou do jedné zápatní a jedné volno-seznamové stránky). Pokud je tato paměť zaplněna, bude část z volných stránek vypsána do volno-seznamové stránky. Volné stránky jsou rozděleny do stránek. které mohou být znovu alokovány ihned, a do stránek, které mohou být alokovány až v příští transakci (typicky volno-seznamové stránky). Volné stránky, které mohou být alokovány až v příští transakci jsou vypisovány do volno-seznamových stránek přednostně. V určitých případech je vhodné, aby nová alokace byla stejná stránka, která je právě uvolněna.

Ne všechny stránky mohou být uvolněny:

- záhlavní a zápatní stránky nikdy nejsou uvolněny,
- B-stromové a přetékací stránky mohou být uvolněny pouze, pokud byly vytvořeny ve stejné transakci, která se je pokouší uvolnit (toto se dá triviálně zjistit porovnáním jejich transakčních čísel, viz Kapitola 5),
- volno-seznamové stránky mohou být vždy uvolněny, ale alokovány mohou být až v další transakci.

1 **pokud** je p stránka záhlavní či zápatní
2 | stránka p nemůže být uvolněna
3 **jinak pokud** je p stránka B-stromová či přetékací a její číslo transakce se shoduje se současnou transakcí
4 | přidej stránku p na seznam volných stránek v paměti
5 **jinak pokud** je p stránka volno-seznamová
6 | přidej stránku p na seznam volných stránek v paměti, které mohou být alokovány až v další transakci
7 **konec podmínky**
8 **pokud** je počet stránek v pamětních seznamech volných stránek roven $\frac{P-1-8-8-8}{8} + \frac{P-1-8}{8}$
9 | vezmi ze seznamu stránek, které mohou být použity až v další transakci až $\frac{P-1-8}{8}$ ukazatelů na volné stránky
10 | vezmi případný zbytek tohoto počtu ze seznamu stránek, které mohou být použity ihned
11 | vypiš volno seznamovou stránku s těmito ukazateli a s ukazatelem na případný předešlý volno-seznam
12 | aktualizuj pamětní ukazatel posledního volno-seznamu na tuto stránku
13 **konec podmínky**

Algoritmus 4: Uvolňení stránky p

8 Katalog

SŘBD rozeznává následující SQL datatypy hodnot:

- `NULL` - nulová hodnota určena k reprezentování absence hodnoty,
- `BOOL` - pravdivostní hodnoty pravda `TRUE` či nepravda `FALSE`,
- `INT` - celé číslo v rozmezí -2^{63} až $2^{63} - 1$,
- `TEXT` - UTF-8 enkódovaný řetězec znaků.

Tyto hodnoty jsou kódovány do bajotvých polí neohraničené velikosti a poté vkládány do B-stromů jako klíče i jako hodnoty.

Kódované hodnoty začínají typovým bajtem:

- `0x01` - hodnota je `NULL`,
- `0x02` - hodnota je `FALSE`,
- `0x03` - hodnota je `TRUE`,
- `0x04` - hodnota je `INT`, následuje 8 bajtů integeru,
- `0x05` - hodnota je `TEXT`, následují bajty UTF-8 kódovaného textu zakončené nulovým bajtem `0x00`.

B-stromy představují tabulky. Klíč v tabulkovém B-stromu jsou hodnoty primárního klíče příslušné tabulky (respektive příslušného řádku). Hodnota v tabulkovém B-stromu jsou jednotlivé uspořádáné sloupce tabulky (respektive řádku).

K popisování vlastností některých tabulek si vypůjčíme anglický termín „readonly“ (znamenajíc „pouze pro čtení“). Když je některá tabulka „readonly,“ znamená to, že uživatel do ní nemůže psát přímo. SŘBD do těchto tabulek stále však píše.

Indexy jsou jedny ze speciálních readonly tabulek. V indexových tabulkách je klíč tvořen hodnotami indexovaných výrazů následovaný hodnotami primárního klíče indexováné tabulky (respektive indexovaného řádku). Tímto je řešena situace, kdy se dva řádky shodují v jejích indexovaných hodnotách, jelikož je každý primární klíč podle definice v tabulce unikátní.

Dalšími speciálními readonly tabulkami jsou interní tabulky. Tyto tabulky jsou kritické pro správné fungování SŘBD.

8.1 Interní tabulky

8.1.1 `matdb_tables`

Interní tabulka `matdb_tables` obsahuje číslo kořenové stránky všech ostatních tabulek (tedy kromě tabulky `matdb_tables` samotné, číslo kořenové stránky

`matdb_tables` se nachází v zápatní stránce, viz kapitola 5.2). SŘBD tuto tabulku používá kdykoliv čte z databáze. Její schéma je následující:

```
CREATE READONLY TABLE matdb_tables (
  name TEXT PRIMARY KEY,
  tree_id INT NOT NULL
);
```

8.1.2 `matdb_schemas`

Interní tabulka `matdb_schemas` obsahuje schémata všech tabulek (včetně sebe a ostatních interních tabulek). Kdykoliv SŘBD potřebuje zjistit schéma nějaké tabulky, najde ho v tabulce `matdb_schemas` (poznámka: toto se nevztahuje na interní tabulku `matdb_tables`, protože by to způsobilo nekonečnou rekurzi, proto je schéma tabulky `matdb_tables` pevně zakódováno, ale i tak se nachází v tabulce `matdb_schemas` a je plně platné). Schéma samotné je reprezentované jako text ve formě SQL příkazu, který by danou tabulku vytvořil. Schémata ukázaná v této kapitole jsou proto ta stejná, která by mohla být nalezena v tabulce `matdb_schemas`. Schéma tabulky `matdb_schemas`:

```
CREATE READONLY TABLE matdb_schemas (
  name TEXT PRIMARY KEY,
  schema TEXT NOT NULL
);
```

8.1.3 `matdb_revs`

V interní tabulce `matdb_revs` se nachází seznam všech předešlých verzí databáze (kromě té poslední, tato je ukládána v zápatní stránce) a jejich korespondující číslo kořenové stránky hlavního B-stromu (tedy kořen tabulky `matdb_tables`). Tabulka `matdb_revs` umožňuje dotazy napříč časem a SŘBD jí použije kdykoliv je takový dotaz zadán. Její schéma je:

```
CREATE READONLY TABLE matdb_revs (
  tx_id TEXT PRIMARY KEY,
  page_number TEXT NOT NULL
);
```

9 B-stromové operace

9.1 Čtecí operace

SŘBD čte z B-stromu algoritmy 5 a 6. Algoritmus 5 konkrétně odpovídá na otázku: jaký pár klíče a hodnoty v B-stromu s kořenem t má nejmenší klíč, který má předponu p, a který, pokud je hodnota c definována, má první po-předponí hodnotu, která má vztah r k hodnotě c?

