Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Optimalizované databázové aplikace pro vyhledávání

Tomáš Roch

Bakalářská práce
2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš Roch**
Osobní číslo:	**I17133**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Optimalizované databázové aplikace pro vyhledávání**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem práce je využití databázových indexů při optimalizaci databázových aplikací. Součástí teoretické části bude podrobný popis jednotlivých kategorií databázových indexů s ukázkami na konkrétních příkladech. Práce se bude zaměřovat na různé databázové systémy, přičemž pro praktickou část práce bude vybrán jeden databázový systém, na kterém budou demonstrovány konkrétní aplikované mechanismy sloužící pro optimalizaci aplikace. Součástí práce jsou pracovní postupy při optimalizaci databázové aplikace.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací: **–**
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

KROENKE, David a David J. AUER. *Databáze*. Přeložil Jakub GONER. Brno: Computer Press, 2015. ISBN 978-80-251-4352-0.

LAURENČÍK, Marek. *SQL: podrobný průvodce uživatele*. Praha: Grada Publishing, 2018. Průvodce (Grada). ISBN 978-80-271-0774-2.

POKORNÝ, Jaroslav a Michal VALENTA. *Databázové systémy*. Praha: České vysoké učení technické v Praze, 2013. ISBN 978-80-01-05212-9.

STRATE, Jason a Grant FRITCHEY. *Expert performance indexing in SQL server*. Second edition. New York: Apress, [2015].

Vedoucí bakalářské práce: **Ing. Monika Borkovcová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. listopadu 2019**
Termín odevzdání bakalářské práce: **7. května 2020**

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 17. prosince 2019

Prohlašuji:

Práci s názvem Optimalizované databázové aplikace pro vyhledávání jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.
Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 7. 8. 2021

Tomáš Roch

PODĚKOVÁNÍ

Děkuji vedoucí mé bakalářské práce Ing. Monice Borkovcové, Ph.D. za cenné rady, věcné připomínky, ochotu a vstřícnost při konzultacích, které mi pomohly tuto práci zkompletovat.

ANOTACE

Cílem této práce je využití databázových indexů při optimalizaci databázových aplikací. Práce se v teoretické části zaměřuje na různé databázové systémy a možnosti databázových indexů. V praktické části je porovnána rychlost před a po optimalizaci relačního databázového systému pomocí indexů. Následně je srovnána rychlost mezi jednotlivými databazovými systémy a složitost implementace.

KLÍČOVÁ SLOVA

Indexování, optimalizace, vyhledávání, Oracle, MySQL, Microsoft SQL

TITLE

Database applications optimization for search

ANNOTATION

The aim of this thesis is to use database indexes to optimize database applications. In the theoretical part, thesis focuses on different database systems and the possibilities of database indexes. In the practical part, the speed is compared before and after the process of optimization relational database system using indexes. Then, the speed of the specific database systems and complexity of implementation is compared.

KEYWORDS

Indexing, optimization, searching, Oracle, MySQL, Microsoft SQL

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM GRAFŮ

SEZNAM ZKRATEK

SQL	Structured Query Language
DDL	Data Definition Language
DML	Data Manipulation Language
ANSI	American National Standards Institute
ISO	International Organization for Standardization
COBOL	Common Business-Oriented Language
BLOB	Binary Large Object
JSON	JavaScript Object Notation
DBMS	DataBase Management System
XML	Extensible Markup Language
RMD	Relační modelování dat
ACID	Atomicicity, Consistency, Independence, Durability
RDBMS	Relational Database Management System
ID	Identification
CIA	Central Intelligence Agency
JSON	JavaScript Object Notation
BSON	Binary JSON
BASIC	Beginner's All-purpose Symbolic Instruction Code
RAM	Random Access Memory
RAC	Real Application Clusters
UTF-8	Unicode Transformation Format – 8-bit
SGA	System Global Area
PGA	Program Global Area
TCP/IP	Transmission Control Protocol/Internet Protocol
VIA	Virtual Interface Adapter
FAT	File Allocation Table
NTFS	New Technology File Systém
LSN	Log Sequence Number
CAP	Consitenty Availability Partition Tolerance
SSD	Solid State Drive
API	Application Programming Interface
IMDb	Internet Movie Database

ÚVOD

Relační databázové systémy jsou stále hojně zastoupeny v databázových aplikacích tam, kde je nutné dodržet pevnou strukturu dat. Stále jsou důležitou součástí například bankovnictví a pojišťovnictví a pravděpodobně ještě dlouhou dobu budou. Nároky na dnešní databázové systémy stále stoupají. S každým rokem přibývá stále více a více dat a požadavky na rychlost načítání dat také nezůstávají konstantní, ale je na ně vyvíjen velký tlak. Potřeby organizací mající nutnost využití relačních databázových systémů se potýkají s rostoucími požadavky na odezvu těchto databází. Nicméně vývojáři stále přicházejí s novými inovacemi a snaží se s náročnou dobou držet krok. Možností optimalizace databázového systému je mnohem více, než tomu bylo v minulosti, a proto je také velká snaha o zautomatizování procesů optimalizace. V dnešní době již optimalizátory monitorují většinu procesů v databázovém systému a podle toho sami dokáží navrhnout plán optimalizace, a dokonce ho i sami implementují. I přes to je dobré optimalizační techniky znát a vědět, jak je použít. I když riziko chyby optimalizátorů je díky implementovaným sofistikovanějším metodám minimální, stále se může objevovat malé procento speciálních případů užití, na které optimalizátor není připravený a je lepší řídit optimalizaci na míru danému případu užití.

V úvodu teoretické části bude stručně probrána historie relačních databází a budou vymezeny základní pojmy pro lepší porozumění a orientaci v dané problematice. Následně budou zmíněny stále častěji používané NoSQL databáze. V další části budou popsány relační databázové systémy od společností Microsoft, Oracle a MySQL. V této části se práce soustředí na jejich historii, architekturu a popisuje novinky z jejich posledních verzí. Poslední popisná část poukazuje na možnosti optimalizace a optimalizační techniky se zaměřením na indexování.

Výstupem praktické části budou výsledky prezentující vliv indexů na výkon dotazů pod zatížením různého množství dat. Pracovní postup celého měření bude podrobně popsán. Dotazy budou testovány s různou selektivitou a bude tak zjištěno, jaký vliv na výkon dotazu selektivita má. Dalším úkolem bude porovnat výsledky mezi jednotlivými relačními databázovými systémy a zjistit, zda je v tomto směru nějaký zvýhodněn oproti ostatním. Čtenář bude také moci sledovat, jak se liší syntaxe SQL jazyka u různých databázových systémů. Zajímavé bude také sledovat, jak si povede MySQL s bezplatnou licencí, oproti svým placeným konkurentům.

1 DATABÁZE

1.1 Historie

Práce s daty by měla být efektivní a rychlá. Data je potřeba někam ukládat a sloužit k tomu mohou například kartotéky, archivy, knihovny nebo seznamy. S přibývajícím množstvím dat jsou kladeny vyšší nároky nejen na práci se samotnými daty, ale i na jejich třídění s důrazem na jejich rychlé vyhledávání. S rostoucím množstvím dat začala snaha o zautomatizování procesu třídění a na konci 19. století se objevil první použitelný pokus o zpracování dat strojem. Jednalo se o systém děrných štítků, které byly zpracovávány na elektromechanických strojích. Ten byl využit ke sčítání lidu ve spojených státech amerických v roce 1890. V padesátých letech minulého století se s rozvojem samočinných počítačů objevila myšlenka jejich využití právě pro zpracování hromadných dat. Nevýhodou fyzicky ukládaných dat je manuální práce s nimi a větší množství těchto dat zabírá také mnoho fyzického místa.

Souborový přístup se stal základem pro zpracování dat, která byla ukládána formou záznamů na vnějších pamětech. Byl to pokrok, ale k databázovým technologiím měl tento přístup daleko. Mezi nevýhody zpracování dat na souborové úrovni patří duplicity uložených dat a zároveň kopie těchto dat nemusí být stejné, izolace dat v nekoordinovaných souborech, problémy při aktualizaci dat více uživateli a bezpečnost z hlediska přístupu k souborům. Hlavní nevýhodou bylo, že mezi záznamy souboru byly velmi omezené možnosti vytváření vazeb mezi nimi. Například vazby mezi knihami, knihovnou a výpůjčkami lze řešit pouze programově s omezeními použitého programovacího jazyka. Cílem se tedy stalo vytvořit novou technologii přístupu k souborům, která nebude trpět uvedenými nevýhodami. Data se tak začala zpracovávat v komplikovanější struktuře zvané databáze. Začaly vznikat první databázové modely. [2][3]

Hierarchický model byl první databázový model, který vznikl v šedesátých letech minulého století, byl předchůdcem relačního modelu databáze. Koncepce hierarchického modelu byla založena na struktuře, kdy jsou data uspořádána ve vztazích rodič – potomek. Nahrazen byl vylepšeným síťovým modelem, kde byly navíc přidány vztahy, kdy rodič může mít více potomků a naopak. Oba modely využívají vzájemně propojené soubory dat.

Koncepce relační databáze se objevila už v roce 1970. Přišel s ní E.F. Codd a ve výzkumné laboratoři IBM se začal vyvíjet její prototyp. Nejprve se jednalo o teoretický přístup, který vrátil data zpátky do fyzicky izolovaných souborů a zároveň přinesl nový pohled na logickou úroveň dat. Relační databáze mají data uložena v tabulkách, kde jsou uspořádány do sloupců a řádků.

Data ve sloupcích mohou být pro některé tabulky společné a tvoří tak jednotlivé relace mezi tabulkami. Trvalo přibližně 10 let, než se tato technologie z hlediska výkonu v reálném prostředí dostala na úroveň hierarchického a síťového modelu. V mezičase se vyvíjela teorie relačního modelu, na které stojí teoretický základ databázových systémů až do současnosti.

Nezměnil na tom nic ani příchod objektových databází v roce 1989. Těchto databázových systémů sice v současnosti existuje několik desítek, ale zůstaly za očekáváním. Jejich výkon a funkce nedosáhly úrovně, aby mohly konkurovat relačním systémům. Z praxe poté vzešla tzv. objektově relační databáze kombinací relační a objektové databáze. Protože je tvořena na základě relací, tak bude-li se hovořit o databázové technologii, bude tím myšlena technologie relační.

Koncem 90. letech se začal objevovat XML databázový model založený na značkovacím jazyce XML. Ten se dělí na dvě základní architektury: databáze zpřístupňující XML data, ten používají relační databáze pro ukládání dat a druhý je nativní XML databáze. To je soubor samostatných nástrojů, který pro ukládání a indexaci dat používá speciální techniky. [3][2]

1.2 Relační model dat

Jak již bylo zmíněno výše, relační model dat reprezentují tabulky. Řádky reprezentují jednotlivá data a v záhlaví jsou jména sloupců – atributů. Každý řádek tabulky obsahuje nějaká data a tvoří tak jeden vztah. E.F. Codd našel podobnost tohoto pojetí pohledu v matematice. Tam se množina prvků nazývá relace a relační databáze se tak dá popsat jako množina relací. Mezi tabulkou a relací jsou jisté menší rozdíly. Zatímco atribut zahrnuje celou doménu, sloupec u tabulky pouze tu aktuální. Dalšími rozdíly jsou, že relace neobsahuje duplicitní n-tice a v relaci také nezáleží na pořadí řádků. Srovnání pojmů tabulkového modelu a RMD je znázorněno v tabulce 1.

Tabulka 1 – Srovnání pojmů tabulkového modelu a RMD

Zdroj: Přepracováno podle (Databázové systémy, 2020)

tabulka	relace
záhlaví tabulky	schéma relace
řádek	n-tice
sloupec	atribut (nepřesně)
počet sloupců	arita (stupeň) relace

Tabulka je tedy pohled na databázovou relaci ať už v tištěné podobě nebo na obrazovce.

Relační modelování dat (RMD) chápeme jako navrhování relací (tabulek). V souvislosti s SQL lze také použít pojem tabulkový model dat. [3]

RDBMS

Relační model je základ pro relační systém řízení dat, anglicky Relational DataBase Management System (RDBMS). RDBMS zajišťuje pohyb dat v databázi, ukládání dat a vracení dat, aby s nimi potom mohlo být manipulováno aplikacemi. RDBMS rozlišuje následující typy operací:

- Logické operace – V tomto případě specifikuje aplikace, jaký obsah je vyžadován. Například může vyžadovat údaj z tabulky nebo přidání záznamu do tabulky.
- Fyzické operace – RDBMS určuje, jak by se měly věci provádět. Například potom, co aplikace podá dotaz na tabulku, databáze může použít index k vyhledání požadovaných dat, načte data do paměti a další akce, než vrátí výsledek uživateli. RDBMS ukládá a vrací tak, aby fyzické operaci byly transparentní databázovým aplikacím.

[7]

Transakční mechanismus

Pro většinu organizací jsou data to nejcennější, co mají a práce s nimi je pro ně každodenní záležitostí. S daty může pracovat více lidí najednou a je očekáváno, že databázový systém umožní přístup k datům rychle, spolehlivě a bezpečně. Proto jsou důležité v databázovém systému dvě komponenty. Souběžné zpracování a zotavení z chyb. Ty zajistí konzistenci databáze i po skončení programu. Tyto vlastnosti jsou skryty pod pojmem transakce. Ta zachová konzistenci databáze i v případě, že je dočasně narušena. Transakce je tedy soubor posloupnosti operací jako čtení, zápis, výpočet a provede se buď řádně celá nebo se neprovede vůbec.

Příkladem transakce může být převod peněz v bance. Ty se převádí z jednoho účtu na druhý a zároveň se musí přepočítat zůstatky obou účtů a případně dalších s tím souvisejících operací.

Transakce jsou úzce spojeny s pojmem ACID. Jsou to čtyři vlastnosti, které musí transakce splňovat. Jedná se o tyto vlastnosti:

- Atomicita (z angl. atomicity) – Transakce je chápana pouze jako celek a musí proběhnout celá nebo vůbec. Transakce je složena z více menších atomických operací. Jedná se tedy o zotavení z chyb příkazem `ROLLBACK`, pokud skončí některá z menších operací chybou nebo se nedokončí.

- Konzistence (z angl. consistency) – Databáze musí být konzistentní před provedením transakce i po provedení transakce.
- Nezávislost (z angl. isolation) – Operace prováděné v transakci jsou nezávislé a nejsou viditelné pro jiné transakce. To znamená, že dokud transakce běží, nemůže zjevit výsledky svých operací ostatním transakcím, dokud nebude celá transakce potvrzená.
- Trvanlivost (z angl. durability) – Změny úspěšně dokončené (potvrzené) transakce jsou uloženy do databáze a přečkají výpadek nebo restart databázového systému.

Vlastnosti ACID tvoří základní princip transakčního mechanismu. [3]

Constraint

Určuje pravidla pro data v tabulce. Je to omezení, které zajišťuje spolehlivost a přesnost dat v tabulce. Může být aplikováno pouze na sloupec nebo na celou tabulku. Běžně používané jsou:

- `NOT NULL` – Zajistí, že sloupec nemůže obsahovat hodnotu `NULL`.
- `UNIQUE` – Zajistí, že všechny hodnoty ve sloupci budou rozdílné.
- `PRIMARY KEY` – Primární klíč je kombinace `UNIQUE` a `NOT NULL`.
- `FOREIGN KEY` – Cizí klíč tvoří vztah s další tabulkou.
- `CHECK` – Kontroluje, že všechny hodnoty ve sloupci vyhovují specifikované podmínce.
- `DEFAULT` – Nastaví výchozí hodnotu pro sloupec, pokud není žádná specifikována.
- `INDEX` – Umožňuje rychlejší vytváření a vyhledávání v datech.

[5]

Primární klíč

Je sloupec, který jednoznačně identifikuje daný řádek, proto musí být jeho hodnota unikátní, nebude se v budoucnu měnit a musí být vyplněna. Primární klíč může být tvořen i kombinací více sloupců. Tabulka bez primárního klíče může existovat, ale v praxi se takové tabulky nepoužívají. Snižuje to efektivitu zpracování tabulky. Také se bez nich neobejde tvorba vzájemných vztahů mezi tabulkami a jejich existence je nutná pro práci s hodnotami v tabulce. Nové řádky lze přidat, ale úpravy nebo mazání jsou omezené. [2]

Cizí klíč

Mají-li mezi sebou dvě tabulky vztah, znamená to, že jsou umístěné hodnoty z jedné tabulky do druhé. Je tedy použit primární klíč první tabulky, který je uložen do sloupce ve druhé tabulce a tam je označen jako cizí klíč. Cizích klíčů může mít tabulka libovolný počet, který závisí na počtu vztahů. Jméno sloupce v první i druhé tabulce se nemusí shodovat, ale hodnoty musí. [1]

NULL

V databázích má hodnota `NULL` svůj speciální význam. Nelze ji zaměnit za nulu, protože nula je hodnota, která má svůj význam stejně jako mezery nebo jiné bílé znaky. Hodnotu `NULL` lze nejlépe popsat jako „nic“. Jinými slovy lze říci, že hodnotu `NULL` je vhodné použít, když je hodnota neznámá. Hodnotu `NULL` lze později změnit jako každou jinou hodnotu. Žádný klíč nesmí hodnotu `NULL` obsahovat tzn. nesmí zůstat prázdný. Pro ostatní sloupce lze nastavit, jestli je vyplnění atributu povinné nebo lze nechat výchozí hodnotu `NULL`. Lze ji využívat také při dotazování k vyhledávání prázdných nebo vyplněných atributů. [16]

1.2.1 Pravidla pro tvorbu tabulek

Správný návrh databáze je důležitý pro efektivní přístup k aktuálním a přesným datům. Zároveň je databáze poté snáze přizpůsobitelná změnám. Jednoduchá databáze se může skládat pouze z jedné tabulky. Pro většinu databází je ale potřeba více než jedna tabulka.

Mezi základní principy patří, že by databáze neměla obsahovat opakující se informace (také nazýváno redundantní data). Dalším základním předpokladem je, že všechna data jsou přesná a kompletní. Obecně se správný návrh skládá z několika postupně na sebe navazujících částí.

Určení konkrétního účelu databáze

Prvním krokem by mělo být zjištění, jakým způsobem se databáze bude používat a kdo ji bude používat. Tedy kolik lidí ji bude používat, kdy a jak. V případě malého rodinného podniku může jít o jednoho člověka a v případě velké korporátní firmy, může jít i o stovky či tisíce uživatelů v jeden okamžik.

Shromáždění všech požadovaných dat a jejich rozčlenění vhodně do tabulek

Všechny požadované informace, které chceme ukládat v databázi je potřeba sesbírat a vhodně zanalyzovat. Ty poté rozdělit na hlavní entity, které se stanou tabulkami. V tabulce vybrat vhodné sloupce, které mají být uloženy a které popisují pouze daný objekt, který tabulka reprezentuje. Každý sloupec by se měl týkat malé a konkrétní části. Každá informace by měla být uložena pouze jednou. Pokud je potřeba danou informaci použít vícekrát, měla by být uložena v samostatné tabulce. Tímto lze ušetřit místo na disku a vyhnout se problémům při

změně informace, kterou provedeme pouze na jednom místě. Změna bude aplikována pro všechny záznamy, ve kterých se informace vyskytuje a tím je docílena správná konzistence dat. Při návrhu tabulek musí být určen unikátní primární klíč, který bude jednotlivé záznamy jednoznačně identifikovat.

Vytvoření relací mezi tabulkami

Po rozdělení dat do tabulek je nutné některé tabulky spojit opět dohromady pomocí smysluplných vztahů, ale spojení není podmínkou. Tato spojitost se nazývá „kardinalita vztahu“ a existují následující typy.

- 1:M – Záznam z jedné tabulky slouží pro více záznamů v tabulce druhé, kde jsou reprezentovány cizím klíčem. Jako příklad může sloužit tabulka produktů, které mohou mít stejného dodavatele.
- M:N – Záznam z jedné tabulky slouží pro více záznamů v tabulce druhé a zároveň to platí také naopak. Řeší se dvěma vztahy 1:M ke třetí tzv. vazební tabulce, která bude obsahovat primární klíče z obou tabulek. Příkladem je objednávka obsahující více produktů a zároveň jeden produkt může být obsažen ve více objednávkách.
- 1:1 – Záznam jedné tabulky odpovídá právě jednomu záznamu v druhé tabulce. Jedná se spíše o ojedinělý vztah. Využít ho lze při zpřehledňování větších tabulek. Obsahovat může doplňkové informace, které nebudou často využívány.

Relace může být i spojením tabulky sama se sebou. To lze uplatnit například u vedoucího zaměstnance, který pod sebou může mít další zaměstnance.

Zároveň lze určit, zda účast ve vztahu bude volitelná tzn. konkrétní záznam může být použitý ve vztahu, ale není to povinnost.

Aplikování normálních forem

Normalizační formy zajišťují korektní strukturu tabulek. Aplikování těchto pravidel se nazývá také normalizace. Popsány budou první tři normální formy, které tvoří základ pro většinu relačních databázových návrhů.

První normální forma – V každém jednom atributu musí být uložena jediná samostatná hodnota. Jedno pole nesmí obsahovat více hodnot.

Druhá normální forma – Každý sloupec je závislý na celém primárním klíči, nikoliv pouze na jeho části. Platí v případě, kdy se primární klíč skládá z více sloupců.

Třetí normální forma – Sloupec, který není klíčem nesmí být závislý na jiném sloupci, který také není klíčem. Jinými slovy musí být neklíčový sloupec závislý pouze na primárním klíči. Nezávislost sloupce znamená, že jeho změna nesmí mít vliv na ostatní sloupce. [15]

1.3 SQL

Relační databáze pracují s dotazovacím jazykem SQL. Je to neprocedurální jazyk, který popisuje to, co se od databáze požaduje a nejde o to, jak se to má provést. Ten začal být vyvíjen již v roce 1974 pod názvem Sequel. K psaní jeho příkazů si lze vystačit pouze s textovým editorem.

V 80. letech se kromě šíření osobních počítačů začali na implementaci SQL soustředit i světoví výrobci software. Klíčová událost pro SQL byla jeho standardizování organizací ANSI v roce 1986 a její přijetí ISO v roce 1987. Nazýván byl také jako SQL86. Následovaly standardy SQL89 a SQL92. Jeho další vývoj byl pod názvem SQL3 až do roku 1999, kdy se přešlo na značení typu SQL:rrrr, například tedy SQL:1999, které zůstalo do dnes.

Mezi základní rysy SQL patří, že data uložená v databázi jsou skutečné tzn. odpovídají databázovému schématu, anebo virtuální, kdy se jedná pouze o pohledy. Program, kterému vrací SQL data, se nemusí vůbec starat o umístění nebo fyzickou strukturu těchto dat. Umístění tabulek a sloupců není důležité, protože jsou identifikovány jménem. Řádky jsou vždy identifikovány hodnotami ve sloupcích a bez ohledu na vnitřní strukturu jsou data vždy prezentována jako tabulky.

V SQL je možnost pracovat s indexy. To jsou datové struktury, díky kterým lze zpracovávat uživatelské dotazy rychleji, protože dokáží mít rychlejší přístup k řádkům tabulek. Indexy však nejsou standardizovány, protože nepatří do logického datového modelu SQL. Spravuje je správce databázového systému nebo databázový systém samotný. Během existence databáze tedy mohou být definovány, upravovány a zase smazány. [1]

Většina dnešních databázových systémů jako například Microsoft SQL Server, Oracle Database a MySQL mají vytvořené nástroje s grafickým rozhraním, které prací s jazykem SQL usnadňují. Jeho znalost je však pro administrátora stále více než žádoucí, neboť tyto nástroje nepokrývají všechny jeho funkcionality. [2][3]

Základní příkazy SQL jazyka můžeme rozdělit na několik kategorií.

DDL (Data Definition Language)

Určují podobu databázové struktury.

- `CREATE` – Slouží k vytváření databáze, tabulek, indexů, pohledů, funkcí.
- `ALTER` – Mění databázové objekty.
- `DROP` – Maže objekty z databáze nebo samotnou databázi.
- `TRUNCATE` – Smaže všechny záznamy v tabulce.
- `RENAME` – Přejmenuje objekt v databázi.

DML (Data Manipulation Language)

Obstarávají vyhledávání a úpravy dat v databázi.

- `INSERT` – Vkládá data do tabulky.
- `UPDATE` – Upravuje data v tabulce.
- `DELETE` – Maže data z tabulky.

DQL (Data Query Language)

Tato kategorie obsahuje pouze jediný příkaz:

- `SELECT` – Vrací data z databáze.

DCL (Data Control Language)

Starají se o práva a privilegia do databázového systému.

- `GRANT` – Uděluje uživatelům oprávnění přístupu do databáze.
- `REVOKE` – Odebírá uživatelům oprávnění přístupu do databáze.

TCL (Transaction Control Language)

Dovolují uživateli přímé řízení transakcí.

- `COMMIT` – Signalizuje úspěšnost provedení transakce.
- `ROLLBACK` – Signalizuje chybu v transakci a její neúspěšné provedení.
- `SAVEPOINT` – Vytvoří bod určující hranici vrácení změn po chybě v transakci.
- `SET TRANSACTION` – Specifikuje další charakteristiky transakce.

[6]

ROWID

Vyskytuje se v databázovém systému Oracle. Každý řádek v databázi má svoji hodnotu v pseudosloupci `ROWID`. Tento pseudosloupec vrací adresu daného řádku a na úrovni tabulky je unikátní. Obsahuje informace, které jsou nezbytné k určení umístění řádku:

- Číslo identifikující objekt na konkrétním řádku.
- Blok v datovém souboru, ve kterém se řádek nachází.
- Pozice řádku v bloku. První řádek je na pozici 0.
- Datový soubor, ve kterém se řádek nachází. První soubor je značen číslem 1. Toto číslo je relativní k tabulkovému prostoru.

Protože mohou být různé tabulky ve stejném clusteru (běží na dvou nebo více serverech) můžou se tyto adresy z různých tabulek shodovat. Obvykle je však tato adresa v databázi unikátní.

Hodnoty v pseudosloupci `ROWID` jsou reprezentovány jako řetězce `a` mají stejnojmenný vlastní datový typ `ROWID` nebo `UROWID`. `UROWID` je zkratka „universal rowid“ a byl vytvořen pro indexově organizované a cizí tabulky (z jiného databázového systému). Běžné tabulky (organizované na haldě) používají `ROWID`.

`ROWID` hodnoty lze využít jako nejrychlejší způsob přístupu k řádku. Zároveň nám ukazují způsob, jak jsou řádky v tabulce uloženy. Slouží také k jednoznačné identifikaci řádků v tabulce. I přesto, že jsou hodnoty unikátní, neměl by být pseudosloupec `ROWID` zvolen primárním klíčem. Důvodem je, že v případě smazání řádku, může být stejná hodnota `ROWID` přiřazena nově vloženému řádku.

Ačkoliv lze hodnoty `ROWID` číst, nelze je vkládat, modifikovat nebo mazat, protože hodnoty nejsou uloženy přímo v databázi. [25]

1.4 NoSQL databáze

NoSQL databáze jsou zmíněny v této práci, protože jednou z jejich klíčových vlastností je vysoká optimalizace pro vyhledávání. Jedná se o speciální druh nerelačních databází a jejich funkcionalita je často omezena na prosté ukládání dat. Umožňují replikace a také vertikální rozložení dat, to znamená, že dvě části záznamu mohou být uloženy na různých fyzických serverech. Tyto databáze ve většině případů nevyužívají jazyk SQL. Název NoSQL je databázovou komunitou interpretován jako „not only SQL“.

Zejména s rozvojem webových aplikací se začaly rozvíjet také požadavky na databázový systém. Většina velkých firem jejichž hlavním prostředkem k podnikání je internet a webové aplikace začaly mít problém velkých dat (z angl. Big Data problem). Na velký objem dat, který se každým rokem významně zvětšuje přestaly tradiční relační databázové systémy stačit. Problém NoSQL databází se týká dosažení vlastností ACID u transakcí. Praxe ukazuje, že tyto vlastnosti jsou požadovány jen v konkrétních situacích jako jsou například banky a obchody. U velkého objemu dat, kde není potřeba řešit vzájemné vztahy se dostávají do výchozího postavení právě NoSQL databáze.

Při návrhu webových aplikací nejsou důležité ACID vlastnosti, ale vlastnosti CAP, označované také jako garance. První vlastností je konzistence (z angl. consistency), aby dotazovaná data byla vždy aktuální. Další vlastností je dostupnost (z angl. availability), tedy minimální odezva při operacích čtení i zápisu. Ta může být dosažena za pomocí propojení většího množství fyzických serverů, kde bude uloženo více kopií dat. Poslední vlastností je odolnost vůči rozdělení sítě (z angl. partitioning tolerance). To znamená, že z databáze lze číst a zapisovat do ní i v případě, že bude část fyzicky propojených serverů nedostupná. Tyto operace se poté provedou na serverech, které jsou ještě dostupné a zapíší se na nedostupné, jakmile budou k dispozici. Vztahy mezi těmito vlastnosti popisuje CAP teorém, říká se mu také Brewerův teorém podle Erica Brewera, který jej navrhl a prokázal. Tento teorém říká, že neexistuje systém sdílení dat, který dokáže zajistit všechny tři CAP vlastnosti najednou. Webové aplikace obvyklé vybírají mezi konzistencí a dostupností, zatímco korporátní databázové systémy volí maximální odolnost vůči rozdělení sítě. Správná volba tedy závisí na typu aplikace a technických podmínkách.

V širším pojetí patří do NoSQL i XML, dokumentové a objektové databáze. Oproti SQL databázím je v NoSQL databázích datový model spíše intuitivní a nemá žádný formální základ. Patří sem modely:

- **Úložiště typu klíč-hodnota** – Obsahuje množinu dvojic (klíč, hodnota). V přirovnání k relačním databázím představuje klíč atribut nebo název sloupce. Hodnota je typicky řetězec, ale může se jednat i o ukazatel nebo BLOB (datový typ nespecifikovaných binárních dat).
- **Sloupcové NoSQL databáze** – Základem je opět množina dvojic (klíč, hodnota), které lze seskupovat do skupin, do kterých lze přidávat nové atributy (sloupce).

- **Dokumentově orientované NoSQL databáze –** Jedná obvykle o semistrukturované dokumenty formátu JSON vybavené indexy.
- **Grafové databáze –** Velmi specifický koncept. Uzly a hrany grafů jsou uživatelská data strukturovaná jako množiny dvojic (klíč, hodnota). Vyznačují se přímočarou reprezentací dat a dotazování nad grafem je přirozené.

NoSQL databáze zatím svou propracovaností nedosahují úrovně relačních databází, které za sebou mají mnoho let vývoje náskok a představují velké investice a spolehlivost. NoSQL databáze budou spíše využívány ve speciálních projektech s nestrukturovanými daty a vysokými požadavky na škálování. V praxi se většinou NoSQL databáze využívají v kombinaci s relačními databázemi, kdy hlavní roli pro ukládání dat představují relační databáze a NoSQL databáze slouží pro konkrétní další účely, např. pro vyhledávání ve webových aplikacích z důvodu rychlého přístupu k datům apod. [3]

2 ORACLE DATABASE

2.1 Historie

V roce 1977 Larry Ellison, Bob Miner a Ed Oates založili firmu Software Development Laboratory, která byla později přejmenována na Relational Software. V roce 1983 byla firma opět přejmenována, a to na Oracle Systems Corporation a ještě později na Oracle Corporation. Oracle byl původně název projektu, který měla společnost vytvořit pro CIA. Cílem projektu bylo shromažďovat velké množství dat a zároveň umožnit rychlé vyhledávání v těchto datech.

První verze v roce 1978 byla napsána v assembleru pro počítač PDP-11.

V roce 1979 byl představen Oracle V2 (verze 2) první komerčně dostupný RDBMS. Pro relační databáze to byla historicky významná událost. Problémem těchto verzí byla přenositelnost na jiné hardwarový platformy, a tak bylo přijato zásadní strategické rozhodnutí, že další verze budou z toho důvodu psány v jazyce C.

Verze 3 vyšla v roce 1983 a byla již napsána v jazyce C. Jednalo se o první databázovou platformu, která byla přenositelná na úrovni zdrojového kódu a byla použita na sálových počítačích i minipočítačích.

S verzí 6 v roce 1988 přišla možnost vytvoření velkých transakčních systémů. Počítače se v případě výpadku dokáží nahradit a sdílí spolu diskový prostor. Verze 8i v roce 1997 přinesla integraci jazyka Java pro aplikace klient/server. Tato verze byla podporována také systémem Linux a písmenko „i“ značí orientaci této databázové platformy na internetové prostředí. V roce 2000 je tu verze 9i, se kterou se objevila snadná možnost rozšiřování systému tzv. škálovatelnost. Tato verze přinesla hlavně technologie, které významně zvýšily výkon databázového systému. Verze 10g v roce 2003 přinesla zpracování ve gridu, podle toho je také písmenko „g“ v označení verze. Myšlenka byla mít všechen výpočetní výkon na jednom místě a dynamicky ho přidělovat podle priorit organizace. [7][17]

2.1.1 Verze 11g

Další verze 11g vydaná v roce 2008 přišla s významným vylepšením poměru komprese dat, možností zobrazení tabulek zpátky v čase a vylepšené funkce na obnovu databáze v případě výpadku. V dalším odstavci jsou popsány změny týkající se optimalizace pro vyhledávání.

Doménové indexy v této verzi aktualizují svoje metadata během úprav tabulky pomocí příkazu `ALTER TABLE` a metadata jsou tak vždy aktuální. Zároveň byl vylepšen systém pro správu doménových indexů, který zvyšuje jejich výkon a možnosti spravování. Vylepšeny byly také

XML indexy, kde jsou indexové operace odděleny od operací `INSERT` a `UPDATE`. To znamená, že aplikace nemusí čekat na dokončení indexování před dokončením transakce. Byla přidána možnost označit index jako „neviditelný“. Ten je stále dostupný pro DML operace, ale optimalizátor ho ignoruje, pokud není použitý jako nápověda (z angl. hint). Neviditelný index může být použitý pro dočasné operace nebo když daný index není potřeba, ale není nutné ho hned mazat. Vytvoření nebo obnovení indexu vyžadovalo krátké zablokování DML operací. Ve verzi 11g již toto blokování není potřeba. Byla vylepšena funkce `ANALYZE VALIDATE CASCADE`, která hledala poškozené indexy, ta nyní využívání porovnání hashů a je výrazně rychlejší. Dále bylo přidáno nové rozhraní pro správu „Textových“ indexů. Poslední novinkou z hlediska indexů bylo přidání podpory „bitmap join indexů“ pro indexově organizované tabulky, které zvyšují jejich výkon. [40]

2.1.2 Verze 12c

Verze 12c vyšla v roce 2013 a přinesla velké změny. Písmeno „c“ v názvu, znamená cloud. Umožňuje všem databázím snadný přechod do cloudového prostředí a slibuje rychlejší, škálovatelnější a spolehlivější technologii. Představuje multitenantní architekturu, která dovoluje spravovat více databází jako jeden celek. Dále je představena funkce automatického ukládání velkých tabulek do mezipaměti (z angl. big table caching), která významně zvyšuje výkon procházení celých tabulek (z angl. full table scan). Dále pokud je volná paměť větší, než velikost databáze je možné uložit do paměti celou databázi, a to přináší velké výkonnostní benefity. Nyní lze nastavit sloupce jako „neviditelné“ a nebudou zobrazovány ve výsledcích dotazů, pokud nebudou konkrétně definovány v dotazu. Další novou funkcí je automatická optimalizace a komprese dat podle jejich stáří a četnosti využití.

Z hlediska indexů byl vylepšen kompresní poměr všech podporovaných indexů bez omezení rychlosti přístupu k indexu. Velkou změnou je možnost vytvoření více indexů na množinu stejných sloupců. Předpokladem proto je odlišná charakteristika indexů například unikátní/neunikátní nebo založené na B-stromu/bitmapové. Zároveň je tím také zajištěna migrace bez nutnosti smazat a znovu vytvořit index. Globální údržba indexů je oddělena a zpožděna od operací `DROP` a `TRUNCATE`. Díky tomu jsou tyto operace provedeny rychleji a méně zatěžují systém. Dále je k dispozici nová funkce „Real-Time Indexing“. Je vytvořena pracovní tabulka pro indexy, která je relativně malá a je uložena ve vyrovnávací paměti. Ta zvyšuje výkon při časté synchronizaci hlavního indexu bez zpomalení dotazů. [18]

2.1.3 Verze 18c

Po verzi 12c přišlo s verzí 18c číslování verzí podle roku. Oracle tím chce přejít na častější a pravidelnější vydávání. Verze 18c je spíše rozšiřující aktualizace verze 12c, která nepřináší mnoho velkých novinek a dlouhodobou podporu. Tato verze přináší přímou integraci Active Directory od společnosti Microsoft. Platforma Oracle Application Express dostala množství nových funkcionalit s verzí 5.1. Dále vylepšení týkající se podpory JSON a PL/SQL.