Kde kořen t B-stromu je B-stromová stránka, která představuje kořen B--stromu, předpona p je pole hodnot o libovolné velikosti (může být i prázdné), hodnota c je jakákoliv hodnota, a vztah r je jeden z těchto vztahů:

- $>$ první po-předponí hodnota v klíči je striktně větší než c
- $\geq$ první po-předponí hodnota v klíči je větší či rovna c

Buňky jsou algoritmem procházeny a jejich klíče jsou postupně porovnávány s předponou p a popřípadě s hodnotou c.

```
1  pokud je stránka t listová
2  | procházej buňky
3  | | pokud je předpona současného klíče větší než předpona p
4  | | | vrať nulovou hodnotu
5  | | konec podmínky
6  | | pokud je předpona současného klíče rovna předponě p
7  | | | pokud není hodnota c definována
8  | | | | vrať současnou buňku
9  | | | jinak pokud je první po-předponová hodnota klíče menší než c
10 | | | | opakuj smičku
11 | | | jinak pokud je první po-předponová hodnota klíče větší než c
12 | | | | vrať současnou buňku
13 | | | jinak pokud je první po-předponová hodnota klíče rovna c
14 | | | | pokud je vztah r definován jako ≥
15 | | | | | vrať současnou buňku
16 | | | | jinak
17 | | | | | ukazatel := ukazatel + 1
18 | | | | | opakuj smičku
19 | | | | konec podmínky
20 | | | konec podmínky
21 | | konec podmínky
22 | konec smičky
```

```
23  | vrať nulovou hodnotu
24 jinak pokud je stránka t vnitřní
25  | pokud je pole buněk prázdné
26  | | vrať nulovou hodnotu
27  | konec podmínky
28  | ukazatel := 0
29  | opakuj
30  | | pokud je ukazatel větší než velikost pole
31  | | | ukonči smičku
32  | | konec podmínky
33  | | pokud je předpona současného klíče menší než předpona p
34  | | | ukazatel := ukazatel + 1
35  | | | opakuj smičku
36  | | konec podmínky
37  | | pokud je předpona současného klíče větší než předpona p
38  | | | ukonči smičku
39  | | konec podmínky
40  | | pokud je předpona současného klíče rovna předponě p
41  | | | pokud není hodnota c definována
42  | | | | ukonči smičku
43  | | | konec podmínky
44  | | | pokud je první po-předponová hodnota klíče menší než c
45  | | | | ukazatel := ukazatel + 1
46  | | | | opakuj smičku
47  | | | konec podmínky
48  | | | pokud je první po-předponová hodnota klíče větší než c
49  | | | | ukonči smičku
50  | | | konec podmínky
51  | | | pokud je první po-předponová hodnota klíče rovna c
52  | | | | pokud je vztah r definován jako ≥
53  | | | | | ukonči smičku
54  | | | | jinak
55  | | | | | ukazatel := ukazatel + 1
56  | | | | | opakuj smičku
57  | | | | konec podmínky
58  | | | konec podmínky
59  | | konec podmínky
60  | konec smičky
```

```
    rekurzivně aplikuj Algoritmus 5, kdy kořen bude potomková stránka
61  buňky na levo od současné buňky, nebo nejlevější ukazel pokud je-li
    současná buňka ta první
62 konec podmínky
```

Algoritmus 5: Hledání prvního páru klíče a hodnoty v B-stromu

Algoritmus 6 je uzpůsoben hledání páru následujícího po páru vyhledaným buď algoritmem 5, nebo předchozí iterací algoritmu 6. Obdobně jako algoritmus 5 příjímá kořen B-stromu t, dále pak předchozí klíč k_p. Algoritmus 6 v podstatě vyhledává první buňku s klíčem větším než klíč k_p.

```
 1 pokud je stránka t listová
 2     pokud je pole buněk prázdné
 3         vrať nulovou hodnotu
 4     konec podmínky
 5     procházej buňky
 6         pokud je současný klíč větší než předchozí klíč k_p
 7             vrať současnou buňku
 8         konec podmínky
 9     konec smičky
10     vrať nulovou hodnotu
11 jinak pokud je stránka t vnitřní
12     pokud je pole buněk prázdné
13         vrať nulovou hodnotu
14     konec podmínky
15     procházej buňky
16         pokud je současný klíč větší než předchozí klíč k_p
17             vrať předchozí ukazatel (buď ukazatel předchozí buňky nebo nejlevější
               ukazatel)
18         konec podmínky
19     konec smičky
20     vrať nulovou hodnotu
21 konec podmínky
```

Algoritmus 6: Hledání následujícího páru klíče a hodnoty v B-stromu

Tyto dva algoritmy jsou spojené v algoritmu 7, díky kterému SŘBD dokáže číst klíče B-stromu v intervalu s předponou pár po páru, i kdyby byly stránky B-stromu

změněny mezi iteracemi. Algoritmus znova přijímá kořenovou stranu t, předponu p, a nově interval i. Předešlý klíč p_k je nyní součástí stavu programu a zůstává mezi iteracemi algoritmu.

```
1  pokud je p_k nenulový
2      spusť Algoritmus 6
3      pokud má výsledek shodnou předponu a jeho další hodnota se nachází v
       intervalu i
4          vrať tento výsledek
5      jinak
6          nastav p_k na nulovou hodnotu
7          vrať nulovou hodnotu
8      konec podmínky
9  jinak
10     pokud i = (x, y) nebo i = (x, y]
11         definuj c jako x a r jako >
12     jinak pokud i = [x, y) nebo i = [x, y]
13         definuj c jako x a r jako ≥
14     konec podmínky
15     spusť Algoritmus 5
16     nastav p_k na nulovou hodnotu
17     vrať výsledek
18 konec podmínky
```

Algoritmus 7: Procházení párů v B-stromu s klíči v intervalu s předponou

9.2 Psací operace

Psací operace jsou v určitém smyslu jednodušší, a v určitém smyslu složitější. Již nemáme problém spletitého hledání klíčů, ale máme nový problém: stromové rebalancování.

9.2.1 Rebalancovácí operace

Aby zůstal B-strom efektivní, musí si udržet určité vlastnosti. Jmenovitě každý z jeho vrcholů nesmí být ani moc plný, ani moc prázdný. Pokud je vrchol moc plný, bude rozdělen ve dva vrcholy a tyto vrcholy budou úmístěny v jemu nadřazený vrchol. Samozřejmě toto může způsobit, že jemu nadřazený vrchol bude nyní také příliš plný. V tomto případě bude i on rozdělen atd.

Naopak pokud je vrchol příliš prázdný, bude sloučen s jiným vrcholem na stejné úrovni. Samozřejmě toto může způsobit, že jemu nadřazený vrchol bude příliš prázdný atd.

9.2.1.1 Dělení vrcholů

V případě, že je vrchol po nějaké psací operaci příliš velký (konkrétně pokud součet velikostí všech buněk vrcholů je větší, něž kolik je místa ve stránce) bude rozdělen v polovinu.

Pokud se jedná o listový vrchol, budou jeho buňky jednoduše rozděleny na dvě poloviny, z niž budou vytvořeny dva nové vrcholy. Levý vrchol zaujímá pozici původního vrcholu v nadřazeném vrcholu s novým klíčem, který je totožný s jeho největším (co se pořadí týče) klíčem. Pravý vrchol zaujímá pozici nové buňky napravo od buňky levého vrcholu, má klíč buňky původního vrcholu. Stránka a všechny buňky původního vrcholu jsou pokud možno uvolněny.

Obdobně, pokud se jedná o vnitřní vrchol, bodou také rozděleny na dvě poloviny, ale nově bude prostřední klíč vyjmut (jeho ukazel se stane nejlevějším ukazatelem pravé poloviny). Vyjmutý prostřední klíč zaujme místo napravo od původního klíče a jeho ukazatel bude ukazovat na vrchol vytvořený z pravé poloviny. Původní klíč ukazuje na levou polovinu. Obdobně je stránka a všechny buňky původního vrcholu pokud možno uvolněny.