Vylepšen byl také JSON vyhledávací index. Ten v předchozích verzích podporoval velikost klíče o maximální velikost 64 bajtů a nyní podporuje až 255 bajtů. Proces na pozadí nyní může přesunout čekající aktualizovaná data indexu přímo do hlavního indexu automaticky. Nevzniká tak fragmentace hlavního indexu. Mnoho aplikací, tak nyní nemusí pouštět optimalizaci indexu ručně. U textových indexů lze nyní nově zvolit limit počtu tokenů, které budou aktualizovány a typy sekcí k indexaci. [41]

2.2 Oracle Database Release 19c

Verze 19c je aktuální stabilní RDBMS od Oracle Corporation z roku 2019 a je to nástupce verze 18c. Jedná se o hlavní verzi s dlouhodobou podporou do roku 2024 s možností placené prodloužené podpory do roku 2027. Kvůli dlouhodobé podpoře je doporučeno aktualizovat z verzí 11,12 a 18, kterým nedávno skončila nebo skončí podpora. Doporučeno je samozřejmě aktualizovat i starší verze, ale tam již není podpora přímé aktualizace, ale je nezbytné aktualizovat nejdříve na verze zmíněné výše. [9]

2.2.1 Novinky verze 19c

Flashback na primární databázi bude automaticky proveden i na záložní

Flashback je obnova celé databáze v čase na starší bod obnovy. V předchozích verzích, pokud byla primární databáze obnovena v čase, tak záložní setrvala v původním stavu. Bylo tedy nutné manuální procedurou obnovit záložní databázi na totožný bod obnovy jako primární. Nyní je k dispozici nový parametr, který zapne provádění tohoto procesu automaticky.

Záplatování systému bez výpadku

Záplatování ve verzi 19c probíhá odděleně a Oracle zaručuje, že během záplatování nebudou přerušeny žádné databázové operace a nebude to mít vliv ani na jejich výkon. Novou funkci podporuje aplikace *Oracle Grid Infrastructure*, kde databáze běží pouze na jednom serveru a také aplikace *Oracle Real Application Clusters* (RAC). Ta slouží pro dva a více serverů, které běží v tzv. clusteru. V obou případech, by operace neměly být přerušeny ani rychlostně omezeny na žádném ze serverů.

SQL Karanténa

SQL dotazy, které jsou ukončeny aplikací *Oracle Database Resource Manager*, protože nadměrně zatěžují procesor a vstupní/výstupní prostředky můžou být nyní automaticky uloženy do karantény. Zde budou uloženy jako prevence před znovu provedením. Tato funkce chrání databázi před poklesem výkonu.

Automatické indexování

Tato funkce využívá algoritmy strojového učení (z angl. machine learning) pro vytváření a stálé vylepšování indexů pro výkon a šetření nákladů. To znamená, že při vytvoření databáze lze ponechat vytváření indexů kompletně na systému. Systém během krátké doby zjistí, jaká data a jakým způsobem jsou dotazována a pro jejich efektivní vykonávání vytvoří indexy.

Vylepšená podpora formátu JSON

Podpora formátu JSON je zlepšena. Oracle vylepšil funkce a zjednodušil jejich syntaxi. Byla přidána funkce umožňující provést více změn do více JSON dokumentů jediným příkazem. Nyní lze také převádět SQL objekty a kolekce do JSON formátu a naopak.

Hybridní rozdělení tabulek

Tato novinka umožňuje spravovat tabulku mezi vnitřním úložištěm databáze a úložištěm mimo databázi. To významně rozšiřuje funkcionality pro rozdělení velkých dat, která mohou být rozdělena a uložena na externí úložiště. Umožňuje tak sjednotit tabulku uloženou na interním a externím oddíle do jediné.

Další novinky jsou podrobně popsány ve zdroji. [8]

Verze 21c

V současné době je dostupná již verze 21c pro produkční využití, která je označena jako „vydání inovací“. V této verzi lze již teď používat novinky jako například přístup ke strojovému učení pomocí technologie AutoML a desetkrát rychlejší skenování díky podpoře formátu BSON. Tyto a další novinky budou začleněny do dalších stabilních verzí s dlouhodobou podporou. [19]

2.3 Architektura

Architektura se skládá ze tří hlavních struktur. Paměťové, procesní a úložné. Databázový systém Oracle se skládá z databáze a aspoň jedné instance. Instance databáze je kombinace operační paměti a procesů, které jsou součástí běžící instalace. Databáze se skládá ze souborů, ve kterých jsou uložena data.

Instance může běžet sama o sobě bez přístupu k souborům databáze, a naopak databáze může existovat bez instance, ale nelze se bez ní dostat k datům. Jedna instance může přistupovat pouze k jedné databázi. Může však více instancí přistupovat k jedné centrální databázi, tento stav se nazývá také cluster. Ten zajišťuje vysokou dostupnost a škálovatelnost. [30]

2.3.1 Fyzické úložné struktury

Jsou soubory, které ukládají data. Jedná se o soubory, které se vytvoří po zadání příkazu `CREATE DATABASE`. Jedná se o tyto typy souborů.

- Datové soubory – Obsahují reálná data, která jsou uložená v tabulkách. Zároveň jsou zde uloženy také informace o logické struktuře tabulek a indexech.
- Kontrolní soubory – Soubory obsahující metadata fyzické struktury databáze. Například jméno databáze a umístění datových souborů.
- Online redo log soubory – Jedná se o soubory, které zaznamenávají veškeré změny v datových a kontrolních souborech. Lze je využít k vrácení stavu databáze po pádu. *Redo* v názvu znamená v překladu „znovu udělat“ a týká se to změn provedených před pádem. Tyto soubory se také archivují a fungují jako záloha.

Dalšími důležitými soubory, které patří do této kategorie jsou soubory parametrů, síťové a zálohovací soubory.

2.3.2 Logické úložné struktury

Tyto struktury používá Oracle k efektivnímu využití diskového prostoru. Jedná se o následující rozdělení.

- Datové bloky (z angl. data blocks) – Datový blok odpovídá specifickému počtu bajtů na disku. Jsou v nich ukládána data. Jedná se o nejmenší úložnou jednotku databáze Oracle. Datové bloky jsou také nazývány logické bloky, Oracle bloky nebo stránky.
- Rozsahy (z angl. extents) – Jedná se o specifický počet logicky za sebou uložených datových bloků. Ukládají jeden konkrétní typ informace.
- Segmenty (z angl. segments) – Skládá se z jednoho nebo více rozsahů a je alokovaný pro ukládání objektů. Například tabulky, indexu nebo i dočasných dat.
- Tabulkové prostory (z angl. table spaces) – Databáze je rozdělená na logické úložné jednotky označované jako tabulkové prostory. Tabulkový prostor je logický kontejner pro jeden nebo více segmentů. Každý tabulkový prostor je tvořen minimálně jedním

datovým souborem. Tím je vytvořen vztah mezi logickými a fyzickými úložnými strukturami.

2.3.3 Instance databáze

Databázová instance tvoří rozhraní mezi klientskými aplikacemi a databází. Tato instance se skládá ze tří hlavních částí. První je systémová globální oblast (z angl. system global area), dále jen SGA. Další částí je programová globální oblast (z angl. program global area), dále jen PGA. Poslední části jsou procesy běžící na pozadí.

SGA je struktura sdílené paměti, která je alokovaná při startu instance a opět uvolněná při ukončení. Obsahuje data a kontrolní informace pro danou databázovou instanci, která jsou dostupná všem procesům. Naproti tomu PGA je privátní paměťová oblast, která je alokovaná po startu konkrétního sezení (z angl. session) klientského procesu a uvolněna po jejím skončení. [31]

Hlavní procesy běžící na pozadí

- PMON – Monitor procesů, který řídí ostatní procesy. Tento proces musí vždy běžet. V intervalech skenuje procesy a sleduje, jestli neočekávaně nespadly.
- SMON – Systémový monitor procesů provádí údržbové operace. Mezi jeho hlavní činnosti patří obnova instance po pádu a odstraňování dočasných souborů.
- DBWn – Proces, který zapisuje do databáze. Každá zápisová operace je nejprve zapsána do paměti, protože je to mnohonásobně rychlejší a efektivnější než na disk. Tento proces tedy čte z disku a zapisuje na něj zpátky. Těchto procesů běží více a jsou označovány číslem na místo písmena „n“ například DBW0.
- CKPT – Každé 3 sekundy kontroluje, jestli nebyla přeplněna paměť nad stanovený limit. Aktualizuje hlavičky datových souborů a kontrolní soubory. Dává signály DBWn procesu.
- LGWR – Zapisuje do redo log souborů. Je tedy klíčový pro obnovu databáze. Každou změnu v databázi zapíše LGWR nejprve do paměti a poté do redo log souborů na disku.
- ARCn – Archivační proces, který ukládá obsah redo log souborů do archivu těchto souborů. Procesů může běžet více a jsou označovány číslem na místo písmena „n“ například ARC0. Více procesů lze využít při zapisování velkého množství dat nebo při archivaci na vícero umístění. Pro každé umístění by měl být jeden proces.

- MMON – Shromažďuje informace o běhu procesů. Tyto informace lze použít k detekci chyb a pro účely optimalizace. Sbírá statistická data o systému, sezení, službách a SQL voláních.

[30]

Mezi hlavní vlastnosti databáze Oracle patří:

- Jednoduchá obnova dat v porovnání s ostatními databázovými systémy.
- Systém dokáže řídit a manipulovat s velkým množstvím dat.
- Dovoluje kdykoliv změnit platformu.
- Dává možnosti rozšíření a škálovací strategie.
- Dokáže nasimulovat aktuální zátěž včetně přihlášených uživatelů v testovacím prostředí.
- Široká podpora hardwaru a virtualizačních technologií.
- Nepřerušitelné zpracování požadavků uživatelů, které eliminuje potřebu manuálního obnovení dat.
- Umožňuje vytvořit repliku primární databáze jako pohotovostní databázi, kterou lze využít jako zálohu, pro reporty, testování nebo pro snížení zátěže primární databáze.
- Má komplexnější ale více efektivní syntaxi.
- Lze stáhnout verzi s volnou licencí.
- Používá speciální bitmapové indexování založené na funkcích a reverzních klíčích.
- V transakčním procesu lze využít `ROLLBACK.`
- Změny hodnot v transakci nejsou provedeny, dokud není zavolán příkaz `COMMIT`.
- Každé připojení do databáze se zpracovává jako nová transakce.
- Všechny databázové objekty jsou seskupeny podle schématu. Databázové objekty a jejich podmnožiny jsou sdíleny mezi všemi schématy a uživateli.

[12]

2.3.4 Datové typy

Mezi často vyskytované datové typy patří:

- VARCHAR2 – Ukládá textové hodnoty. VARCHAR2 je pro text psaný latinkou a NVARCHAR2 pro Unicode, lze tak použít větší škálu znaků. Jediný parametr je počet znaků, který může být maximálně 4 000. Ukládá počet znaků skutečně vložených, nikoliv kolik je uvedeno parametru.
- LONG – Určen pro ukládání velkých textových hodnot. Lze uložit text o velikosti až 2 GB.
- NUMBER – Jedná se o číselný datový typ, který je definován celkovým počtem číslic a počtem číslic vyskytujících se za desetinnou čárkou. Datový typ povoluje celkový maximální počet číslic 38.
- DATE – Slouží pro ukládání data a času. Ukládat lze hodnoty od 1. 1. 4712 př. n. l. do 31. 12. 9999. Datum ukládá databáze jako 7bytové číslo.
- BINARY_FLOAT a BINARY_DOUBLE – Poskytuje základní funkcionality jako NUMBER, ale je vždy přesný na dvě desetinná místa. Proto rychleji zpracovává aritmetické operace a zabírá méně místa. BINARY_FLOAT je 32bitový a BINARY_DOUBLE je jeho 64bitová varianta.

[14]

3 MICROSOFT SQL SERVER

3.1 Historie

V roce 1988 vyšla první verze SQL Serveru. Byla vytvořena společností Microsoft a Sybase. Cílem bylo vytvořit databázový systém, který bude konkurovat DB2 od IBM a Oracle Corporation. Microsoft a Sybase se dohodly, že Sybase bude mít všechna prodejní práva a profit z databázového systému, který nebude pro zařízení s operačním systém od Microsoftu a naopak. První verze byla vyvinuta pouze pro platformu OS/2, další verze už byly vyvíjeny také pro Windows NT. Microsoft se rozhodl, že vývoj SQL Serveru bude úzce spojený s vývojem NT systémů. Po boku Windows NT 3.1 tak vyšel i SQL Server 4.2.

Filozofie společnosti Microsoft byla kombinace výkonné databáze a rozhraní jednoduchého na používání. Tato filozofie se stala úspěšnou a Microsoft se rychle stal druhým nejpopulárnějším prodejcem relačních databázových systému a stal se hlavním konkurentem pro Oracle Corporation. [10][12]

V roce 1994 koupil Microsoft všechna práva od Sybase a vydal v dalším roce SQL Server 6.0. Toto vydání výrazně zasáhlo do jádra technologie SQL serveru a zlepšilo výkon databáze. Začalo poskytovat integrované replikace a přineslo centralizovanou administraci. Zároveň tato verze podporovala webové aplikace. Verze 6.5 vyšla pro Windows NT. Verze 7.0 byla nazývána webovou databází. Poté vyšla verze SQL Server 2000. S tou přišla podpora Bussiness Intelligence. Ta byla významně vylepšena i ve verzi SQL Server 2005, která zároveň přinesla XML jako výchozí datový typ. Verze SQL Server 2008 dál vylepšovala stávající funkce a přinesla i funkce nové. Mimo jiné rozšířenou správu různých geografických údajů například z GPS zařízení. [42]

Verze SQL Server 2012 nabídla například verzi serveru bez grafického rozhraní a sekvenční objekty. Také přišla se sloupcově ukládanými indexy (z angl. columnstore index). Tento typ indexu dominuje u datových skladů a analytických dat. Oproti řádkově ukládanému indexu nabízí násobně lepší výkon a kompresi dat. [45]

Ve verzi SQL Server 2014 byla představena funkce „In-Memory OLTP“, která výrazně zvyšuje výkon ukládáním tabulek do vyrovnávací paměti. Byla přidána možnost ukládat data cloudového prostředí Microsoft Azure a zároveň na toto úložiště provádět i zálohy databáze. Také byla přidána možnost ukládání dat na SSD disky. Byly dále vylepšeny také sloupcově ukládané indexy. [46]

3.1.1 SQL Server 2016

Nová funkce „Query Store“ ukládá dotazy, exekuční plány a výkonnostní měření. Díky tomu lze jednoduše analyzovat problémy týkající se výkonu. Je monitorován čas trvání dotazů a kolik zabral prostředků paměti a procesoru. Nová funkce „Always Encrypted“, která má svůj šifrovací klíč k citlivým datům. Klíč není nikdy předán SQL Serveru. Do cache paměti lze nyní nahrát tabulky o velikosti až 2 TB doteď bylo maximum 256 GB. Zároveň byly vylepšeny sloupcově uložené indexy pro rychlejší třídění v cache paměti. Byly přidány vlastní indexy pro zlepšení výkonu dotazů. [44]

3.1.2 SQL Server 2017

Verze SQL Server 2017 znamená pro Microsoft velký krok vpřed, protože tuto verzi lze nainstalovat i na Linux a kontejnery Docker. Funkce automatického ladění databáze umožňuje najít potencionální výkonnostní problémy a navrhnout jejich řešení. Nová generace funkce provádění dotazů adaptuje optimalizační strategii pro potřeby aplikace dle zátěže aplikace. Transakce mezi databázemi jsou nyní podporovány všemi databázemi, které jsou součástí skupiny „Always On Availability Group“. „SQL Server R“ služby byly přejmenovány na „SQL Server Machine Learning“ služby.

Z hlediska indexů přichází podpora pokračování operace „online obnovy indexu“. Nyní lze pokračovat v místě zastavení této operace, ať jde o výpadek systému nebo plánované pozastavení. [43]

3.2 SQL Server 2019

SQL Server 2019 je postaven na předchozích verzích a snaží se rozvíjet platformu SQL Server jako jediný produkt. SQL server se snaží být postaven, tak aby si zákazník mohl vybrat vývojový jazyk, operační systém a umístění u sebe nebo v cloudovém řešení.

3.2.1 Novinky verze SQL Server 2019

Vylepšení inteligentního provádění dotazů

Jedná se o sadu vylepšení, která ovlivňuje chování optimalizátoru dotazů. Tato komponenta generuje exekuční plán pro dotazy a také se stará například o dynamické přidělování paměti. Optimalizátor je zapnutý ve výchozím nastavení a celkový výkon prováděných dotazů by se měl zlepšit bez složité implementace.

Accelerated Database Recovery (ADR)

Tato nová funkce je nový způsob provádění obnovy databáze. Například v případě pádu nebo restartu databázového systému během zpracovávání datových operací nebo vrácení transakce zpět klientem. Po aktivaci této funkce, by se měla výrazně snížit doba této obnovy.

Vylepšení funkce Always Encrypted

První verze této technologie byla představena již ve verzi SQL Server 2016. Poskytuje transparentní zašifrování sloupců, aniž by zpřístupnila administrátorům dešifrovací klíče. Nevýhodou první verze bylo, že SQL Server nebyl schopný použít zašifrovaná data pro výpočty a ani s nimi manipulovat. Nyní dokáže SQL Server zašifrovat část paměti, kde probíhají výpočty s daty a zároveň nejsou tyto dešifrované hodnoty přístupné administrátorům ani jiným procesům.

Query store nyní umožňuje nastavit vlastní filtry

Tento nástroj ukládá, monitoruje a umožňuje optimalizovat dotazy. Jeho nevýhodou je, že může ukládat informací až příliš, a to včetně dotazů, které nemusí být pro sledování zajímavé. Například ty, které jsou součástí jiných nástrojů. A to může znamenat zátěž systémových prostředků navíc. Nyní byla přidána funkce „vlastních politik“, které umožní filtrovat sledované dotazy. Filtrovat lze například na základě statistik, jak dlouho běží, jak vytěžují procesor a jak často jsou prováděné. Sledování je nyní efektivnější a přehlednější.

Mezi další novinky patří optimalizace metadat systémové databáze *tempdb.* Metadata, která mohla brzdit velké databázové systémy, nyní spoléhají kompletně na operační paměť a jsou optimalizována pro přístup z ní. Vytváření indexů lze pozastavit a opět obnovit v jiném čase. Zároveň byly zoptimalizovány indexy, které obsahující sekvenční klíč. [26]

3.3 Architektura

Architektura databázového systému Microsoft SQL Server je založená na architektuře klient-server. Na úrovni protokolové vrstvy používá SQL Server čtyři protokoly. Prvním je sdílená paměť, kdy je klient i server na stejném fyzickém serveru. Druhým protokolem jsou pojmenované roury, které se používají pro komunikaci v lokální síti. Dalším protokolem je TCP/IP využívaný pro vzdálené připojení z vnější sítě. Poslední protokol představený ve verzi Microsoft SQL Server 2012 je protokol VIA (Virtual Interface Adapter). Tento protokol vyžaduje implementaci přímo od výrobce specializovaného hardwaru vyrobeného přímo pro tento účel. Tento protokol dokáže vytvořit velmi výkonné spojení.

Významné komponenty této architektury jsou následující.

- Parser dotazů a kompilátor (z angl. query parser and compiler) – Kontroluje syntaxi dotazu a překládá dotaz do strojového kódu.
- Optimalizátor dotazů (z angl. Query Optimizer) – Výstupem optimalizátoru je exekuční plán vytvořený na základě dotazu, statistik a speciálního stromu. Tento strom je speciálním formátem, který slouží jako vstup právě pro optimalizátor dotazů. Tento speciální formát vznikne po kontrole původního příkazu. Kontroluje se, zda je validní, existují sloupce a také se identifikují datové typy.
- Exekuční plán (z angl. execution plan) – Slouží jako plán, kde je krok za krokem popsáno, která část dotazu se má vykonat.
- Vykonávač dotazů (z angl. query executor) – Zde je dotaz za pomoci exekučního plánu proveden a je kontaktován úložný engine.
- Úložný engine (z angl. storage engine) – Jeho úkolem je manipulace s daty včetně čtení, zamykání a spravování transakcí.
- SQL OS – Obstarává všechny aktivity mezi operačním systémem a databázovým systémem. Řeší například správu paměti nebo stav uváznutí (z angl. deadlock).
- Proces kontrolního bodu (z angl. checkpoint process) – Proces, který zapisuje modifikované stránky z vyrovnávací paměti na disk. To samé dělá se záznamy logů. Tento proces běží automaticky ve specifikovaném intervalu a je díky němu zkrácena doba obnovy po pádu systému.
- Proces Lazy Writer – Pokud začne SQL Server potřebovat víc paměti tento proces začne uvolňovat paměť tím, že bude okamžitě modifikované stránky z paměti zapisovat na disk. Díky tomuto procesu by tak mělo vždy být dostupné určité množství paměti. Tento proces běží interně a nelze ho ovlivnit nastavením. Pokud je tento proces stále aktivní, může to indikovat problémy s nedostatkem paměti.

Typy kontrolních bodů

- Automatické – Nejběžnější kontrolní bod běžící jako proces na pozadí. Zajišťuje, že databáze může být obnovena ve zvoleném časovém limitu nazývaném jako interval obnovy.
- Indirektní – Byly představeny s verzí SQL Server 2012. Běží také na pozadí. Lze si vybrat na kterou databázi bude aplikován a na té přepíše interval obnovy definovaný automatickým kontrolním bodem.

- Manuální – Spustí se ručně jako T-SQL příkaz. Lze nastavit maximální délku čekání na dokončení kontrolního bodu.
- Interní – Tyto kontrolní body nemůže uživatel ovládat a jsou spuštěny automaticky při specifických událostech. Například při plánovaném vypnutí, během zálohy databáze nebo při příkazu, který manipuluje s datovými nebo log soubory.

3.3.1 Architektura paměti

Jednou z charakteristik databázového systému je minimalizovat čtení/zápis na disk, protože se jedná o operace, které nejvíc zatěžují hardwarové prostředky. *Buffer management* je klíčová komponenta, která zajišťuje efektivitu operací čtení a zápisu. Tato komponenta se skládá ze dvou mechanismů. *Buffer manager* řídí přístup a aktualizuje stránky. *Buffer pool* reguluje operace zápisu a čtení. *Buffer manager* se skládá z dvou dalších významných částí *buffer cache* a *procedure cache*. *Buffer cache* drží datové stránky v paměti, aby mohly být často přistupované stránky ihned k dispozici. Alternativou by bylo neefektivní čtení z disku, které trpí pomalejším přístupem, a právě zvýšenou frekvencí operací čtení/zápis. *Procedure cache* drží uložené procedury a exekuční plány, které tak nemusí být pokaždé znovu generovány. Další části *buffer poolu* se starají alokaci místa pro zásobníky, které jsou vytvářeny spolu s každým dalším vláknem SQL Serveru. Drží metadata o instanci databáze a o stavu připojení. A také rezervují vyrovnávací paměť pro transakční log stránky.

3.3.2 Architektura databázových souborů

Soubory databáze mohou být pro administrační účely přiřazeny do skupin. Soubor může být součástí pouze jediné skupiny. To neplatí pro log soubory, který jsou spravovány samostatně a nemůžou být součástí žádné skupiny. Existují dva druhy skupin. Primární skupiny a definované uživatelem. Všechny stránky systémových tabulek a hlavní datové soubory jsou přiřazeny do primární skupiny, pokud to uživatel nespecifikuje jinak. Do uživatelem definované skupiny lze soubory přidat pomocí klíčového slova při vytváření nebo modifikaci databáze. Každá databáze má svoji výchozí primární skupinu, pokud to uživatel nedefinuje jinak.

Databáze používá tři typy souborů. Primární a sekundární datové soubory a log soubory. Každá databáze má jeden primární datový soubor. Všechny ostatní soubory jsou sekundární, ale nemusí být v databázi žádný sekundární soubor. Databáze musí mít aspoň jeden log soubor. Ten obsahuje informace k obnově databáze v případě pádu. Umístění všech souborů je umístěno v hlavní databázi a také v primárním databázovém souboru. Soubory mají logické a fyzické pojmenování. Logické pojmenování používají příkazy T-SQL a fyzické

využívá operační systém. Soubory můžou být uložené pouze na souborovém systému FAT nebo NTFS.

Stránky (z angl. pages) – Jsou základní úložnou jednotkou databázového systému SQL Server. Velikost jedné stránky je 8 KB. Na začátku každé stránky je hlavička, která ukládá informace o typu stránky, volném místě a také je zde umístěn ID objektu, který stránku vlastní. Je celkově devět typů stránky a jsou popsány ve zdroji [32].

Rozsahy (z angl. extents) – Jsou další úložnou jednotkou a jejich místo je alokováno tabulkám a indexům. Skládají se z osmi za sebou uložených stránek. Jsou dvojího typu. Jednotné patří jednomu objektu. Naproti tomu smíšené mohou být sdílené až osmi objekty.

3.3.3 Architektura log souborů

Každý log záznam je identifikovaný číslem, které se nazývá „Log Sequence Number“ (LSN). Každý záznam obsahuje ID transakce, ke které patří.

Log záznamy pro modifikaci dat jsou dvojího typu. První možností je záznam operace, která byla provedena. Druhá varianta je, že se vytvoří záznam, kde je kopie dat před změnou a po změně. Při této variantě se při obnově obnoví kopie dat před nebo po změně. U první možnosti se buď operace provedu znovu nebo reverzně. Transakční log zaznamenává následující operace.

- Začátek a konec každé transakce.
- Vložení, úprava nebo smazání dat. Včetně změn provedených systémem.
- Každá alokace a dealokace rozsahu nebo stránky.
- Vytvoření nebo smazání tabulky nebo indexu.

Zaznamenávány jsou i `ROLLBACK` operace. Každá transakce si rezervuje speciální místo pro `ROLLBACK`, který by mohl nastat v případě chyby, když probíhá `ROLLBACK`. Po úspěšném provedení transakce je toto místo opět uvolněno.

SQL Server databázový engine interně rozděluje fyzické log soubory na virtuální log soubory. Počet těchto virtuálních souborů není specifikován a ani jejich velikost. Velikost je volena dynamicky při vytváření těchto virtuálních souborů. Velikost a počet nemůže nastavit ani administrátor. V případě malého fyzického log souboru může vzniknout velký počet malých virtuálních log souborů, které mohou ovlivnit výkon databáze. Zpomalit start databáze

a operace zálohování a obnovy. Tomu lze předcházet větší velikostí fyzického log souboru při inicializaci a nastavit relativně větší hodnotu parametru *growth_increment*. [32]

SQL Server byl dlouho vyvíjen pouze pro platformu Windows. Od verze SQL Server 2017 podporuje i Linux a kontejnery Docker. Oproti řešení od Oracle Corporation má SQL Server jednodušší syntaxi, ale ne vždy s ní lze dosáhnout vysoké efektivity. Další vlastnosti SQL serveru jsou:

- Nabízí plno rozšiřujících aplikací jako SQL Server Profiler, BI tools, SQL Server Management Studio a Database Tuning Advisor.
- Poskytuje rozšířené možnosti pro smazání, přejmenování objektů a mapování datových typů.
- Monitor aktivit s filtrováním a automatickým obnovováním.
- Integrované samostatné prostředí pro správu přidělování práv a SQL Server Database Enginu.
- Plánování úloh probíhá přes SQL Server Agenta.
- Změna hodnot v transakčním procesu proběhne ještě před příkazem `COMMIT`.
- Při změně záznamu blokuje záznam. Nelze ho tedy v tu samou chvíli číst.
- Používá globální alokaci paměti, která snižuje šance na chyby způsobené člověkem.
- Každá databáze na serveru má svůj vlastní nesdílený soubor.
- Nabízí online podporu, dokumentaci a přímou podporu k zakoupenému produktu.

[12]

T-SQL

Transact-SQL je proprietární rozšíření jazyka SQL od firmy Microsoft a Sybase. Přidává do jazyka SQL funkce jako například podporu kontroly transakcí, výjimek a zachytávání chyb. Obsahuje také procedurální programování a lokální proměnné. [13]

3.3.4 Datové typy

Často používané datové typy jsou:

- INT – Integer Data Type ukládá celá čísla. Jedná se o 32bitové číslo, které může být kladné i záporné. Často je využíván jako ID, primární klíč.

- DECIMAL – Jedná se o číselný datový typ, který je definován celkovým počtem číslic a počtem číslic vyskytujících se za desetinnou čárkou. Datový typ povoluje celkový maximální počet číslic 38.
- VARCHAR a NVARCHAR – Ukládá textové hodnoty. VARCHAR je pro text psaný latinkou a NVARCHAR pro Unicode, lze tak použít větší škálu znaků. Jediný parametr je počet znaků, který může být maximálně 8 000 znaků respektive 4 000 pro NVARCHAR.
- DATETIME – Slouží pro ukládání data a času. Čas ukládá až na přesnost tisícin. V případě potřeby lze zapsat jen datum a čas se nastaví automaticky na vynulovaný. Ukládat lze hodnoty od 1. 1. 1753 do 31. 12. 9999.
- FLOAT – Je to 64bitový číselný datový typ. Nepřebírá žádný parametr. Podporuje desetinná čísla. Vhodný, když nestačí DECIMAL.
- BIT – Ukládá pouze hodnoty 1, 0 nebo `NULL`. Jinými slovy jde o hodnoty pravda a nepravda. [2][11]
- TEXT a NTEXT – Slouží pro ukládání dlouhých textových hodnot o velikosti až 2 GB. TEXT je pro text psaný latinkou a NTEXT pro Unicode. [21]

4 MYSQL

4.1 Historie

Historie MySQL začala v roce 1979, kdy Michael Widenius, přezdívaný „Monty“ vytvořil pro malou firmu TcX, které byl také spoluvlastníkem první reportovací nástroj. Ten napsal v programovacím jazyce BASIC, aby ho později přepsal do jazyka C. Tento nástroj byl nazýván Unireg.

Díky prostředí, které Montyho limitovalo nízkým výpočetním výkonem, si Monty osvojil schopnost psát velmi efektivní kód. Také psal kód stylem, který umožňoval jeho snadné rozšiřování v budoucnu.

Začátkem devadesátých let začali klienti firmy tlačit, aby jejich data byla uložena v SQL systému. Jednou z možností bylo nahrát data do komerční databáze. Monty však nebyl spokojený s rychlostí komerčních produktů a jako programátor se rozhodl, že si napíše vlastní řešení.

A tak v květnu 1996 vyšla pro omezenou skupinu klientů verze MySQL 1.0, aby poté v říjnu téhož roku vyšla verze 3.11.1. To bylo první veřejné vydání a bylo dostupné pro Solaris. O měsíc později vyšla verze i pro Linux.

V následujících letech byla MySQL rozšířena na další operační systémy a zároveň byla stále vylepšována. MySQL vycházela pod speciální licencí umožňující její volné komerční využití pod podmínkou, že nebude součástí softwaru, který organizace dále prodává. Byla dostupná i placená licence pro zákazníky, kteří ji chtěli svázat se svým produktem a prodávána byla také komerční podpora navíc.

Další verze 3.22 byla velmi rychlá a stabilní, ale stále postrádala podporu transakcí, poddotazů, cizích klíčů, ukládání procedur a pohledů. I přesto množství firem místo sofistikovanějších řešení od konkurenčních firem Oracle a Microsoft zvolilo MySQL s cílem ušetřit náklady na licenční poplatky a mít vyšší výkon.

V roce 2000 vznikla rozdělením společnost MySQL AB. Vyšla verze 3.23 s podporou transakcí. Největší novinkou ve verzi 4.0, která přišla v roce 2003, bylo přidání mezipaměti dotazům, která významně ovlivnila rychlost většiny aplikací.

Souběžně s verzí 4.1, která přinesla poddotazy, začal vývoj verze 5.0. Zatímco vývoj verze 4.1 se soustředil na stabilitu, tak verze 5.0 měla přinést velké množství novinek. Ve verzi 5.0 bylo

přidáno ukládání procedur, kurzory na straně serveru, triggery, zoptimalizování dotazování a další. Stabilní verze 5.0 vyšla v roce 2005. [20]

4.1.1 MySQL 5.6

Ve verzi 5.6 bylo přidáno silnější šifrovaní hesla a zároveň lze nyní vytvořit politiky na kontrolu síly hesla. Pomocí nově přidaného „memcached API“ mohou nyní vývojáři vytvářet mnohem výkonnější webové služby. Bylo vylepšeno dělení tabulek na menší spravovatelnější části.

Byla přidána funkce „Index Condition Pushdown“ (ICP), která převádí zpracování podmínky `WHERE` převážně na úložiště, což vede ke zvýšení výkonu snížením komunikace mezi databázovým serverem a úložištěm. Další funkce je „Multi-Range Read“ (MRR), která skenuje rozsahy indexů v dotazu a zrychluje tak skenování indexů a `JOIN` operace na indexovaných sloupcích. Dále byly zoptimalizovány dotazy, které současně používají klauzule `ORDER BY` a `LIMIT`, a to zejména na neindexovaných sloupcích. Bylo vylepšeno monitorování statistik indexů a jejich konzistence po restartu. Statistiky jsou tak přesnější a optimalizátor dotazů se může lépe rozhodovat, které indexy použít. [48]

4.1.2 MySQL 5.7

Verze 5.7 přinesla výrazné zvýšení výkonu a škálovatelnosti. Měla by být až třikrát rychlejší než předchozí verze. Byl zvýšen výkon dočasných tabulek. Byly přidány nové šifrovací algoritmy. Navíc je nyní možné vypnout a zapnout uživatelský účet. Byly přidány nové funkce replikace, které zvyšují dostupnost a výkon. Bylo vylepšeno „Performance Schema“, která se stará o monitoring. Dále byly přidány nové možnosti monitorování a zjednodušeno používání. Výrazně byl také předělán optimalizátor.

Byla implementována možnost přidat sekundární indexy na některé typy generovaných sloupců. Pro zlepšení rychlosti čtení/zápisu InnoDB byla odebrána funkce zamknutí indexu, která zamykala celou stromovou strukturu. Ta byla nahrazena funkcí zamknutí bloku, která zamyká pouze části stromu. Operace vytváření indexu jsou nyní rychlejší. Také byla přidána funkce „index level hints“, kde může administrátor určit indexy pro některé funkce optimalizátoru. „Full-text“ indexy nyní podporují použití externího parseru. Byla přidána podpora prostorových indexů (z angl. spatial indexes) založených na R-stromu. [47]

4.2 MySQL 8.0

Verze MySQL 8.0 přišla po verzi 5.7. Tento přeskok v číslování verzí má svůj důvod. Ještě předtím, než společnost MySQL AB koupila společnost Sun Microsystems, tak pracovali

vývojáři MySQL AB na verzi MySQL 6. Tento projekt byl ambiciózní a změna vlastnictví ho postihla natolik, že už vývoj MySQL 6 nepokračoval a tento projekt je již zapomenut. Verzování s číslem 7 používá produkt MySQL Cluster, který je již vyvíjen léta. Modernizace verze 5.7 a přidané novinky přesvědčili vývojáře, že další verzí bude tedy MySQL 8.0. [27]

4.2.1 Novinky verze MySQL 8.0

Podpora Unicode

Výchozí znaková sada v MySQL 8 je *utf8mb4*. Ve starších verzích to bylo kódování *latin1*. Standard UTF-8 kódování používá dnes většina webových serverů. Přechod na UTF-8 usnadňuje vývojářům práci s textem a možnostmi vyhledávání.