9.2.1.2 Slučování vrcholů

V případě, že je vrchol po nějaké psací operaci příliš malý, bude sloučen se sousedním vrcholem.

Pokud se jedná o listový vrchol, bude sloučen se sousedním listovým vrcholem. Buňky levého vrcholu budou jednoduše přidány před buňky pravého vrcholu a levý vrchol bude vymazán.

Pokud se jedná o vnitřní vrchol, bude obdobně sloučen se sousedním vnitřním vrcholem.

Samozřejmě se může stát, že bude po této operaci vrchol příliš velký. V tomto případě bude rozdělen.

9.2.2 Vkládání páru

Nejdříve je potřeba najít vhodné místo pro klíč páru. Jelikož je nyní celý klíč znám, je toto triviální. Pokud je toto místo již obsazené, je stávající pár přepsán. V opačném případě je na tomto místě nový pár vložen. Pokud je možné stránku

uvolnit, je přepsána. V opačném případě je vypsána jakožto nová stránka v souladu s kapitolou 7. Následuje rebalancování.

9.2.3 Mazání páru

Mazání páru je totožné s vkládáním, ale na místo přepisování či vkládání je pár vyjmut (v případě, že pár s daným klíčem neexistuje, není nic vykonáno) a pokud možno uvolněn. Následuje rebalancování.

9.2.4 Mazání stromu

Při mazání stromu je strom projit každým klíčem a větví od kořene až k listovým vrcholům a poté jsou jeho stránky a buňky v obrácené posloupnosti uvolňovány (pokud je toto možné podle kapitoly 7). Poté co je strom celý uvolněn může být pokládán za smazaný.

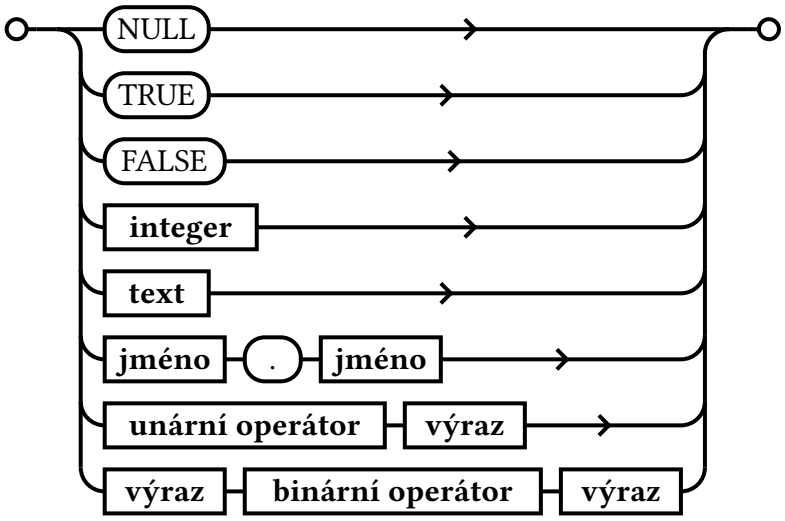
10 Vykonavatel

Vykonavatel pracuje s plánem provedení. Plán provedení má formu stromu, který je složen z takzvaných plánových vrcholů. V současné době existují následující typy plánových vrcholů:

- `None` - prázdná množina,
- `RangeScan` - množina řádků dané tabulky v určitém rozmezí, může využívat indexů,
- `Join` - kartézský součin dvou množin A a B, množiny A a B jsou podřazenými vrcholy,
- `Filter` - podmnožina množiny A, která splňuje určitou podmínku, množina A je podřazeným vrcholem,
- `Select` - výběr výrazů z množiny A reprezentované podřazeným vrcholem,
- `Delete` - vymazání řádků z tabulky množiny A reprezentované podřazeným vrcholem,
- `Update` - aktualizace řádků v tabulce množiny A reprezentované podřazeným vrcholem.

Vykonavatel prochází plán provedení od nejnižších listů a postupně vyhodnocuje všechny vrcholy. Po vyhodnocování vrcholu je vrácen vždy jeden nebo žádný řádek. S tímto řádkem poté vykonavatel vyhodnocuje vrcholy nadřazené.

10.1 Vyhodnocování výrazů



Obrázek 17: Zjednodušený diagram možných výrazů v plánu provedení v matdb. Obrázek autora

Jak je možné vidět v obrázku 17, vyhodnocování výrazů v plánu provedení je v celku jednoduché. Podle obrázku existuje pouze 8 druhů výrazů, 5 z čehož jsou doslovné výrazy (vyhodnocením se hodnota těchto výrazů nemění). Zbylé 3 druhy se dělí na 1 atomický (nedělitelný na další výrazy) a 2 druhy složité.

Výraz formy `[jméno].[jméno]` odkazuje prvním jménem na tabulku a druhým na sloupec stejné tabulky. Z tohoto důvodu si vykonavatel udržuje informace o původu jednotlivých řádků a sloupců během vykonávání plánu. Poté je vyhodnocení tohoto výrazu jen najít příslušný řádek v paměti.

V současnosti existují pouze dva unární operátory a to: `NOT` a `-`. Operátor `NOT` obrací pravdivostní hodnoty. Operátor `-` obrací znaménko číselných hodnot. V obou případech operátory pracují na vyhodnocení příslušného výrazu napravo od nich.

Binárních operátorů je 13. Jsou to následující operátory: `=`, `!=`, `>`, `>=`, `<`, `<=`, `+`, `-`, `*`, `/`, `%`, `AND`, a `OR`. Prvních 6 operátorů slouží k porovnávání hodnot. Je dobré vědět, že hodnoty jsou porovnávány v pořadí, ve kterém jsou seřazovány. V praxi toto znamená, že hodnota `NULL` je považována za menší než jakákoliv jiná hodnota. Následujících 5 operátorů je aritmických. Zde je dobré podoknout, že matdb nepodporuje jiná čísla než celá čísla, takže výsledek operací bude vždy zaokrouhlen. Poslední 2 operátory jsou logické, kombinují pravdivostní hodnoty. Stejně jako u unárních výrazů, i zde operátory pracují na příslušných výrazech, které jej obklopují.

10.2 Vyhodnocování jednotlivých vrcholů plánu

10.2.1 `None`

Vyhodnocení vrcholu `None` nikdy nevrátí žádný řádek. Vrchol `None` není nijak vyhodnocován, existuje pouze pro indikaci, že optimizátor usoudil, že některá část plánu nemůže nikdy žádné řádky vrátit.

10.2.2 `RangeScan`

`RangeScan` je založen na algoritmech z kapitoly 9.1. Při každém vyhodnocení načte další řádek z tabulky (pokud čte z indexové tabulky, pak také nachází korespondující řádek v klasické tabulce). Mezi vyhodnoceními si udržuje stav, což mu umožnuje najít další vrchol.

`RangeScan` nachází stromy tabulek v interní tabulce `matdb_tables` a, pokud je zadán dotaz se specifickou verzi, pak najde `matdb_tables` v interní tabulce `matdb_revs`.