Ukládání do dokumentů, použití jako „NoSQL“

Webová technologie je založena na dokumentech. Tradiční struktura relačních databází je opak nestrukturovaných webových dat uložených v dokumentech. Tento fakt byl hlavním důvodem vzestupu NoSQL databází jako je například MongoDB. MySQL 8 představila možnost ukládání do dokumentů. Lze si tak vybrat mezi tradičním relačním modelem databáze a dokumentovým schématem.

Vylepšená podpora formátu JSON

Podpora formátu JSON je významně zlepšena. Byly přidány tabulkové, agregační a slučovací funkce. Bylo vylepšeno seskupování a řazení. Vylepšená konzistentnost transakcí nyní poskytuje návrat na bod obnovy (angl. rollback) u JSON dat. V nové verzi je navíc vylepšena validace JSON dat. Nyní se tedy například kontroluje, jestli je číselný datový typ ve specifikovaném rozmezí. [28][29]

Datový slovník

Doteď byla metadata databáze uložena ve složce s databází v podobě množství malých souborů. Toto staré řešení zde bylo z důvodu ochrany metadat v případě pádu. Technologie úložiště *InnoDB,* která je MySQL využívána je odolná vůči pádu a ztrátě dat. Nyní jsou metadata uloženy tedy přímo v databázi. Zároveň byl odstraněn s tím související limit pro počet tabulek. To také zlehčuje používání `ALTER TABLE` příkazů. Nový datový slovník umožňuje start transakcí a otestování změn před jejich potvrzením. [28]

Jak je uvedeno ve zdroji [29], MySQL 8 by měla být až dvakrát rychlejší než její předchozí verze. Byl zde kladen důraz na zlepšení výkonu V/V (vstupně-výstupních) operací a rychlosti čtení a zápisu při zatížení. Vývojáři se zaměřili na efektivnější využití úložných zařízení.

Databázový systém by tak měl lépe využívat hardwarové prostředky, disponovat nižší odezvou a efektivněji souběžně zpracovávat transakce a obecně zvládnout vyšší zátěž. [29]

4.3 Architektura

Na rozdíl od databází od firem Oracle a Microsoft je MySQL velmi flexibilní. Nabízí různé technologie úložiště a díky tomu může sloučit různým účelům. Protože různé technologie úložiště používají svoje specifické komponenty a mění tak vnitřní fungování databáze, bude tato práce zaměřena na databázový systém MySQL s hlavní a výchozí technologií úložiště InnoDB. [33]

MySQL je založena na modelu klient-server stejně jako databázové systémy od firem Microsoft a Oracle. Jedná se o třívrstvou architekturu. První vrstva se stará o připojení, autentizaci a bezpečnost. Druhá vrstva se stará o aplikační logiku jako je dotazování, optimalizaci vestavěné funkce a další. Poslední vrstva se stará o data. O jejich ukládání a vyzvedávání.

4.3.1 MySQL klient-server protokol

Protokol je poloviční duplex (z angl. half-duplex), to znamená, že server může kdykoli přijímat a odesílat data, ale nemůže dělat obojí současně. Klient tedy odešle dotaz na server a poté už jen čeká na odpověď.

Každé sezení (z angl. session) klienta má na serveru své vlastní vlákno. Toto vlákno poté vykonává dotazy klienta. Server je dokáže držet ve vyrovnávací paměti, takže nemusí vlákna zaniknout a být znovu vytvořená s každým novým připojením. Od verze MySQL 5.5 navíc dokáže malá skupina vláken obsloužit větší množství připojení. Server poté klientovi pošle odpověď nebo vyhodí chybu. V praxi to vypadá, že klient výsledky stahuje ze serveru v reálném čase, ale ve skutečnosti je pouze přijímá a nemůže již posílání probíhající na serveru zastavit.

4.3.2 Aplikační vrstva

Prvním krokem serveru při obdržení dotazu je, že se podívá do mezipaměti dotazů (z angl. query cache) a pokud zde dotaz najde, tak rovnou výsledek pošle zpět. Velikost této mezipaměti je definována parametrem a alokuje se při startu serveru. Pokud dotaz není v mezipaměti, musí se pro něj vytvořit exekuční plán, a to má několik kroků. Parsování, zpracování a optimalizaci. Během těchto procesů může také celý proces skončit chybou.

Parser zpracuje dotaz na tokeny, které jsou uspořádány do stromové struktury. Dále kontroluje, jestli jsou tokeny validní z hlediska syntaxe a jsou ve správném pořadí. Poté kontroluje

preprocesor existenci sloupců, tabulek a nakonec oprávnění. Protože k výsledku dotazu se lze dopracovat různými cestami, tak dotaz zpracuje ještě optimalizátor. Ten by měl zvolit nejefektivnější variantu. Každé variantě optimalizátor určí její „cenu“ a zvolí tu „nejlevnější“ variantu. Ve zdroji [34] je zdůrazněno, že optimalizátor nezvolí pokaždé tu nejlepší variantu. Mezi důvody patří neúplná přesnost využívaných statistik, odhadovaná cena dotazu může být v realitě jiná. Ideální varianta podle optimalizátoru nemusí být ideální pro potřeby klienta. Dále nejsou brány v potaz souběžně běžící dotazy a také není počítáno s cenou operací, kterou nemá pod kontrolou, jako jsou uložené a uživatelem definované funkce.

MySQL na rozdíl od většiny databází negeneruje exekuční plán v bajtkódu (z angl. bytecode), ale jako strom instrukcí. Tyto instrukce zpracuje „query execution engine“, který komunikuje s úložištěm a vrátí požadovaný výsledek. [34]

4.3.3 InnoDB

Technologie úložiště InnoDB podporuje transakce a splňuje vlastnosti ACID. Nabízí funkce `REDO` a `UNDO`. První slouží v případě pádu a umožňuje znovu vykonat nedokončenou transakci. `UNDO` zase umožňuje vracet data do stavu před transakcí. Je to obdoba funkce `ROLLBACK`, kterou používá databázový systém od Microsoftu. Fyzická data jsou ukládána do logických struktur.

Úložiště InnoDB je rozděleno na tabulkové prostory. Každý tabulkový prostor se skládá ze stránek, rozsahů a segmentů.

- Stránky (z angl. pages) – Nejmenší a základní datový blok v InnoDB. Výchozí velikost je 16 KB. Velikost je závislá na velikosti řádku. Můžou být také nazývány bloky.
- Rozsahy (z angl. extents) – Skládá se ze skupiny stránek. Velikost rozsahu může být až 1 MB. Pro lepší propustnost dat zapisuje/čte InnoDB celé rozsahy najednou.
- Segmenty – Kolekce souborů v tabulkovém prostoru InnoDB. Jeden segment obsahuje 4 rozsahy.

Významnou součástí InnoDB je *InnoDB buffer pool*. Jedná o hlavní paměť, která slouží jako vyrovnávací a jsou v ní uložena data tabulek a indexy. Paměť je sdílená mezi všemi sezeními. Jsou zde uloženy nejčastěji používaná data. Dalšími komponentami jsou *change buffer* a *redo log buffer*. První jmenovaný drží v mezipaměti změny a druhý zajišťuje data pro obnovu transakcí.

Systémový tabulkový prostor ukládá veškerá metadata například o tabulkách, sloupcích a indexech. Hlavní tabulkový prostor ukládá data tabulek. „REDO log soubory“ jsou používaný na obnovu po pádu systému a transakce, které se nestihly dokončit jsou znovu spuštěny po startu systému. „UNDO tabulkový prostor“ spravuje konzistenci čtení zachováváním původních dat, které jsou aktuálně modifikovány aktivní transakcí. Dočasný tabulkový prostor je využíván pro dočasné tabulky, které mohou vzniknout během obnovování databáze nebo stavu před transakcí.

Alternativami pro technologii InnoDB jsou například NDB pro MySQL Cluster. Netransakční MylSAM zaměřená na vysokou rychlost čtení. MEMORY, kde jsou všechna data uložená v paměti. CSV ukládající data do souborů stejnojmenného formátu. ARCHIVE pro ukládání velkého množství komprimovaných a neindexovaných dat. [33]

4.3.4 Datové typy

Datové typy v MySQL jsou rozděleny do tří kategorií. Řetězcové, číselné a časové datové typy. Mezi často používané patří:

- INT(velikost) – Integer Data Type je pro ukládání celých čísel. Jedná se o 32bitové číslo, které může být kladné i záporné. Velikost specifikuje maximálně počet číslic. Před typ můžeme dát ještě klíčové slovo „unsigned“ , tzn. že číslo bude vždy kladné a vejde se do něj dvakrát větší číslo. Typů INT je celkem pět a liší se velikostí čísla. Další jsou TINYINT, SMALLINT, MEDIUMINT a BIGINT, který je největší a může obsahovat 64bitové číslo.
- DOUBLE(velikost, zaokrouhlení) – Ukládá 64bitové desetinné číslo s přesností na šestnáct desetinných míst. Velikost specifikuje opět maximální počet číslic, který by měl být větší než druhý parametr. Ten určuje na kolik míst bude číslo automaticky zaokrouhleno. FLOAT je 32bitová obdoba DOUBLE, která ukládá číslo s maximální přesností na 8 desetinných míst.
- BOOLEAN, BOOL, TINYINT(1) – MySQL nepodporuje přímo datový typ boolean. Využívá proto TINYINT(1). Všechny tři zápisy jsou synonyma. Lze uložit pouze dvě hodnoty. Nepravda je reprezentována jako nula a pravda jako číslo jedna.
- VARCHAR(velikost) – Slouží pro ukládání řetězců o velikosti až 16 bitů. Nelze však překročit velikost danou parametrem. V paměti však bude zabírat velikost vloženého řetězce nikoliv stanovený limit.

- LONGTEXT(velikost) – Vhodný pro ukládání dlouhých řetězců i celých článků. Má více možností než VARCHAR. Lze uložit až 64bitový řetězec. Existují jeho modifikované varianty TEXT, TINYTEXT, MEDIUMTEXT a liší se pouze maximální velikostí.
- DATETIME – Datový typ pro datum a čas. Lze zadat datum od „1000-01-01 00:00:00“ až do „9999-12-32 23:59:59“. Definice sloupce umožňuje nastavit „DEFAULT“ a „ON UPDATE“ díky kterým se automaticky vloží aktuální datum a čas při inicializaci a modifikaci záznamu.

[22]

5 INDEX

Index ukládá hodnoty do indexovaného sloupce a je uložen v indexovém souboru. Je to samostatná struktura, která je spojená s datovým souborem. Jeho vytvoření a smazání tak nijak neovlivňuje data v tabulce. Index obsahuje hodnotu vyhledávacího klíče a adresu logického záznamu, kde se v datovém souboru nachází. Index lze přirovnat k vytvoření rejstříku v knize. Index je seřazený a když se poté hledají data, není nutné procházet všechna data standardně sekvenčně, ale použije se uspořádání u použitého typu indexu. Efektivita indexování se výrazně zvyšuje, čím méně záznamů hledáme. [39]

Indexovat je obecně vhodné sloupce, které jsou součástí klauzulí `WHERE`, `JOIN` a `GROUP BY`. Index tedy významně zvyšuje rychlost vyhledávání dat v tabulce, ale nese s sebou také negativní stránky. Zabírá místo na disku a také v paměti. Zároveň má i negativní dopad na výkon u operací vkládání, modifikovaní a mazání záznamu. Po každé této operaci musí být index znovu zorganizován. Je tedy nutné zvážit každou situaci individuálně a určit, zda se indexovat vyplatí. [51]

Tabulka bez indexu se při hledání hodnoty musí prohledávat celá (angl. full table scan) a hledání je s přibývajícími daty náročnější na výkon. Při použití indexu se přistupuje přímo k hledaným datům podle klíče indexu. Díky tomu se lze vyhnout zamčení celé tabulky (z angl. full table lock) při vyhledávání bez indexace a lze tak současně provádět na tabulce DML operace. Indexování je výhodné provádět při hledání co nejmenšího počtů řádků a na řádky, které jsou často dotazované. [52]

Příkazem `CREATE INDEX` lze vytvořit index pro jeden nebo i kombinaci sloupců. Jedna tabulka může mít definovaný libovolný počet indexů. `CLUSTER` je speciální typ indexu, který může definován pouze jeden na tabulku. Příkazem `UNIQUE` označujeme primární index. Většina indexů je v dnešní době implementována pomocí B-stromů. [3]

Viditelnost a použitelnost indexu

Index může být označen jako nepoužívaný a tím lze zvýšit výkon DML operací. Nepoužívaný index nebude používat ani optimalizátor dotazů a nebude zabírat ani místo na disku. Toho lze využít, když bude index potřeba později. Není nutné ho mazat. Neviditelný index je ignorován optimalizátorem dotazů, ale lze ho využívat „manuálně“ a to zejména pro účely testování. Ovlivňuje výkon DML operací, protože je při nich index aktualizován. [52]

Clustered (primární) a Non-Clustered (sekundární) index

Clustered index, nazývaný také jako primární index definuje, v jakém pořadí jsou data fyzicky uložena v tabulce. Data mohou být seřazena pouze jedním způsobem, a proto může být pro každou tabulku pouze jeden clustered index. Clustered index většinou vytváří primární klíč, ale může být vytvořen i na kombinaci sloupců. Clustered index seřazuje hodnoty v sestupném pořadí podle hodnoty indexovaného sloupce a v tomto pořadí budou data uložena i fyzicky. Pokud bude přidána nová hodnota, tak bude fyzicky uložena na příslušnou pozici podle indexu. Pokud má být vytvořen nový clustered index musí se nejprve smazat původní.

Non-clustered index, také nazývaný sekundární index neseřazuje fyzicky data v tabulce. Většinou slouží k vyhledávání hodnot pomocí sloupců, které nejsou primárním klíčem. Non-clustered index je uložen na jiném místě než tabulka, a to umožňuje vytvářet více non-clustered indexů pro jednu tabulku. To také znamená, že zabírají místo na disku navíc oproti clustered indexu. Data v tabulce tedy budou pořád uložena podle clustered indexu. V non-clustered indexu jsou data uložená ve specifickém pořadí. Jsou zde uloženy hodnoty indexovaného sloupce a adresa záznamu v tabulce, ke kterému hodnota patří. Pokud je vykonáván dotaz proti indexovanému sloupci, databáze jde nejprve do souboru indexu a získá adresu záznamu v tabulce. Poté jde do tabulky, kde získá data z ostatních sloupců daného záznamu. Kvůli tomuto kroku navíc je non-clustered index pomalejší než clustered index. [50]

Index založený na B-stromě

Většina databázových systémů používá výchozí index založený na B-stromě. Datová struktura B-strom je podobná binárnímu stromu s tím rozdílem, že zatímco v binárním stromě má každý prvek další dva potomky u B-stromu může být potomků libovolně. Je vhodný zejména pro porovnávání. Každý list stromu odpovídá jedné indexované hodnotě, která odpovídá jednomu záznamu. [38]

Index bitmapový

Bitmapový index je záležitostí databázového systému od společnosti Oracle. Bitmapový index vytvoří pro každou indexovanou hodnotu sérii `ROWID` všech záznamů. Poté se u dané indexované hodnoty pro každé `ROWID` vyplní jednička nebo nula, to znamená, jestli daný záznam hodnotu obsahuje (1) nebo ne (0).

Výhoda bitmapového indexu je, že se skládá z bitů a zabírá násobně méně paměti než index na B-stromě. Čím méně nabývá indexovaný sloupec rozdílných hodnot, tím je rozdíl v obsazení paměti větší. Z toho plyne, že je vhodné na sloupce, které nabývají například pouze hodnoty pravda/nepravda použít bitmapový index. Platí to také pro sloupce vyplněné `NULL` hodnotou.

Bitmapový index má jednu velkou nevýhodu. V případě přidání, smazání nebo modifikace řádku je potřeba index synchronizovat. V případě přidávání záznamu s indexovanou hodnotou jsou všechny záznamy s tou stejnou hodnotou včetně nových vložení zamčeny, dokud není potvrzeno přidání tohoto záznamu. A to platí i pro změnu indexované hodnoty, všechny záznamy se starou nebo novou hodnotou jsou zamčeny a nelze s těmito hodnotami přidávat v tu chvíli ani nové. Proto jsou bitmapové indexy nevhodné pro aplikace, kde více sezení najednou mění data v jedné tabulce a tyto aplikace v reálném světě převažují. Indexy založené na B-stromě tímto problémem netrpí a jejich použití je pro tyto typy aplikací vhodnější. [35]

Index založený na hashovací tabulce

Index založený na hashovací tabulce vytváří hash hodnotu pro každý klíč v indexu. Hodnota pak určuje, kam bude klíč umístěn v indexu. Jsou rychlejší, pokud hledáme jeden nebo menší množství výsledků. Od začátku zabírá v paměti fixní velikost. Tento typ indexu není často používán. [38][36]

Index Hint

Z překladu „nápověda pro index“. Každý databázový systém má svůj optimalizátor dotazů. Ten se snaží zvolit „nejlevnější“ cestu provedení dotazu a získání výsledků. Jeho algoritmy jsou na to přesně dělané a ve většině případů vrací nejlepší řešení provedení dotazu. Optimalizátor nemusí být vždy dokonalý nebo neví informace navíc, které zná administrátor a poté je potřeba použít jinou cestu, než kterou volí optimalizátor. V tomto případě lze využít nápovědu pro index. Administrátor vynutí použití jím zvoleného indexu v dotazu. Nejedná se tedy o „nápovědu“ spíše o příkaz. Dotaz může být s použitím „hintu“ opravdu rychlejší než původní verze optimalizátoru. Je však doporučeno použít tuto nápovědu pouze jako krátkodobé řešení například při odlaďování a testování. Algoritmy optimalizátoru jsou podloženy léty vývoje a jejich řešení bývá z dlouhodobého hlediska nejefektivnější. Vynucení indexu může přinést krátkodobý efekt, ale to neznamená, že tyto výkonnostní benefity budou dále trvat i po přidání velkého množství dalších dat. [49]

5.1 Kategorie

Tyto typy indexů se v identické nebo mírně modifikované podobě nachází ve všech třech databázových systémem, kterými se práce zabývá.