10.2.3 `Join`

Každé vyhodnocení vrcholu `Join` je dalším řádkem z kartézského součinu jeho podřazených vrcholů. Podřazené vrcholy jsou vždy dva a rozdělují se na levý a pravý. Pravý podřazený vrchol může obsahovat výrazy, které se odkazují na levý podřazený vrchol. Levý však nemůže.

Vrchol si udržuje stav, kdy si ukládá výsledek levého vrcholu a pravý vrchol vyhodnocuje dokud bude vracet řádky. Až když pravý vrchol řádky vracet přestane, je levý vrchol znovu vyhodnocen a proces se opakuje.

10.2.4 `Filter`

Vrchol `Filter` má vždy právě jeden podřazený vrchol a jeden podmínkový výraz. Při každém vyhodnocení je vyhodnocen podřazený vrchol a výraz je vyhodnocen vzhledem k vrácenému řádku. Pokud je výraz vyhodnocen na hodnotu `TRUE`, pak je tento řádek vrácen. V opačném případě je podřazený vrchol znovu vyhodnocen. Pokud podřazený vrchol nevrátí řádek pak `Filter` také řádek nevrátí.

10.2.5 `Select`

Vrchol `Select` má vždy právě jeden podřazený vrchol a jeden nebo více výrazů. Při každém vyhodnocení je vyhodnocen podřazený vrchol a tyto výrazy jsou vyhodnoceny vzhledem k vrácenému řádku. `Select` vrací nový řádek, který je tvořen sloupci s jmény, které jsou odvozeny od příslušných výrazů. Název tabulky

v těchto sloupcích je `select`. Pokud podřazený vrchol nevrátí řádek pak `Select` také řádek nevrátí.

10.2.6 `Delete`

Vrchol `Delete` má vždy právě jeden podřazený vrchol a jméno tabulky, ze které je mazáno. Při každém vyhodnocení je vyhodnocen podřazený vrchol a vrácený řádek je vymazán z tabulky. Vrchol `Delete` vždy vrací původní řádek nezměněn. Pokud podřazený vrchol nevrátí řádek pak `Delete` také řádek nevrátí.

Při mazání řádku je nejdříve zkontrolováno, zda toto neporuší některý cizí klíč. Pro každou tabulku, která má cizí klíč na tabulku (toto je ukládáno v schématu tabulky v `matdb_schemas`), ze které je mazáno, je zkontrolováno v příslušném indexu, pokud některý řádek z této tabulky má řádek, který podle tohoto cizího klíče odkazuje na řádek který je právě mazán. Pokud ano, pak je vyhozena chyba a transakce je zrušena. V opačném případě je řádek uspěšně smazán z cílové tabulky i z příslušných indexových tabulek v souladu s kapitolou 9.2.3.

10.2.7 `Update`

Vrchol `Update` má vždy právě jeden podřazený vrchol, jeden či více SET klauzulí, a tabulku, která je aktualizavána. Při každém vyhodnocení je vyhodnocen podřazený vrchol a sloupce vráceného řádku jsou v tabulce aktualizovány v souladu se SET klauzulemi. Vrchol `Update` vždy vrací upravený řádek. Pokud podřazený vrchol nevrátí řádek, pak `Update` také řádek nevrátí.

Před aktualizací řádku je nejprve zkontrolováno, že nový řádek neporuší cizí klíč nebo podmínku `UNIQUE`. Porušení cizího klíče je kontrolováno víceméně stejně jako u `Delete`, ale nově musí být kontrolováno nejen, zda je na starou formu řádku odkazováno, ale také, zda nová forma řádku cizí klíč znovusplatní. Podmínka `UNIQUE` je zkontrolována použitím příslušného indexu. Pokud je jedna z těchto podmínek porušena, bude vyhozena chyba a transakce bude zrušena. V opačném případě je řádek nejprve smazán a poté znovu vložen do příslušné tabulky i do příslušných indexů v souladě s kapitolami 9.2.2 a 9.2.3.

11 Optimizátor

Optimizátor přijímá neoptimalizovaný plán provedení a vrací optimalizovaný plán provedení. V matdb je optimizátor založen na knihovně „egg“ (Willsey et al., 2021). Knihovna egg je založena na takzvaných e-grafech. E-grafy zaznamenávají rovnocennost výrazů. Na začátku optimalizace je e-graf tvořen všemy vstupnímy výrazy. V této chvíli má každý výraz v e-grafu pouze jeden rovnocenný výraz: sám sebe. Dále jsou aplikována přepisovácí pravidla, čímž výrazy v e-grafu získávají další rovnocenné výrazy. Když už nelze aplikovat žádná přepisovácí pravidla, je přepisovácí fáze dokončena a e-graf by měl obsahovat veškeré možné způsoby, jak původní výrazy vyjádřit. Poté je, podle nějaké metriky, jedno vyjádření vybráno, a to se stává optimalizovaným výrazem.

Jelikož má „egg“ svojí vlastní reprezentaci výrazů, která je optimalizovaná pro použití v e-grafech, musí být náš neoptimalizovaný plán provedení nejdříve přeložen na tuto reprezentaci a po optimalizaci musí být z této reprezentace znovu přeložen zpět.

Kromě přepisovacích pravidel, optimizátor také provádí jednoduchou analýzu, takzvané „skládání konstant.“ Během skládání konstant jsou výrazy, které pouze obsahují konstanty, pokud možno předčasně vyhodnocovány, a tyto výsledky jsou zavedeny do e-grafu jakožto rovnocenné výrazy. Například výraz `2 + 2` bude „složen“ na výraz `4`, který bude zaveden do e-grafu jakožto jeho rovnocenný výraz.

Matdb využívá následující přepisovací pravidla:

$$a = b \to b = a$$
$$a \wedge \mathrm{TRUE} \to a$$
$$a \wedge \mathrm{FALSE} \to \mathrm{FALSE}$$
$$a \wedge b \to b \wedge a$$
$$a + b = c \to a = c - b$$
$$a - b = c \to a = c + b$$
$$\neg(a = b) \to a = a \neq b$$
$$\neg(a > b) \to a \leq b$$
$$\neg(a \geq b) \to a < b$$
$$\neg(a < b) \to a \geq b$$
$$\neg(a \leq b) \to a > b$$

$$\text{Filter}(\text{FALSE}, a) \rightarrow \text{None}$$

$$\text{Filter}(\text{TRUE}, a) \rightarrow a$$

$$\text{Filter}(a, \text{Filter}(b, c)) \rightarrow \text{Filter}(b, \text{Filter}(a, c))$$

$$\text{Filter}(a \wedge b, c) \rightarrow \text{Filter}(a, \text{Filter}(b, c))$$

$$\text{Filter}(a, \text{Join}(b, c)) \rightarrow \text{Join}(b, \text{Filter}(a, c))$$

$$\text{Join}(a, b) \rightarrow \text{Join}(b, a), \text{pouze pokud } b \text{ nezmiňuje } a$$