Jednoduchý index

Také nazývaný normální. Základní index, který nemusí být unikátní a může mít prázdnou hodnotu. [38]

Složený index (z angl. composite index)

Je to index složený z více sloupců. Na pořadí sloupců záleží. A to z toho důvodu, že není nutné používat všechny sloupce k dosažení výkonnostních benefitů. Složený index se řídí podle „leftmost prefix“, to znamená, že stačí hledat podle libovolného počtů sloupců definovaných zleva. V případě čtyř indexovaných sloupců lze tedy hledat pouze podle prvního, prvních dvou, prvních třech nebo všech čtyřech sloupců viz tabulka 2. [51]

Tabulka 2 – Schéma složeného indexu

Zdroj: vlastní zpracování

složený index:	(sloupec1, sloupec2, sloupec3, sloupec4)
automaticky lze použít také indexy:	(sloupec1, sloupec2, sloupec3)
	(sloupec1, sloupec2)
	(sloupec1)

Unikátní indexy (z angl. unique indexes)

Zaručuje, že uložená hodnota ve sloupci se vyskytne v celé tabulce pouze jednou. Tento typ indexu je většinou vytvářen automaticky a současně s primárním klíčem.

Indexy založené na funkci (z angl. function-based indexes)

Jedná se o indexy, u kterých je při vytváření na sloupec aplikována nějaká funkce. Může to být například funkce *upper*, která změní všechna písmena v řetězci na velká. Je třeba brát v potaz, že Microsoft databáze na rozdíl od databází MySQL a Oracle nerozlišuje malá a velká písmena.

Sestupné indexy (z angl. descending indexes)

Indexy jsou datovou strukturou, kde má každý záznam své místo a data jsou logicky řazená. Ve výchozím nastavení jsou řazená od nejmenšího k největšímu. A tímto indexem můžeme nastavit opačné pořadí. Toto má význam, pokud máme více výsledků a chceme je mít v sestupném pořadí. Operace `ORDER BY` a `DESC` jsou totiž dražší na prostředky a tento index pro takovou situaci je efektivnější řešení.

Textové indexy (z angl. text indexes)

Tyto indexy jsou navrženy pro složitější vyhledávání slov v řetězcových datech, které obsahují dlouhé texty například obsahy článků. Každé klíčové slovo z řetězce indexuje zvlášť. [35]

Prostorový index (z angl. spatial index)

Zaměřuje se na efektivní provádění operací se sloupci, které jsou geometrického datového typu. Tento index snižuje počet objektů, na které mají být tyto operace aplikovány. Tyto operace bývají náročné na prostředky.

JSON vyhledávací indexy (z angl. JSON search indexes)

Jedná se vlastně o index založený na funkci. V tomto případě je použita funkce *json_value*. Ta dokáže výrazně zrychlit dotazování v tabulkách, které jsou uloženy jako JSON dokumenty. [35]

5.2 Oracle Database

Index se vytváří na jeden nebo více sloupců. V případě použití více sloupců jedná o index „složený“. Pořadí sloupců poté hraje roli v tom, jestli optimalizátor index použije. Výchozí index v Oracle databázi je založený na B-stromě. [35]

Tabulky bez indexu (organizované na haldě) vkládají záznamy tam, kde se zrovna vejdou. Aplikace manipulují s indexovanými tabulkami stejně jako s těmi na organizovanými na haldě. V tabulce 3 jsou podrobněji popsaný rozdíly mezi tabulkou organizovanou na haldě a indexově organizovanou.

Tabulka 3 – Srovnání tabulek podle způsobu organizace

Zdroj: Přepracováno podle (Oracle Corporation, 2015)

Organizovaná na haldě	Indexově organizovaná
ROWID identifikuje záznam. Primární klíč je volitelný.	Primární klíč musí být definovaný, identifikuje záznam.
K záznamům lze přistupovat přímo pomocí ROWID.	K záznamům se přistupuje nepřímo pomocí primárního klíče.
Mohou být uložené v tabulkovém clusteru.	Nemohou být uložené v tabulkovém clusteru.
Může obsahovat sloupce typu LONG.	Nemůže obsahovat sloupec typu LONG.
Může obsahovat virtuální sloupce.	Nemůže obsahovat virtuální sloupce.

V Oracle databázi se vyskytují čtyři základní schémata indexů, které se doplňují svojí funkcionalitou. Jsou to indexy založené na B-stromě, bitmapové indexy, indexy založené na funkci a aplikační doménové indexy.

Indexy založené na B-stromě

Jedná se o standardní typ indexů, který se hodí pro primární klíče a vysoce selektivní dotazy. Data jsou seřazena podle indexovaného sloupce. Balancovaný strom se skládá z větví, které slouží k hledání a z listů, kde jsou uloženy hodnoty. Strom je balancovaný proto, že všechny listy jsou na stejné úrovní zanoření. Proto načtení jakéhokoliv záznamu trvá stejně dlouhou dobu. Výška indexu je počet zanoření od kořene stromu k listu. Větvě ukládají minimálně prefix klíče potřebný k tomu, aby dokázaly rozhodnout, kam se budou dále větvit. Tato technika umožňuje uložit v každé větvi maximum dat. Každá větev také obsahuje ukazatel na další

dceřinou větev. Počet klíčů a ukazatelů je omezen velikosti bloku větve. Každý list obsahuje indexovanou hodnotu a odpovídající `ROWID` k určení umístění aktuálního záznamu. Bloky listů jsou také propojeny se sousedícími listy.

Index scan. Při hledání indexované hodnoty trvá najít hodnotu N operací, kdy N je výška indexu stromu. Pokud SQL dotaz přistupuje pouze k indexovaným sloupcům poté je hodnota načtena přímo z indexu. Pokud SQL dotaz přistupuje i k neindexovaným sloupcům prochází střídavě blok indexu a blok tabulky.

Full index scan. Databáze přečte celý index v pořadí od listu nejvíce vlevo k listu nejvíce vpravo. Tento typ skenování je dostupný při použití klauzule `WHERE` v příkazu, kde je odkazováno na indexovaný sloupec. Data jsou seřazená podle klíče indexu.

Fast full index scan. Databáze přistupuje k datum v indexu bez přístupu k tabulce. Data nejsou čtena v žádném konkrétním pořadí. Jedná se o alternativní variantu k procházení celé tabulky (angl. full table scan). Podmínkou je, že index musí obsahovat všechny dotazované sloupce a nesmí se dotazovat záznam, který by měl všechny hodnoty `NULL` (například jeden sloupec musí být označený jako `NOT NULL`).

Index range scan. Používá se, když jeden nebo více indexovaných sloupců je specifikovaných v podmínce. Podmínka musí vracet hodnoty `TRUE`, `FALSE` nebo `UNKNOWN`. Klíč obvykle odpovídá žádné, jedné nebo více hodnotám.

Index unique scan. Je opakem předchozího. S klíčem indexu může být spojen pouze jeden nebo žádný záznam. Používá se obvyklé ve spojení s operátorem „rovná se“. Po nalezení záznamu okamžitě končí, protože další záznam se stejným klíčem není možný.

Index skip scan. Dokáže logicky subindexovat složitý index a použít jednotlivé indexy odděleně. Těží z toho dotazy, kdy jeden sloupec má minimum unikátních hodnot a druhý sloupec indexu jich má mnoho.

Index clustering factor. Měří seřazení řádků ve vztahu k indexované hodnotě. Jedná se o hrubý odhad počtů operací potřebných k přečtení celé tabulky. Čím je faktor nižší, tím mají rozsahy klíčů sklon být ve stejném datovém bloku. Databáze tak nemusí číst znovu stejné datové bloky.

Key Compression. Oracle databáze umožňuje kompresi části primárních klíčů, čímž lze ušetřit místo, které bude index zabírat.

Bitmapové indexy

Každý záznam indexu ukazuje na více řádků oproti B-stromu, kde každý index ukazuje na jediný řádek. Je vhodný na sloupce s nízkou kardinalitou (malý počet unikátních hodnot). Většinou tedy pro sloupce, kde je násobně méně unikátních hodnot, než je záznamů v tabulce nebo pro tabulky, které jsou určené především pro čtení a nejsou tolik zatěžovány DML operacemi. To proto, že indexovaný klíč ukazuje na všechny záznamy se stejnou hodnotou klíče a ty všechny musí při DML operaci zamknout.

Bitmap join index

Lze vytvořit index na záznamy jedné tabulky a jako klíč mu dát požadované hodnoty z druhé tabulky. Není potřeba `JOIN` tabulek, protože index již obsahuje požadovanou informaci.

Indexy založené na funkci

Tento typ indexů je založený na sloupci, jehož data jsou transformována funkcí nebo aritmetickým výrazem. Můžou být založené na B-stromu nebo jako bitmap index. Jsou efektivní na dotazy, které obsahují funkci ve `WHERE` klauzuli. Pokud funkce v dotazu není, tak nejsou použity.

Aplikační doménové indexy

Jeden z nejméně používaných typů indexů. Jedná se o specializovaný index, jehož strukturu si může navrhnout sám vývojář. Tento typ indexu je vytvořen pro data ve specifické doméně. Například dokumenty, obrázky nebo videa. Tradiční indexová struktura nemusí být použita a může být uložen v databázi jako tabulka nebo externě jako soubor.

V databázi se většinou neindexují řádky, ve kterých je klíč `NULL`. Výjimkou jsou bitmapové indexy.

Ukládání indexů

Databáze ukládá data indexu do index segmentu. Datový blok obsahuje hlavičku bloku, vstupní hlavičku, `ROWID` a jeden bajt pro každou hodnotu indexu. Tabulkový prostor segmentu je ve výchozím nastavení tabulkový prostor vlastníka nebo je specifikovaný při vytváření indexu. [52]

Indexy s reverzní klíčem (z angl. reverse key indexes)

Jedná se o indexy, které jak již název vypovídá ukládají svoje hodnoty v opačném pořadí bajtů. Používají se například u sekvenčních primárních klíčů a řeší problém, kdy se více sezení najednou snaží vložit nový záznam. Tento problém se často řeší jinými způsoby a toto je vzácně používané řešení. [35]

5.3 Microsoft SQL Server

Špatný návrh nebo nedostatek indexů jsou hlavní zdroje slabého výkonu databázové aplikace.

Index jako takový je uložen ve stránkách, které tvoří indexové stránky v Microsoft SQL Serveru (dále jen „SQL Server“). Používá se zde stromová analogie, kde první stránka indexu se nazývá kořenová stránka. Ta může odkazovat na další indexové stránky (větve), které nakonec budou odkazovat na koncové stránky, kde jsou ukazatele na konkrétní data (listy).

SQL Server index je uložen na disku nebo v paměti a je spojený s tabulkou nebo pohledem. Index obsahuje klíče vytvořené z jednoho nebo více sloupců. Pro indexy na disku platí, že klíče jsou uloženy ve struktuře B-stromu. Index ukládá data logicky organizovaně jako tabulku s řádky a sloupci a fyzicky řádkově ve formátu zvaném sklad řádků (z angl. rowstore) nebo sloupcově ve formátu zvaném sklad sloupců (z angl. columnstore).

Výběr správných indexů pro databázi a její zatížení je komplexní úkon, který musí vyvažovat rychlost dotazů a cenu prostředků za provádění DML operací. Jednoduché indexy zabírají méně místa na disku a vyžadují menší režii při údržbě. Složitější indexy na druhou stranu pokryjí větší množství dotazů. Indexy mohou být přidány, upravovány nebo mazány bez vlivu na databázové schéma nebo aplikační návrh.

Optimalizátor dotazů v SQL serveru většinou spolehlivě vybírá nejefektivnější index. Návrh indexů by měl být tedy rozmanitý, aby měl optimalizátor dostatečný výběr a mohl zvolit nejefektivnější variantu. Redukuje se tak čas na analýzu a zlepšuje se výkon v množině různých situací. Který index při vykonávání dotazu optimalizátor zvolí lze zobrazit v exekučním plánu. Index nemusí vždy zlepšovat výkon a optimalizátor index použije pouze tehdy, pokud skutečně dojde ke zlepšení výkonu.

Při návrhu indexu je tedy nutné brát v potaz, jestli se bude jednat o databázi s častým modifikováním dat, kde je potřeba vysoká propustnost nebo datový sklad, který musí velmi rychle zpracovat velké množiny dat. Dále je potřeba brát v potaz charakteristiku nejčastěji používaných dotazů. Porozumět charakteristice sloupců používaných v dotazech. Pro index jsou ideální sloupce číselného datového typu a ty které nejsou prázdné.

V SQL Serveru lze použít aplikaci „Database Engine Tuning advisor“, které zanalyzuje databázi a doporučí, které indexy vytvořit. [55]

Columnstore index

Ukládá a spravuje data na sloupcově založeném úložišti a využívá zpracování dotazů založené na sloupcích. Je vhodný pro velké datové sklady, které jsou určené primárně pro čtení dat. Vyznačuje se vysokým výkonem dotazů a možnostmi významné komprese dat.

Optimalizovaný pro paměť neclusterový index (z angl. memory-optimized nonclustered index)

Je podobný Hash indexu. Je založený na speciální datové struktuře nazývané „Bw-Tree“, což je modifikovaná varianta datové struktury B-strom.

Dále je zde dostupný index se sloupci. Jedná se o neclusterový index, který je rozšířen o neklíčové sloupce navíc.

Filtrovaný index

Optimalizovaný neclusterový index. Navrhnutý speciálně pro dotazy, které vyhledávají v dobře definovaných podmnožinách dat. Používá filtr a tím definuje indexy pouze specifickým řádkům. Tím se zvyšuje efektivita a index zabírá také méně místa na disku. [36]

5.4 MySQL

MySQL se vždy snaží vyhnout prohledávání největšího množství záznamů a pokud má na výběr z více indexů používá ten nejvíc selektivní. Některé typy indexů se v MySQL mohou mírně lišit ve funkčnosti podle typu použitého úložiště. [37]

Indexy v MySQL jsou založené na B-stromě nebo hashovací tabulce a prostorový index výjimečně na R-stromě. InnoDB navíc nabízí adaptivní hash index, který funguje jako vyrovnávací paměť. Ukládá často hledané dotazy, ke kterým poté umožňuje rychlý přístup. Pokud dotaz uložený nemá, tak použije B-strom. [38]

MySQL používá indexy pro rychlé dotazování na záznamy s klauzulí `WHERE`. Dále je využívá pro eliminaci největšího počtu záznamů. Pokud je na výběr z více indexů, tak se vybere ten nejvíce selektivní. To znamená sloupec s nejvíce unikátními daty. Zbyde potom nejméně záznamů.

Při dotazování dat s použitím klauzule `JOIN` dokáže MySQL efektivně použít indexy na sloupcích, které jsou stejného datové typu a velikosti. Například `VARCHAR(10)` a `CHAR(10)` jsou stejného typu a mají také deklarovanou stejnou velikost, proto lze použít index. Pokud se jedná o nebinární řetězcové sloupce, tak musejí používat stejnou znakovou sadu jinak index

nelze použít. Porovnání odlišných sloupců jako je například řetězcový s číselným, zabrání použití indexů, pokud nemohou být porovnány přímo bez konverze.

Pokud má být tabulka seřazena pomocí indexovaného klíče sestupně, tak klíč je přečten v obráceném pořadí. Toho lze využít i při vzestupném řazení, pokud je vytvořený index sestupný. Tyto pravidla lze použít také na složený index, pokud jsou všechny sloupce ze složeného indexu, které mají být seřazeny s klauzulí `DESC`.

V některých případech lze dotaz optimalizovat tak, že není potřeba přístup k celým datovým záznamům. Pokud dotazujeme pouze sloupce, které jsou součástí v nějakém indexu, tak mohou být hodnoty načteny přímo z indexu pro větší rychlost. Takovým indexům se říká pokrývající indexy (z angl. covering indexes).

Indexy nejsou tak efektivní na malých tabulkách nebo na větších tabulkách, kde je dotazována většina záznamů z celé tabulky. V tomto případě je sekvenční čtení rychlejší než práce s indexem. [53]

Pokud je dotazována většina záznamů z celé tabulky, tak MySQL optimalizátor nepoužije index, i když je k dispozici. Pokud by však byl takový dotaz omezen klauzulí `LIMIT`, která by omezila výsledky na menší počet záznamů, tak bude použit index vždy.

Při použití klauzule `LIKE` s řetězcem, který má žolíkový znak (z angl. wildcard character) na začátku i na konci se používá „Turbo Boyer-Moore algoritmus“, který inicializuje vzorec pro daný řetězec a ten použije pro rychlejší vyhledání výsledků.

Hash indexy

Mohou být použity pouze pokud je úložný engine typu `MEMORY`. Jsou použity pouze pro porovnání, které používá operátor „rovná se“, kde jsou velice rychlé. Pro ostatní porovnání, které hledají rozsahy hodnot nejsou vhodné. Systémy, které spoléhají na tento typ indexu, který je vhodný pro hledání konkrétní hodnoty jsou označovaný jako „klíč-hodnota úložiště“ (z angl. key-value stores).

Optimalizátor nemůže použít hash index pro zrychlení operací s klauzulí `ORDER BY`, protože je nevhodný pro hledání dalšího záznamu v pořadí. S tím souvisí také to, že MySQL nedokáže přesně určit, kolik je záznamů mezi dvěma hodnotami.

Pouze celé klíče mohou být použity pro nalezení záznamu. Nelze využít „leftmost prefix“ pravidlo složeného klíče. [54]

6 MĚŘENÍ VÝKONNOSTI RELAČNÍCH DATABÁZÍ

6.1 Selektivita

Výkon dotazů výrazně ovlivňuje selektivita a s ní související velikost množiny hledaných výsledků. Ještě před pojmem selektivita je důležité vysvětlit pojem kardinalita. Kardinalita je počet unikátních hodnot v jednom sloupci. Bude-li vydělena kardinalita celkovým počtem záznamů v tabulce, tak výsledné číslo je průměrná selektivita. Selektivita konkrétní hodnoty je vydělení počtu záznamů s touto hodnotou celkovým počtem záznamů. Nízká selektivita a obecně hledání většího množství záznamů je pomalejší než hledání jednoho konkrétního záznamu. Tento rozdíl by měl ještě více zvýraznit index založený na B-stromě, který by měl přinést významné výkonnostní zlepšení v případě hledání konkrétní hodnoty. A naopak v případě nízké selektivity a hledání většího množství záznamů by se jeho výkon měl více přibližovat full table scanu. [58]

6.2 Všeobecná optimalizace dotazů

Dotaz (z angl. query) je požadavek na informaci z databáze. Dotaz může při spouštění využívat různé algoritmy a také lze dotaz, který vrací stejný výsledek zapsat více způsoby. Při optimalizaci dotazů je tedy hledána nejefektivnější varianta. Optimalizace dotazů je velmi důležitá a má následující dopady:

- Zvýšení rychlosti aplikace – Uživateli jsou výsledky zprostředkovány rychleji a lze tedy zvýšit celkový výkon aplikace.
- Snížení nákladů na provoz – Protože optimalizovaný dotaz je zpracován za kratší dobu, umožňuje to systému za stejný čas provést více operací.
- Snížení opotřebení a efektivnější využití hardwaru – Server je schopen běžet více efektivně. Sníží se počet operací zápisu a čtení z disku, menší využití paměti a méně času na procesoru. To zajistí také snížení spotřeby elektrické energie a prodloužení životnosti serveru.

I při rychlém vývoji hardwaru a možnostech jeho škálování nelze optimalizaci dotazů opomíjet. Mohlo by to vést k výkonnostním problémům v budoucnosti. [24]

Rychlost dotazů je komplexní problematika, která závisí na mnoho aspektech. Níže jsou uvedené nejdůležitější z nich.

Databázový model

Návrh databázového modelu je základem efektivního běhu celého databázového systému. Správně strukturované tabulky a jejich kardinalita předchází budoucím nejen výkonnostním problémům. Z hlediska rychlosti dotazování je také důležité volit vhodné datové typy. Nejlépe, tak aby jejich velikost byla co nejmenší a zároveň se do nich vešly požadované informace. Správný návrh databázového modelu je podrobně popsán v kapitole 1.2.1.

Hardware

Paměť RAM, procesory a pevné disky jsou z databázového pohledu nejdůležitější komponenty. Ke škálování (rozšiřování) hardwaru by se mělo přistupovat až po optimalizaci dotazů. Optimalizace dotazů by neměla být nahrazována přidáním dalších prostředků, ale spíše naopak. Zefektivnit chod celé databázového systému bez potřeby dalšího hardwarového rozšíření.

Indexy

Indexy mohou zejména složitější dotazy značně zrychlit. Uvádí se až o desítky procent. Indexem je již sám primární klíč. Efektivní je vybrat další používané sloupce a vytvořit index. V dnešní době disponují databázové systémy nástroji, které dokáží analyzovat dotaz a navrhnout vytvoření indexu. Ten lze vzápětí vygenerovat. Součástí bývá také výpočet zrychlení při použití takto vygenerovaného indexu.

Vhodná struktura dotazu

Různě strukturované dotazy mohou vést ke stejnému výsledku, ale mohou se lišit svým výkonem. Obecně platí zásady, které zefektivňují dotazování a je vhodné se jimi řídit, je-li to možné. Například `SELECT *` nahradit výčtem požadovaných sloupců, a to i v případě, že budou dotazovány všechny sloupce. Při použití hvězdičky se totiž navíc zjišťuje seznam všech sloupců tabulky. Dále je vhodné hlídat si klauzuli `FROM` v kombinaci s `WHERE`. Zejména při hledání ve více tabulkách je vhodné použít `INNER JOIN` jinak by hrozilo hledání dat omezených `WHERE` klauzulí v obou tabulkách. Může za to logické zpracování dotazu, které začne zpracovávat dříve klauzuli `FROM`. Mezi operace náročné na výkon se řadí také hledání řetězců pomocí klauzule `LIKE`. V případě jejího použití je vhodné se vyvarovat zástupného znaku procento, který zastupuje jakýkoliv a jakkoliv dlouhý text a jeho použití je z pohledu optimalizace neefektivní. Pokud to není nutné, je vhodné nepoužívat v dotazech také klauzuli `ORDER BY`. Ta seřazuje výsledky a je také náročná na výkon.

Pro efektivní optimalizaci dotazů je podstatná také volba dotazů, které se budou optimalizovat. Přednost by měly mít především náročné a často používané dotazy. Tím lze dosáhnout významně vyššího výkonu. [23]

6.3 Testovací prostředí

Zde bude popsán veškerý hardware a software, který poslouží pro účely testování a případný popis konfigurace.

6.3.1 Hardware

Měření bude prováděno na osobním ultrabooku s následující technickou specifikací:

Model: Dell ultrabook e7240

CPU: Intel Core i5-4210U 2,4 GHz

RAM: 16 GB

SSD: Crucial MX200 250 GB

OS: Windows 10 Pro, verze 21H1

6.3.2 Software

Docker

Jedná se o virtualizační nástroj, který využívá kontejnery a řeší některé nedostatky klasické virtualizace. Snižují se režijní náklady, protože není potřeba virtualizovat celý OS. Usnadněna je také portabilita a škálování. Nástroj Docker se poprvé objevil v roce 2013, ale předchůdci kontejnerové technologie tu byli již v sedmdesátých letech minulého století. Docker tedy umožňuje izolovat aplikaci se všemi jejími komponenty do kontejneru. Kontejnery pak běží v rámci jednoho virtualizovaného jádra OS a dokáží mezi sebou sdílet operační paměť, knihovny a další zdroje. [56]

Pro účely práce bude použit Docker ve verzi 20.10.7. Každý databázový systém poběží v samostatném kontejneru. Konfigurace Dockeru bude ponechána výchozí. Výchozí nastavení prostředků limituje pouze operační paměť a to na 8 GB, což bude pro účely práce dostačující.

Relační databázové systémy

Měření bude probíhat na aktuálních stabilních verzích databázových systémů:

- Oracle 19c Enterprise Edition
- Microsoft SQL Server 2019 Developer Edition

- MySQL 8.0.26

Verze *Developer Edition* je identická s *Enterprise Edition*. Rozdíl je pouze v licenci, kdy *Developer Edition* nelze používat pro komerční účely. Podrobnosti o databázových systémech jsou v samostatných kapitolách.

DataGrip

DataGrip je nástroj pro pokročilou správu různých databázových systémů od společnosti JetBrains.

V prostředí DataGripu budou připojeny všechny tři databázové systémy a budou zde spouštěny veškeré příkazy a dotazy nad databázemi. V nastavení byla navýšena operační paměť z původních 2 GB na 8 GB z důvodu plynulejší práce s větším množstvím dat a minimalizace šance neočekávaného pádu prostředí. Verze DataGripu je 2021.1.3.

6.4 Scénář měření

Všechny tři databázové systému poběží v Dockeru. V průběhu měření bude běžet pouze databázový systém, na kterém bude prováděno měření. Všechny tři databázové systémy poběží ve výchozím nastavení a případné odlišnosti budou zmíněny v kapitolách týkajících se jednotlivých databázových systémů. Všechny databázové systémy, tak budou mít stejné podmínky a v průběhu měření nepoběží žádné zbytečné programy v hostitelském OS navíc.

Každý databázový systém bude obsahovat pět tabulek, kdy každá tabulka bude reprezentovat určitý počet dat. Měření bude probíhat na každé tabulce. Výsledek každého měření bude zaokrouhlený průměr tří identických měření za sebou, aby bylo dosaženo vyšší přesnosti. Dále bude zajištěno, aby před každým dotazem byla vymazaná související vyrovnávací paměť, která by mohla ovlivňovat výkon dalších měření.

Měřena bude celková rychlost dotazu, která se skládá ze součtu doby provedení dotazu a doby načtení dat. Doba provedení dotazu je doba, ve které se příkaz spustí a identifikuje hledaná data a doba načtení dat je doba za kterou se identifikovaná data načtou.

6.4.1 Tabulky

Zdroj dat pro tabulky bude online filmová databáze IMDb, konkrétně dataset *title.basics.tsv.gz*. Tabulka bude obsahovat následující sloupce:

- tconst – Unikátní identifikátor titulu.
- titleType – Typ titulu, například jestli se jedná o seriál nebo film.

- primaryTitle – Název titulu, používaný filmaři k propagaci.
- originalTitle – Původní název, může být v původním jazyce.
- isAdult – Určuje je, jestli je film pouze pro dospělé.
- startYear – Rok vydání titulu, v případě seriálu začátek série.
- endYear – Rok ukončení seriálu, vyplněný pouze v případě seriálů.
- runtimeMinutes – Délka běhu titulu.
- genres – Obsahuje tři nejčastější žánry spojené s titulem. [57]

Stažená data budou uložena ve formátu CSV pro každou tabulku a budou vložena do databázového systému přes DataGrip. Pro každý databázový systém bude vytvořeno pět těchto tabulek:

- Tabulka movie100 – Obsahuje 100 záznamů dat.
- Tabulka movie1k – Obsahuje 1 000 záznamů dat.
- Tabulka movie10k – Obsahuje 10 000 záznamů dat.
- Tabulka movie100k – Obsahuje 100 000 záznamů dat.
- Tabulka movie1kk – Obsahuje 1 000 000 záznamů dat.

Struktura tabulky i s daty je předvedena na následujícím obrázku.

	tconst	titleType	primaryTitle	originalTitle	isAdult	startYear	endYear	runtimeMinutes	genres
1	tt0000001	short	Carmencita	Carmencita	0	1894	<null>	1	Documentar…
2	tt0000002	short	Le clown et ses …	Le clown et ses c…	0	1892	<null>	5	Animation,…

Obrázek 1 – Struktura tabulky s daty (vlastní zpracování)

6.4.2 Konfigurace měření

1) Dotaz bude proveden ve výchozím nastavení databáze bez indexu a případně se zabrání optimalizátoru vytvořit automatický index před provedením dotazu.
2) Na dotazované sloupce bude vytvořen B-strom index a případně se vynutí jeho použití.
3) Na databázi Oracle bude navíc vytvořen bitmapový index na dotazované sloupce a vynutí se jeho použití.

6.4.3 Objekty měření

a. Zaměření na maximální selektivitu – Bude dotazován konkrétní záznam ve sloupci *primaryTitle* umístěný zhruba v půlce tabulky.

b. Zaměření na střední selektivitu – Budou dotazovány záznamy ve sloupci *genres*.

c. Zaměření na žádnou až minimální selektivitu – Budou dotazovány záznamy ve sloupci *titleType*.

Následující tabulka zobrazuje přesný počet unikátních hodnot jednotlivých sloupců (kardinalitu) v tabulkách o daném počtu dat. Naznačuje tedy průměrnou selektivitu sloupců, které budou dotazovány.

Tabulka 4 – Počet unikátních hodnot ve sloupci – Naznačení průměrné selektivity

Zdroj: vlastní zpracování

Počet dat:	100	1 000	10 000	100 000	1 000 000
originalTitle	99	983	9 648	92 122	726 031
titleType	1	2	2	10	10
genres	14	85	270	1 090	1 720

6.5 Dotazy pro měření obecně

Všechny dotazy pro měření budou založeny na vzoru:

```
SELECT * FROM tabulka WHERE sloupec = 'hodnota';
```

Kde tabulka bude určovat množství dat, na kterém bude měřeno a sloupec bude určovat testovanou selektivitu. Konkrétní hodnota je vybrána tak, aby odpovídala specifikacím měření.

6.6 Základní měření

Jako základní měření bude sloužit dotaz, před kterým ani po něm se nebude mazat ani jiným způsobem manipulovat s vyrovnávací pamětí. Půjde o často opakovaný vzor dotazu, kdy bude vyhledána jedna konkrétní hodnota za pomoci klauzule `WHERE`. Dotaz bude testován na jediné tabulce, do které bude přidáváno po každém měření desetinásobek původních dat. Začátek měření bude se sto záznamy a u posledního dotazu bude tabulka obsahovat milion záznamů. Testován bude základní full table scan a bude vynuceno jeho použití, kvůli optimalizátorům jednotlivých systémů, které by mohli vytvořit a použít index. Následující dotaz bude pro databázový systém Oracle.

```
SELECT /*+ FULL(MOVIE100) */* FROM MOVIE WHERE "originalTitle" = 'Carga de
rurales';
```

Následný dotaz je obdobou předchozího pro Microsoft SQL Server.

```
SELECT * FROM test.movie WITH (INDEX(0)) WHERE originalTitle = 'Carga de
rurales';
```

A poslední pro MySQL.

```
SELECT * FROM movie USE INDEX () WHERE originalTitle = 'Carga de rurales';
```

6.7 Oracle Database

Před každým spuštěním dotazu bude potřeba nejdříve vymazat vyrovnávací paměť následujícími příkazy.

```
alter system flush buffer_cache;
alter system flush shared_pool;
```

6.7.1 Dotazy bez indexu

Aby bylo zajištěno, že optimalizátor dotazů nevytvoří vlastní index a nevyužije ho, tak je nutné vložit do dotazu hint. Ten bude vložen hned za výraz `SELECT` mezi lomítka, viz příkaz níže. Hint „FULL(nazev tabulky)“ zajistí, že tabulka se bude prohledávat celá bez indexu (angl. full table scan). Následují dotazy pro všech pět tabulek, které budou testovat rychlost dotazu s maximální selektivitou.

```
SELECT /*+ FULL(MOVIE1KK) */* FROM MOVIE1KK WHERE "originalTitle" = 'The
Island';
SELECT /*+ FULL(MOVIE100K) */* FROM MOVIE100K WHERE "originalTitle" = 'She
Devil';
SELECT /*+ FULL(MOVIE10K) */* FROM MOVIE10K WHERE "originalTitle" = 'The
Butterfly';
SELECT /*+ FULL(MOVIE1K) */* FROM MOVIE1K WHERE "originalTitle" =
'Esmeralda';
SELECT /*+ FULL(MOVIE100) */* FROM MOVIE100 WHERE "originalTitle" = 'Carga
de rurales';
```

Dalších pět dotazů bude ověřovat rychlost dotazů s žádnou až minimální selektivitou.

```
SELECT /*+ FULL(MOVIE1KK) */* FROM MOVIE1KK WHERE "titleType" = 'short';
SELECT /*+ FULL(MOVIE100K) */* FROM MOVIE100K WHERE "titleType" = 'short';
SELECT /*+ FULL(MOVIE10K) */* FROM MOVIE10K WHERE "titleType" = 'short';
SELECT /*+ FULL(MOVIE1K) */* FROM MOVIE1K WHERE "titleType" = 'short';
SELECT /*+ FULL(MOVIE100) */* FROM MOVIE100 WHERE "titleType" = 'short';
```

Poslední pět dotazů bez indexu, bude hledat podle sloupce *genres*, tedy střední selektivitou.

```
SELECT /*+ FULL(MOVIE1KK) */* FROM MOVIE1KK WHERE GENRES = 'Comedy,Short';
SELECT /*+ FULL(MOVIE100K) */* FROM MOVIE100K WHERE GENRES =
'Comedy,Short';
SELECT /*+ FULL(MOVIE10K) */* FROM MOVIE10K WHERE GENRES = 'Comedy,Short';
SELECT /*+ FULL(MOVIE1K) */* FROM MOVIE1K WHERE GENRES = 'Comedy,Short';
SELECT /*+ FULL(MOVIE100) */* FROM MOVIE100 WHERE GENRES = 'Comedy,Short';
```

6.7.2 Vytvoření indexů

Následují příkazy, které vytvoří indexy. Příkazy jsou pro vytvoření obecného indexu, který je založen na B-stromě. Pro vytvoření indexu je tedy potřeba název indexu, název tabulky a název indexovaného sloupce. Oracle index je jinak strukturovaný než index na MySQL a MS SQL Serveru a z toho důvodu je rozdíl v tom, že název indexu se musí lišit i když samotný index patří jiné tabulce.

```
CREATE INDEX MOVIEBtreeIndexOT ON MOVIE1KK("originalTitle");
CREATE INDEX MOVIEBtreeIndexOT1 ON MOVIE100K("originalTitle");
CREATE INDEX MOVIEBtreeIndexOT2 ON MOVIE10K("originalTitle");
CREATE INDEX MOVIEBtreeIndexOT3 ON MOVIE1K("originalTitle");
CREATE INDEX MOVIEBtreeIndexOT4 ON MOVIE100("originalTitle");

CREATE INDEX MOVIEBtreeIndexTT ON MOVIE1KK("titleType");
CREATE INDEX MOVIEBtreeIndexTT1 ON MOVIE100K("titleType");
CREATE INDEX MOVIEBtreeIndexTT2 ON MOVIE10K("titleType");
CREATE INDEX MOVIEBtreeIndexTT3 ON MOVIE1K("titleType");
CREATE INDEX MOVIEBtreeIndexTT4 ON MOVIE100("titleType");

CREATE INDEX MOVIEBtreeIndexG ON MOVIE1KK("GENRES");
CREATE INDEX MOVIEBtreeIndexG1 ON MOVIE100K("GENRES");
CREATE INDEX MOVIEBtreeIndexG2 ON MOVIE10K("GENRES");
CREATE INDEX MOVIEBtreeIndexG3 ON MOVIE1K("GENRES");
CREATE INDEX MOVIEBtreeIndexG4 ON MOVIE100("GENRES");
```

Vzhledem k tomu, že na databázovém systému od společnosti Oracle se bude testovat také bitmapový index, tak je nutné nejprve původní indexy smazat.

```
DROP INDEX MOVIEBtreeIndexOT;
DROP INDEX MOVIEBtreeIndexOT1;
DROP INDEX MOVIEBtreeIndexOT2;
DROP INDEX MOVIEBtreeIndexOT3;
DROP INDEX MOVIEBtreeIndexOT4;

DROP INDEX MOVIEBtreeIndexTT;
DROP INDEX MOVIEBtreeIndexTT1;
DROP INDEX MOVIEBtreeIndexTT2;
DROP INDEX MOVIEBtreeIndexTT3;
DROP INDEX MOVIEBtreeIndexTT4;

DROP INDEX MOVIEBtreeIndexG;
DROP INDEX MOVIEBtreeIndexG1;
DROP INDEX MOVIEBtreeIndexG2;
DROP INDEX MOVIEBtreeIndexG3;
DROP INDEX MOVIEBtreeIndexG4;
```