Pokud má tabulka t index nebo primární klíč t_i s indexovými výrazy ve tvaru $(t.s_1, t.s_2, ..., t.s_{n-1}, t.s_n)$, pak přibývají následující přepisovací pravidla:

$$t.s_1 = E_1 \wedge t.s_2 = E_2 \wedge ... \wedge t.s_{m-2} = E_{m-2} \wedge t.s_{m-1} = E_{m-1} \wedge t.s_m = E_m$$
$$\downarrow$$
$$\text{RangeScan}(t_i, (E_1, E_2, ..., E_{m-2}, E_{m-1}, E_m), (-\infty, \infty))$$

$$t.s_1 = E_1 \wedge t.s_2 = E_2 \wedge ... \wedge t.s_{m-2} = E_{m-2} \wedge t.s_{m-1} = E_{m-1} \wedge t.s_m < E_m$$
$$\downarrow$$
$$\text{RangeScan}(t_i, (E_1, E_2, ..., E_{m-2}, E_{m-1}), (-\infty, E_m))$$

$$t.s_1 = E_1 \wedge t.s_2 = E_2 \wedge ... \wedge t.s_{m-2} = E_{m-2} \wedge t.s_{m-1} = E_{m-1} \wedge t.s_m \leq E_m$$
$$\downarrow$$
$$\text{RangeScan}(t_i, (E_1, E_2, ..., E_{m-2}, E_{m-1}), (-\infty, E_m\rangle)$$

$$t.s_1 = E_1 \wedge t.s_2 = E_2 \wedge ... \wedge t.s_{m-2} = E_{m-2} \wedge t.s_{m-1} = E_{m-1} \wedge t.s_m > E_m$$
$$\downarrow$$
$$\text{RangeScan}(t_i, (E_1, E_2, ..., E_{m-2}, E_{m-1}), (E_m, \infty))$$

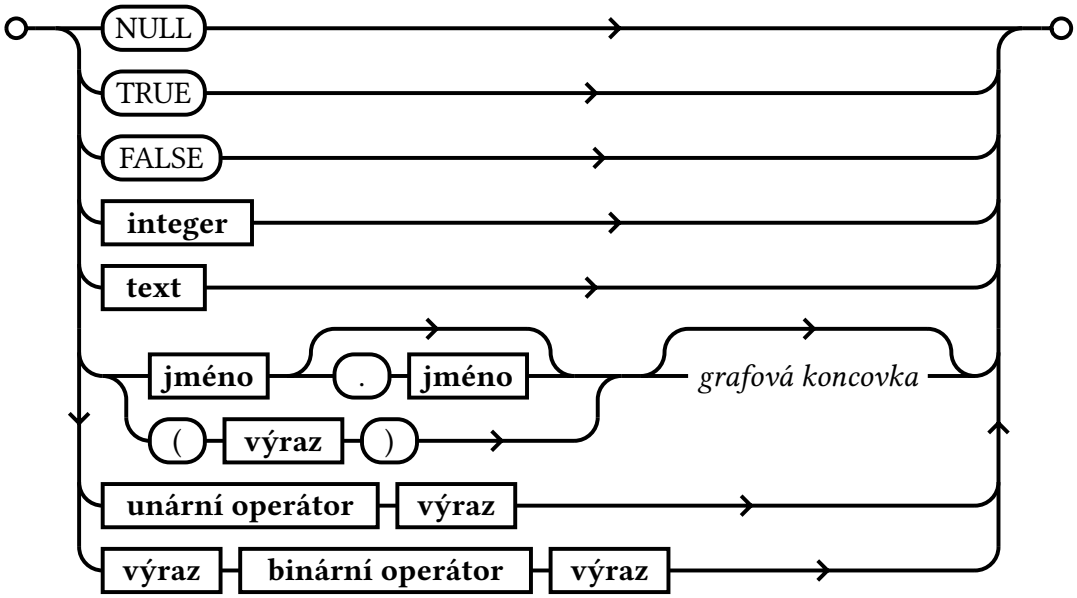
$$t.s_1 = E_1 \wedge t.s_2 = E_2 \wedge ... \wedge t.s_{m-2} = E_{m-2} \wedge t.s_{m-1} = E_{m-1} \wedge t.s_m \geq E_m$$
$$\downarrow$$
$$\text{RangeScan}(t_i, (E_1, E_2, ..., E_{m-2}, E_{m-1}), \langle E_m, \infty))$$

kde $0 < m < n$.

12 Syntaktický analyzátor

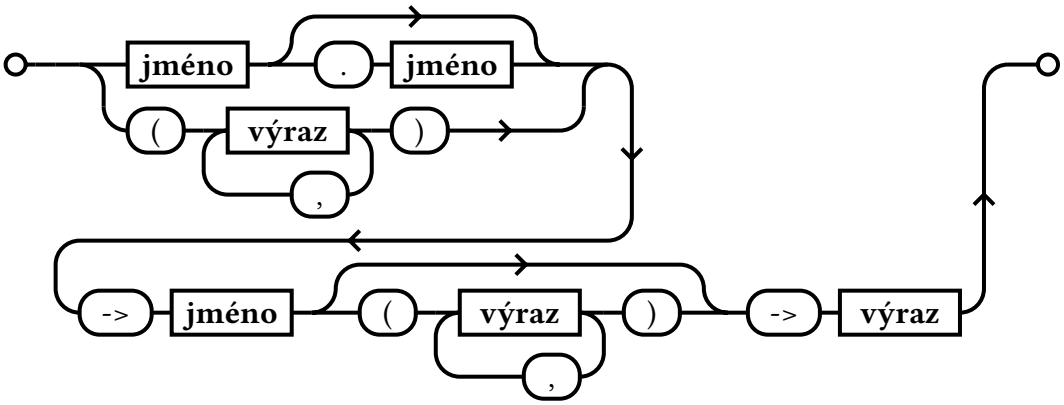
Syntaktický analyzátor přijímá SQL kód a vrací abstraktní syntaktický strom. Tento abstraktní syntaktický strom je v některých případech dále přetvářen na plán provedení. Před samotnou syntaktickou analýzou je prováděna lexikální analýza. Lexikální analýza přetváří text na proud lexémů, které jsou poté použity při syntaktické analýze.

12.1 Výrazy



Obrázek 18: Diagram syntaxe vyrazů. Obrázek autora

V obrázku 18 lze vidět všechny možné syntaxe výrazů. Většina z těchto výrazů byla již zmíněna v kapitole 10.1. Nově přibývají výrazy v závorkách, které upravují pořadí vyhodnocování a grafové výrazy. Grafové výrazy jsou bez rozporu nejsložitějšímy výrazy.



Obrázek 19: Diagram celé syntaxe grafového výrazu. Obrázek autora

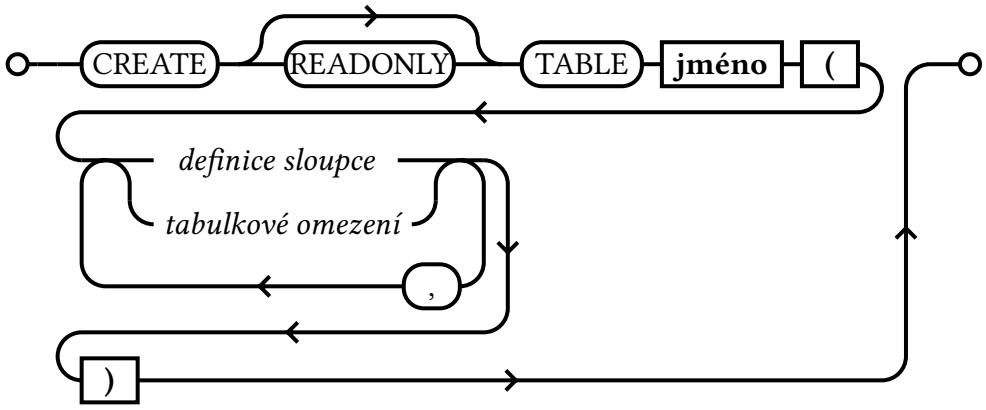
Grafový výraz začíná jednou ze dvou možností: sloupcový výraz ve tvaru `[jméno].[jméno]`, nebo výčtem plnohodnocenných výrazů v závorkách. Následuje šipka `->` představující hranu grafu, po které následuje jméno tabulky, kde se nachází cílový vrchol pomysleného grafu. K najití vrcholu v tabulce je použit jíž zmíněný první výraz (popřípadě první výrazy). Pokud je jméno tabulky následované výčtem výrazů v závorkách, pak jsou spárovány s prvními výrazy. V opačném případě je s prvními výrazy spárován primární klíč tabulky. Následuje další šipka `->` a poté výraz, jehož hodnota bude hodnotou celého grafového výrazu.

V praxi jsou grafové výrazy překládány do klauzulí příkazu `SELECT`. V žádném jiném příkazu nejsou povoleny.

12.2 Příkazy

Úryvek SQL kódů je tvořen libovolným počtem příkazů. Tyto příkazy jsou vždy ukončeny středníkem `;`. Z tohoto důvodu není středník součástí následujících diagramů.

12.2.1 CREATE TABLE



Obrázek 20: Diagram zkrácené syntaxe příkazu CREATE TABLE. Obrázek autora

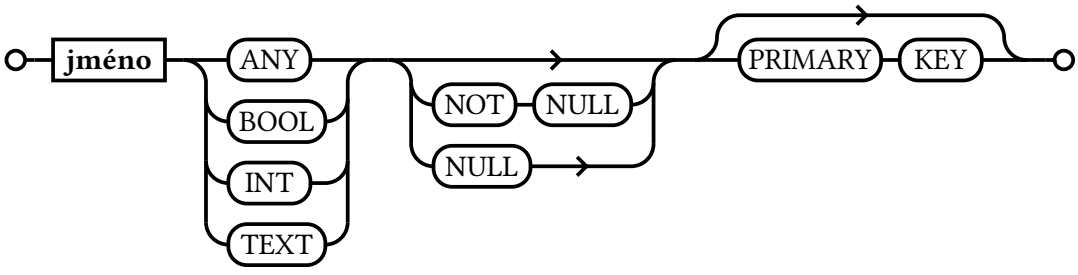
Příkaz CREATE TABLE začíná klíčovými slovy CREATE TABLE (popřípadě READONLY), pokračuje jménem tabulky, a dále definicí tabulky v závorkách.

I když je klauzule READONLY součástí syntaxe, existuje pouze pro interní použití (viz kapitola 8.1.2): nikdy není platné, aby jí použil uživatel.

Příkaz CREATE TABLE nevytváří plán provedení. Když je příkaz CREATE TABLE vykonán, stane se následující:

1. je vytvořena nová prázdná listová B-stromová stránka,
2. tato stránka je přidána do interní tabulky `matdb_tables`,
3. schéma tabulky je přidáno do interní tabulky `matdb_schemas`,
4. podobný proces je opakován pro vytvoření veškerých indexových tabulek,
5. do schématu každé tabulky, která je zmíněna v cizích klíčích je přidána zpětná reference k této tabulce.

12.2.1.1 Definice sloupce

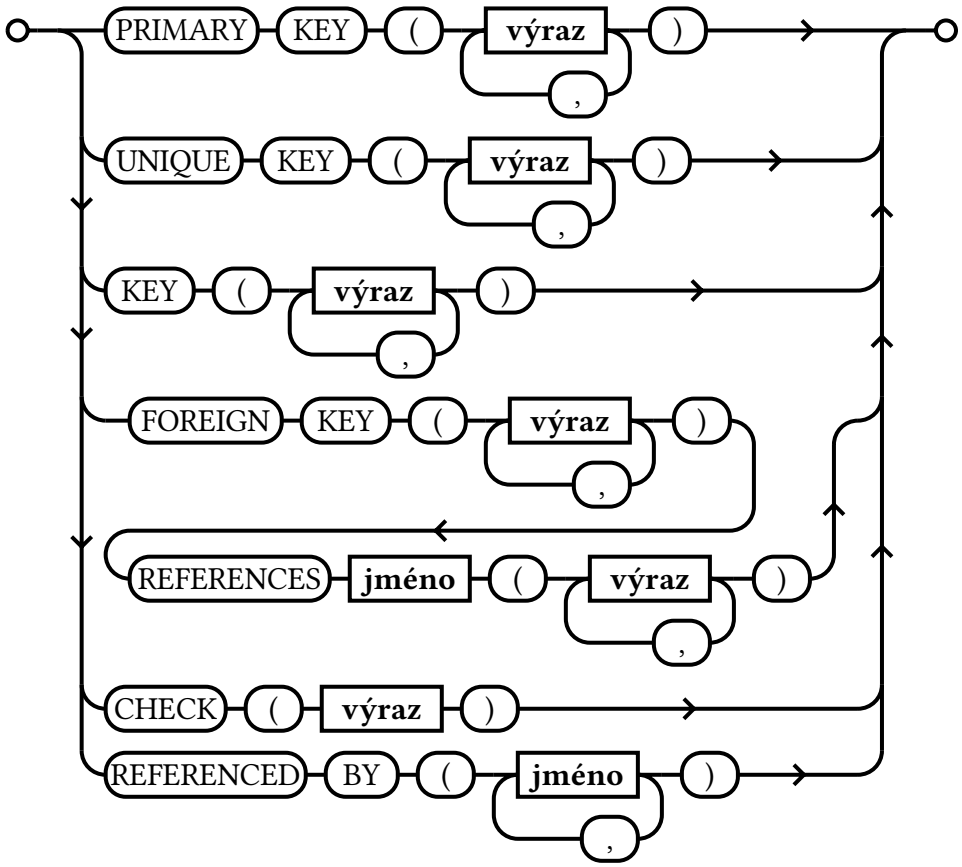


Obrázek 21: Diagram syntaxe definice tabulkového sloupce. Obrázek autora

Definice sloupce začíná jménem sloupce a pokračuje jeho datatypem. Datatyp ANY znamená, že datatyp sloupce není kontrolován. Dále pokračuje volitelná klau-

zule o nulovatelnosti a klauzule `PRIMARY KEY`, která, pokud je přítomna, označuje tento sloupec za primární klíč. Tabulka může mít definován pouze jeden primární klíč.

12.2.1.2 Omezení



Obrázek 22: Diagram syntaxe tabulkových omezení. Obrázek autora

Existuje 5 druhů omezení:

1. primární klíč - definuje primární klíč tabulky na výčet výrazů (znovu platí, že primární klíč může být definován pouze jeden),
2. unikátní klíč - ustanovuje, že každému výčtu hodnot výčtu výrazů může odpovídat pouze jeden řádek tabulky (toto omezení také vytváří index na daný výčet výrazů),
3. prostý klíč - vytváří index na daný výčet výrazů,
4. cizí klíč - ustanovuje, že prvnímu výčtu výrazů musí odpovídat druhým výčtem výrazů řádek v dané tabulce (toto vytváří index na první výčet výrazů ve vytvářené tabulce, cizí tabulka musí mít druhý výčet výrazů jakožto unikátní klíč),

5. `CHECK` - jednoduše kontroluje, že daná podmínka je pravdivá pro každý řádek tabulky,
6. `REFERENCED BY` - výčet tabulek, které cizími klíči odkazují na tuto tabulku. Slouží pouze pro interní použití. Zadání uživatelem není nikdy platné.

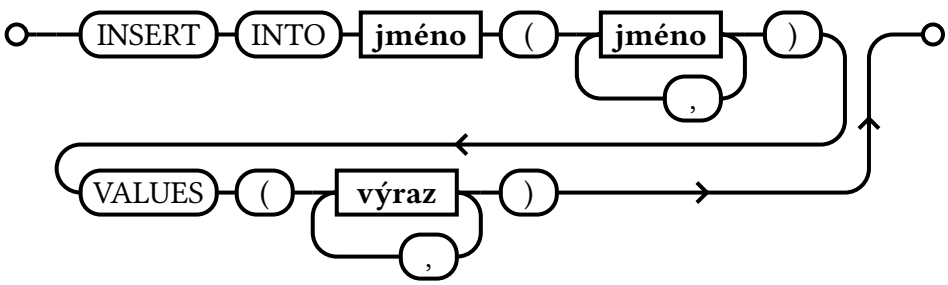
12.2.2 `DROP TABLE`

Obrázek 23: Diagram syntaxe příkazu `DROP TABLE`. Obrázek autora

Z hlediska syntaxe je tento příkaz nejjednodušší. Po klíčových slovech `DROP TABLE` pokračuje jen jméno tabulky.

Při vykonávání je zkontrolováno, zda existuje cizí klíč na tuto tabulku v jiné tabulce. Pokud ne, pak jsou veškeré stromy hlavní tabulky i indexových tabulek smazány a všechny tyto tabulky jsou odstraněny z interních tabulek.

12.2.3 `INSERT`

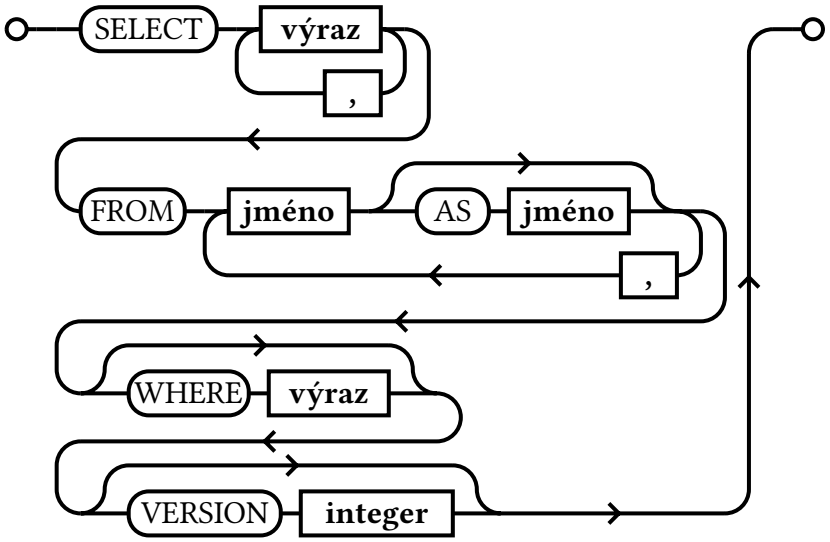


Obrázek 24: Diagram syntaxe příkazu `INSERT`. Obrázek autora

Po klíčových slovech `INSERT INTO` následuje jméno tabulky, do které bude řádek vložen. Poté v závorkách následuje výčet jmen sloupců, které budou v rámci tohoto příkazu nastaveny a po klíčovém slově `VALUES` jejich hodnoty (respektive výrazy, které budou vyhodnoceny na jejich hodnoty).

Pokud nějaký sloupec není zmíněn, pak bude jeho hodnota nastavena na `NULL`.

12.2.4 SELECT



Obrázek 25: Diagram syntaxe příkazu SELECT. Obrázek autora

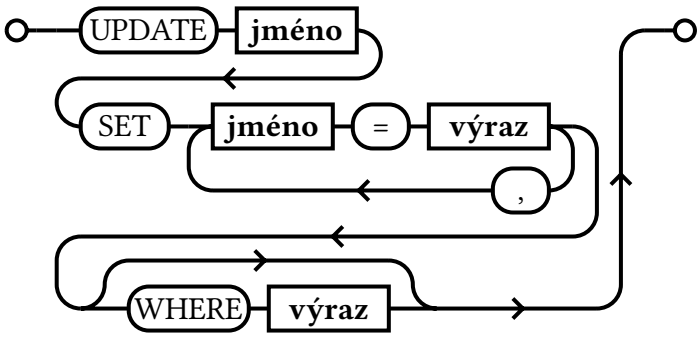
Po klíčovém slovu SELECT následuje výčet výrazů, které budou tvořit sloupce výsledné tabulky. Následuje povinná klauzule FROM, která určuje tabulky, ze kterých budou pocházet řádky, na těchto tabulkách je prováděn kartézský součin.

Následuje volitelná klazule WHERE s výrazem, který je podmínkou pro obsažení daného řádku ve výsledné tabulce.

Následuje volitelná klazule VERSION s číslem verze databáze, ze které bude tento dotaz zodpovězen. Čísla verzí databáze je možné zjistit v interní tabulce `matdb_revs` (viz kapitola 8.1.3).

Příkaz SELECT vytváří plán provedení a je poté vykonán podle kapitoly 10.

12.2.5 UPDATE

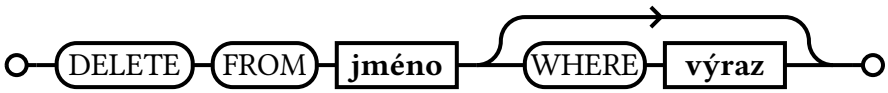


Obrázek 26: Diagram syntaxe příkazu UPDATE. Obrázek autora

Po klíčovém slovu `UPDATE` následuje jméno tabulky, klíčové slovo `SET` a výčet sloupců k aktualizaci a výrazů, kterých výsledné hodnoty budou použity jako nové hodnoty sloupců. Dále pak volitelná klazule `WHERE` s podmínkou, která tyto aktualizace omezuje na řádky splňující tuto podmínku.

Příkaz `UPDATE` vytváří plán provedení a je poté vykonán podle kapitoly 10.

12.2.6 `DELETE`



Obrázek 27: Diagram syntaxe příkazu `DELETE`. Obrázek autora

Po klíčových slovech `DELETE FROM` následuje jméno tabulky a volitelná klazule `WHERE` s podmínkou, která mazání omezuje na řádky splňující tuto podmínku.

Příkaz `DELETE` vytváří plán provedení a je poté vykonán podle kapitoly 10.

13 Použití

V současné době existují dva způsoby, jak použít matdb. Prvním způsobem je skrz API knihovny. Druhý způsob je malý CLI program založený na této knihovně, který je určený k prozkoumávání databázových souborů matdb.

API knihovny matdb má pouze tři funkce:

1. `open` - otevře databázový soubor, popřípadě vytvoří nový,
2. `run` - vykoná úryvek SQL kódu, vrátí hodnoty, které byly vráceny posledním SQL příkazem (pokud byly nějaké vráceny), možnost použití vložených hodnot pomocí syntaxe `?[číslo]` (opatření proti SQL injekcím),
3. `finish_tx` - dokončí transakci. Nová transakce je automaticky započtena otevřením databáze (nebo ukončením předešlé transakce) a není dokončena, dokud není použita tato funkce.

Příklad použití API knihovny:

```
let mut db = Db::open("db.matdb".into())?;

db.run("CREATE TABLE messages (
  id INT PRIMARY KEY,
  body TEXT NOT NULL
);", Vec::new())?;

db.run("INSERT INTO messages (id, body) VALUES (?0, ?1);",
  vec![
    Value::Int(1),
    Value::String("Hello, matdb!".to_string())
])?;

db.finish_tx()?;

println!("{:?}",
  db.run(
    "SELECT body FROM messages WHERE id = 1;",
    Vec::new()
  )?.unwrap().1[0][0]
);
```

CLI program přijímá jedinný argument a to jméno databázového souboru. Tento argument funguje stejně jako funkce `open`. Výstup příkazů je ve formátu CSV.

Příklad použití CLI programu v režimu REPL:

```
$ matdb fiction.matdb
matdb> SELECT
  f2.id,
  author->fiction AS f2(f2.author)->f2.title
FROM fiction WHERE title = 'The Little Prince';
f2.id,f2.title
1226708,The Little Prince
1043167,Airman's Odyssey
1099831,Night Flight
1193620,terre des hommes
1809010,The Little Prince
matdb>
```

CLI program je také možné použít jako součást unixové roury:

```
$ echo "SELECT id, title FROM fiction \
  WHERE author = 'Wilde, Oscar';" | matdb fiction.matdb
fiction.id,fiction.title
1054477,Contes
1056532,Das Bildnis des Dorian Gray
1061249,Der glückliche Prinz und andere Märchen
1062791,Der Sozialismus und die Seele des Menschen
1108978,Savile
1182776,The Complete Works of Oscar Wilde
1204243,De Profundis
1269478,"Vera, or The Nihilists"
1269855,Il ritratto di Dorian Gray
1270197,"Penna, matita e veleno"
1283758,An Ideal Husband
```

Databázový soubor `fiction.matdb` je založen na databázi knihovny LibGen a obsahuje tabulku popisující 1 million knih. Je k dispozici komprimovaný algoritmem bzip3 z https://drive.google.com/file/d/1O6ksqYr-c48pu8wakcXTRQ_Q9ijv8CvZ/.

Závěr

Matdb je systém řízení báze dat určený pro použití v koncově uživatelských programech. Lze jej použít skrz knihovnu nebo CLI program. Disponuje dotazovacím jazykem podobným SQL, schopností dotazování napříč časem, grafovými výrazy, vlastnostmi ACID, optimizátorem dotazů, a B-stromovým úložištěm, které je zcela obsaženo v jedinném souboru.

Samozřejmě, vždy existují další vylepšení a optimalizace, které by mohly být implementovány. Nejnápadněji:

- (selektivní) mazání historie,
- podpora více výrazů,
- agregátní dotazy,
- chybějící klazule příkazu `SELECT` (například `GROUP BY`, `ORDER BY`, `LIMIT`, apod.),
- (rekurzivní) obecné tabulkové výrazy,
- rozšíření grafových dotazů na rámec celého rozmezí DFS,
- množinové funkce (například `UNION`),
- podpora měnícího se schématu tabulek (příkazy `CREATE INDEX`, apod.),
- více datatypů (časové razítka, desetinná čísla, pole, atd.),
- dále pak optimalizace B-stromového úložiště (důmyslnější rebalancovací strategie, varianta stromů optimalizována pro indexové tabulky, atd.).

Seznam použité literatury

ARANGODB, 2024. *Multi-Model Database*. 2024-. Dostupné z: https://arangodb.com/multi-model-db/. [citováno 2024-12-16].

BONCZ, Peter, 2022. *The (sorry) State of Graph Database Systems*. 2022-03-30. Dostupné z: https://www.youtube.com/watch?v=aDoorU4X6Jk. [citováno 2024-12-16].

CODD, Edgar Frank, 1982. Relational database: a practical foundation for productivity. *Communications of the ACM*, roč. 25, č. 2, s. 109–117. Dostupné z: https://doi.org/10.1145/358396.358400.

CONNOLLY, Thomas a Carolyn BEGG, 2015. *Database Systems: A Practical Approach to Design, Implementation, and Management*. Sixth Edition, Global Edition. Edinburgh Gate, Harlow, Essex CM20 2JE, England: Pearson Education Limited. s. 97–102. ISBN 978-1-292-06118-4.

HASHEMI, Mazdak, 2017. *The Infrastructure Behind Twitter: Scale*. 2017-01-19. Dostupné z: https://blog.x.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale. [citováno 2024-12-16].

HIPP, D. Richard, 2024. *Database File Format*. 2024-. Dostupné z: https://www.sqlite.org/fileformat.html. [citováno 2025-03-15].

K., Amit, 2021. *Facebook's Graph Database: TAO*. 2021-02-17. Dostupné z: https://www.linkedin.com/pulse/facebooks-graph-database-tao-amit-kumar. [citováno 2024-12-16].

MARCHUKOV, Mark, 2013. *TAO: The power of the graph*. 2013-06-25. Dostupné z: https://engineering.fb.com/2013/06/25/core-infra/tao-the-power-of-the-graph/. [citováno 2024-12-16].

RED GATE SOFTWARE LTD, 2024. *DB-Engines Ranking*. 2024-12-. Dostupné z: https://db-engines.com/en/ranking. [citováno 2024-12-16].

SIKHINAM, Tharun, 2019. *How does the performance of a graph database such as Neo4j compare to the performance of a relational database such as Postgres?*. 2019-11-12. Dostupné z: https://courses.cs.washington.edu/courses/csed516/20au/projects/p06.pdf. [citováno 2024-12-16].

STEGEMAN, John, 2023. *Native vs. Non-Native Graph Database*. 2023-05-08. Dostupné z: https://go.neo4j.com/rs/710-RRC-335/images/Neo4j_Top5_UseCases_Graph%20Databases.pdf. [citováno 2024-12-16].

SURREALDB LTD., 2024. *The ultimate multi-model database*. 2024-. Dostupné z: https://surrealdb.com/features/multi-model-database. [citováno 2024-12-16].

WEBBER, Jim a Ian ROBINSON, 2017. *The Top 5 Use Cases of Graph Databases: Unlocking New Possibilities with Connected Data*. 2017-. Dostupné z: https://go.neo4j.com/rs/710-RRC-335/images/Neo4j_Top5_UseCases_Graph%20Databases.pdf. [citováno 2024-12-16].

WILLSEY, Max; Chandrakana NANDI; Yisu Remy WANG; Oliver FLATT; Zachary TATLOCK et al., 2021. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages*, roč. 5, č. POPL, s. 1–29. Dostupné z: https://doi.org/10.1145/3434304.

Seznam obrázků

Seznam algoritmů