A poté založit nové. Bitmap index se vytvoří tak, že před klíčové slovo `INDEX` přijde klíčové slovo `BITMAP` a tím bude specifikováno, že nemá být vytvářen výchozí B-strom index, ale bitmapový.

```
CREATE BITMAP INDEX MOVIEBitmapIndexOT ON MOVIE1KK("originalTitle");
CREATE BITMAP INDEX MOVIEBitmapIndexOT1 ON MOVIE100K("originalTitle");
CREATE BITMAP INDEX MOVIEBitmapIndexOT2 ON MOVIE10K("originalTitle");
CREATE BITMAP INDEX MOVIEBitmapIndexOT3 ON MOVIE1K("originalTitle");
CREATE BITMAP INDEX MOVIEBitmapIndexOT4 ON MOVIE100("originalTitle");

CREATE BITMAP INDEX MOVIEBitmapIndexTT ON MOVIE1KK("titleType");
CREATE BITMAP INDEX MOVIEBitmapIndexTT1 ON MOVIE100K("titleType");
CREATE BITMAP INDEX MOVIEBitmapIndexTT2 ON MOVIE10K("titleType");
CREATE BITMAP INDEX MOVIEBitmapIndexTT3 ON MOVIE1K("titleType");
CREATE BITMAP INDEX MOVIEBitmapIndexTT4 ON MOVIE100("titleType");

CREATE BITMAP INDEX MOVIEBitmapIndexG ON MOVIE1KK("GENRES");
CREATE BITMAP INDEX MOVIEBitmapIndexG1 ON MOVIE100K("GENRES");
CREATE BITMAP INDEX MOVIEBitmapIndexG2 ON MOVIE10K("GENRES");
```

```
CREATE BITMAP INDEX MOVIEBitmapIndexG3 ON MOVIE1K("GENRES");
CREATE BITMAP INDEX MOVIEBitmapIndexG4 ON MOVIE100("GENRES");
```

6.7.3 Dotazy s indexem

Dotazy s indexem jsou téměř identické jako dotazy bez indexu. Liší se pouze hint, který zde vynucuje použití zvoleného indexu. Je to z důvodu, aby nezasáhl optimalizátor a neprovedl jinou variantu. Hint se tedy skládá z klíčového slova `INDEX` a v závorce je název tabulky následovaný názvem indexu. Následujících pět dotazů bude sloužit pro testování maximální selektivity.

```
SELECT /*+ INDEX(MOVIE1KK MOVIEBtreeIndexOT) */* FROM MOVIE1KK WHERE
"originalTitle" = 'The Island';
SELECT /*+ INDEX(MOVIE100K MOVIEBtreeIndexOT1) */* FROM MOVIE100K WHERE
"originalTitle" = 'She Devil';
SELECT /*+ INDEX(MOVIE10K MOVIEBtreeIndexOT2) */* FROM MOVIE10K WHERE
"originalTitle" = 'The Butterfly';
SELECT /*+ INDEX(MOVIE1K MOVIEBtreeIndexOT3) */* FROM MOVIE1K WHERE
"originalTitle" = 'Esmeralda';
SELECT /*+ INDEX(MOVIE100 MOVIEBtreeIndexOT4) */* FROM MOVIE100 WHERE
"originalTitle" = 'Carga de rurales';
```

Dalších pět pro testování žádné až minimální selektivity.

```
SELECT /*+ INDEX(MOVIE1KK MOVIEBtreeIndexTT) */* FROM MOVIE1KK WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE100K MOVIEBtreeIndexTT1) */* FROM MOVIE100K WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE10K MOVIEBtreeIndexTT2) */* FROM MOVIE10K WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE1K MOVIEBtreeIndexTT3) */* FROM MOVIE1K WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE100 MOVIEBtreeIndexTT4) */* FROM MOVIE100 WHERE
"titleType" = 'short';
```

A posledních pět pro testování střední selektivity.

```
SELECT /*+ INDEX(MOVIE1KK MOVIEBtreeIndexG) */* FROM MOVIE1KK WHERE GENRES
= 'Comedy,Short';
SELECT /*+ INDEX(MOVIE100K MOVIEBtreeIndexG1) */* FROM MOVIE100K WHERE
GENRES = 'Comedy,Short';
SELECT /*+ INDEX(MOVIE10K MOVIEBtreeIndexG2) */* FROM MOVIE10K WHERE GENRES
= 'Comedy,Short';
SELECT /*+ INDEX(MOVIE1K MOVIEBtreeIndexG3) */* FROM MOVIE1K WHERE GENRES =
'Comedy,Short';
SELECT /*+ INDEX(MOVIE100 MOVIEBtreeIndexG4) */* FROM MOVIE100 WHERE GENRES
= 'Comedy,Short';
```

Dalších patnáct dotazů provede totéž, co předchozích patnáct s tím rozdílem, že je použitý bitmapový index. Bitmapový index je speciální strukturou, která by měla v rychlosti předčit index na B-stromě v případech malé selektivity.

```
SELECT /*+ INDEX(MOVIE1KK MOVIEBitmapIndexOT) */* FROM MOVIE1KK WHERE
"originalTitle" = 'The Island';
SELECT /*+ INDEX(MOVIE100K MOVIEBitmapIndexOT1) */* FROM MOVIE100K WHERE
"originalTitle" = 'She Devil';
```

```
SELECT /*+ INDEX(MOVIE10K MOVIEBitmapIndexOT2) */* FROM MOVIE10K WHERE
"originalTitle" = 'The Butterfly';
SELECT /*+ INDEX(MOVIE1K MOVIEBitmapIndexOT3) */* FROM MOVIE1K WHERE
"originalTitle" = 'Esmeralda';
SELECT /*+ INDEX(MOVIE100 MOVIEBitmapIndexOT4) */* FROM MOVIE100 WHERE
"originalTitle" = 'Carga de rurales';

SELECT /*+ INDEX(MOVIE1KK MOVIEBitmapIndexTT) */* FROM MOVIE1KK WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE100K MOVIEBitmapIndexTT1) */* FROM MOVIE100K WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE10K MOVIEBitmapIndexTT2) */* FROM MOVIE10K WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE1K MOVIEBitmapIndexTT3) */* FROM MOVIE1K WHERE
"titleType" = 'short';
SELECT /*+ INDEX(MOVIE100 MOVIEBitmapIndexTT4) */* FROM MOVIE100 WHERE
"titleType" = 'short';

SELECT /*+ INDEX(MOVIE1KK MOVIEBitmapIndexG) */* FROM MOVIE1KK WHERE GENRES
= 'Comedy,Short';
SELECT /*+ INDEX(MOVIE100K MOVIEBitmapIndexG1) */* FROM MOVIE100K WHERE
GENRES = 'Comedy,Short';
SELECT /*+ INDEX(MOVIE10K MOVIEBitmapIndexG2) */* FROM MOVIE10K WHERE
GENRES = 'Comedy,Short';
SELECT /*+ INDEX(MOVIE1K MOVIEBitmapIndexG3) */* FROM MOVIE1K WHERE GENRES
= 'Comedy,Short';
SELECT /*+ INDEX(MOVIE100 MOVIEBitmapIndexG4) */* FROM MOVIE100 WHERE
GENRES = 'Comedy,Short';
```

6.8 Microsoft SQL Server

Následující dva příkazy budou sloužit pro smazání vyrovnávací paměti a budou spuštěny před každým dotazem.

```
DBCC FREEPROCCACHE;
DBCC DROPCLEANBUFFERS;
```

6.8.1 Dotazy bez indexu

I u SQL Serveru bude nutná pojistka, že optimalizátor nevytvoří a nepoužije vlastní index. Použitá zde budou klíčová slova `WITH` a `INDEX`, který má v parametru 0. Díky tomu se vynutí full table scan. Následujících pět příkazů otestuje sloupec s maximální selektivitou.

```
SELECT * FROM test.movie1kk WITH (INDEX(0)) WHERE originalTitle = 'The
Island';
SELECT * FROM test.movie100k WITH (INDEX(0)) WHERE originalTitle = 'She
Devil';
SELECT * FROM test.movie10k WITH (INDEX(0)) WHERE originalTitle = 'The
Butterfly';
SELECT * FROM test.movie1k WITH (INDEX(0)) WHERE originalTitle =
'Esmeralda';
SELECT * FROM test.movie100 WITH (INDEX(0)) WHERE originalTitle = 'Carga de
rurales';
```

Dalších pět sloupec s žádnou až minimální selektivitou.

```
SELECT * FROM test.movie1kk WITH (INDEX(0)) WHERE titleType = 'short';
SELECT * FROM test.movie100k WITH (INDEX(0)) WHERE titleType = 'short';
```

```
SELECT * FROM test.movie10k WITH (INDEX(0)) WHERE titleType = 'short';
SELECT * FROM test.movie1k WITH (INDEX(0)) WHERE titleType = 'short';
SELECT * FROM test.movie100 WITH (INDEX(0)) WHERE titleType = 'short';
```

A posledních pět je pro střední selektivitu.

```
SELECT * FROM test.movie1kk WITH (INDEX(0)) WHERE genres = 'Comedy,Short';
SELECT * FROM test.movie100k WITH (INDEX(0)) WHERE genres = 'Comedy,Short';
SELECT * FROM test.movie10k WITH (INDEX(0)) WHERE genres = 'Comedy,Short';
SELECT * FROM test.movie1k WITH (INDEX(0)) WHERE genres = 'Comedy,Short';
SELECT * FROM test.movie100 WITH (INDEX(0)) WHERE genres = 'Comedy,Short';
```

6.8.2 Vytvoření indexů

Protože je i u SQL Serveru výchozí index založený na B-stromě, tak je příkaz na jeho vytvoření identický s příkazem pro Oracle. Rozdíl je pouze v tom, že indexy, které jsou pro jinou tabulku mohou mít stejné názvy. Následující příkazy tedy vytvoří požadované indexy.

```
CREATE INDEX movieBtreeIndexOT ON test.movie1kk(originalTitle);
CREATE INDEX movieBtreeIndexOT ON test.movie100k(originalTitle);
CREATE INDEX movieBtreeIndexOT ON test.movie10k(originalTitle);
CREATE INDEX movieBtreeIndexOT ON test.movie1k(originalTitle);
CREATE INDEX movieBtreeIndexOT ON test.movie100(originalTitle);

CREATE INDEX movieBtreeIndexTT ON test.movie1kk(titleType);
CREATE INDEX movieBtreeIndexTT ON test.movie100k(titleType);
CREATE INDEX movieBtreeIndexTT ON test.movie10k(titleType);
CREATE INDEX movieBtreeIndexTT ON test.movie1k(titleType);
CREATE INDEX movieBtreeIndexTT ON test.movie100(titleType);

CREATE INDEX movieBtreeIndexG ON test.movie1kk(genres);
CREATE INDEX movieBtreeIndexG ON test.movie100k(genres);
CREATE INDEX movieBtreeIndexG ON test.movie10k(genres);
CREATE INDEX movieBtreeIndexG ON test.movie1k(genres);
CREATE INDEX movieBtreeIndexG ON test.movie100(genres);
```

6.8.3 Dotazy s indexem

Dotazy s indexem opět používají kombinaci klíčových slov `WITH` a `INDEX`, který v tomto případě bude mít za parametr název indexu, který bude použit. Následujících pět příkazů je tedy pro testování maximální selektivity s indexem.

```
SELECT * FROM test.movie1kk WITH (INDEX(movieBtreeIndexOT)) WHERE
originalTitle = 'The Island';
SELECT * FROM test.movie100k WITH (INDEX(movieBtreeIndexOT)) WHERE
originalTitle = 'She Devil';
SELECT * FROM test.movie10k WITH (INDEX(movieBtreeIndexOT)) WHERE
originalTitle = 'The Butterfly';
SELECT * FROM test.movie1k WITH (INDEX(movieBtreeIndexOT)) WHERE
originalTitle = 'Esmeralda';
SELECT * FROM test.movie100 WITH (INDEX(movieBtreeIndexOT)) WHERE
originalTitle = 'Carga de rurales';
```

Dalších pět pro žádnou až minimální selektivitu.

```
SELECT * FROM test.movie1kk WITH (INDEX(movieBtreeIndexTT)) WHERE
titleType = 'short';
```

```
SELECT * FROM test.movie100k WITH (INDEX(movieBtreeIndexTT)) WHERE
titleType = 'short';
SELECT * FROM test.movie10k WITH (INDEX(movieBtreeIndexTT)) WHERE
titleType = 'short';
SELECT * FROM test.movie1k WITH (INDEX(movieBtreeIndexTT)) WHERE titleType
= 'short';
SELECT * FROM test.movie100 WITH (INDEX(movieBtreeIndexTT)) WHERE
titleType = 'short';
```

A posledních pět pro střední selektivitu.

```
SELECT * FROM test.movie1kk WITH (INDEX(movieBtreeIndexG)) WHERE genres =
'Comedy,Short';
SELECT * FROM test.movie100k WITH (INDEX(movieBtreeIndexG)) WHERE genres =
'Comedy,Short';
SELECT * FROM test.movie10k WITH (INDEX(movieBtreeIndexG)) WHERE genres =
'Comedy,Short';
SELECT * FROM test.movie1k WITH (INDEX(movieBtreeIndexG)) WHERE genres =
'Comedy,Short';
SELECT * FROM test.movie100 WITH (INDEX(movieBtreeIndexG)) WHERE genres =
'Comedy,Short';
```

6.9 MySQL

Odlišností u MySQL je to, že ve výchozím nastavení nepoužívá žádnou vyrovnávací paměť pro dotazy. Ověřit to lze následujícím příkazem. Nebude tedy potřeba před každým dotazem mazat vyrovnávací paměť.

```
SHOW VARIABLES LIKE 'have_query_cache';
```

6.9.1 Dotazy bez indexu

Vynucení použití indexu se v MySQL liší stejně jako u Oracle a MS SQL Serveru. V MySQL se používá kombinace klíčových slov `USE` a `INDEX`. Při ponechání prázdného parametru je vynucený full table scan. Následujících pět příkazů tedy opět poslouží k otestování dotazu s maximální selektivitou.

```
SELECT * FROM movie1kk USE INDEX () WHERE originalTitle = 'The Island';
SELECT * FROM movie100k USE INDEX () WHERE originalTitle = 'She Devil';
SELECT * FROM movie10k USE INDEX () WHERE originalTitle = 'The Butterfly';
SELECT * FROM movie1k USE INDEX () WHERE originalTitle = 'Esmeralda';
SELECT * FROM movie100 USE INDEX () WHERE originalTitle = 'Carga de
rurales';
```

Dalších pět pro testování žádné až minimální selektivity.

```
SELECT * FROM movie1kk USE INDEX () WHERE titleType = 'short';
SELECT * FROM movie100k USE INDEX () WHERE titleType = 'short';
SELECT * FROM movie10k USE INDEX () WHERE titleType = 'short';
SELECT * FROM movie1k USE INDEX () WHERE titleType = 'short';
SELECT * FROM movie100 USE INDEX () WHERE titleType = 'short';
```

A posledních pět pro otestování střední selektivity.

```
SELECT * FROM movie1kk USE INDEX () WHERE genres = 'Comedy,Short';
SELECT * FROM movie100k USE INDEX () WHERE genres = 'Comedy,Short';
SELECT * FROM movie10k USE INDEX () WHERE genres = 'Comedy,Short';
```

```
SELECT * FROM movie1k USE INDEX () WHERE genres = 'Comedy,Short';
SELECT * FROM movie100 USE INDEX () WHERE genres = 'Comedy,Short';
```

6.9.2 Vytvoření indexů

Vytváření indexů je stejné jako u databázových systémů Oracle a MS SQL Server.

```
CREATE INDEX movieBtreeIndexOT ON movie1kk(originalTitle);
CREATE INDEX movieBtreeIndexOT ON movie100k(originalTitle);
CREATE INDEX movieBtreeIndexOT ON movie10k(originalTitle);
CREATE INDEX movieBtreeIndexOT ON movie1k(originalTitle);
CREATE INDEX movieBtreeIndexOT ON movie100(originalTitle);

CREATE INDEX movieBtreeIndexTT ON movie1kk(titleType);
CREATE INDEX movieBtreeIndexTT ON movie100k(titleType);
CREATE INDEX movieBtreeIndexTT ON movie10k(titleType);
CREATE INDEX movieBtreeIndexTT ON movie1k(titleType);
CREATE INDEX movieBtreeIndexTT ON movie100(titleType);

CREATE INDEX movieBtreeIndexG ON movie1kk(genres);
CREATE INDEX movieBtreeIndexG ON movie100k(genres);
CREATE INDEX movieBtreeIndexG ON movie10k(genres);
CREATE INDEX movieBtreeIndexG ON movie1k(genres);
CREATE INDEX movieBtreeIndexG ON movie100(genres);
```

6.9.3 Dotazy s indexem

Vynucení použití zvoleného indexu se řídí stejnými pravidly jako při vynucení full table scan s rozdílem, že místo prázdného parametru bude zadán název indexu, který má být použitý. Následujících patnáct příkazů slouží k otestování různých selektivit, stejně jako předchozích patnáct bez indexu.

```
SELECT * FROM movie1kk USE INDEX (movieBtreeIndexOT) WHERE originalTitle =
'The Island';
SELECT * FROM movie100k USE INDEX (movieBtreeIndexOT) WHERE originalTitle =
'She Devil';
SELECT * FROM movie10k USE INDEX (movieBtreeIndexOT) WHERE originalTitle =
'The Butterfly';
SELECT * FROM movie1k USE INDEX (movieBtreeIndexOT) WHERE originalTitle =
'Esmeralda';
SELECT * FROM movie100 USE INDEX (movieBtreeIndexOT) WHERE originalTitle =
'Carga de rurales';

SELECT * FROM movie1kk USE INDEX (movieBtreeIndexTT) WHERE titleType =
'short';
SELECT * FROM movie100k USE INDEX (movieBtreeIndexTT) WHERE titleType =
'short';
SELECT * FROM movie10k USE INDEX (movieBtreeIndexTT) WHERE titleType =
'short';
SELECT * FROM movie1k USE INDEX (movieBtreeIndexTT) WHERE titleType =
'short';
SELECT * FROM movie100 USE INDEX (movieBtreeIndexTT) WHERE titleType =
'short';

SELECT * FROM movie1kk USE INDEX (movieBtreeIndexG) WHERE genres =
'Comedy,Short';
SELECT * FROM movie100k USE INDEX (movieBtreeIndexG) WHERE genres =
'Comedy,Short';
```

```
SELECT * FROM movie10k USE INDEX (movieBtreeIndexG) WHERE genres =
'Comedy,Short';
SELECT * FROM movie1k USE INDEX (movieBtreeIndexG) WHERE genres =
'Comedy,Short';
SELECT * FROM movie100 USE INDEX (movieBtreeIndexG) WHERE genres =
'Comedy,Short';
```

7 POROVNÁNÍ VÝSLEDKŮ

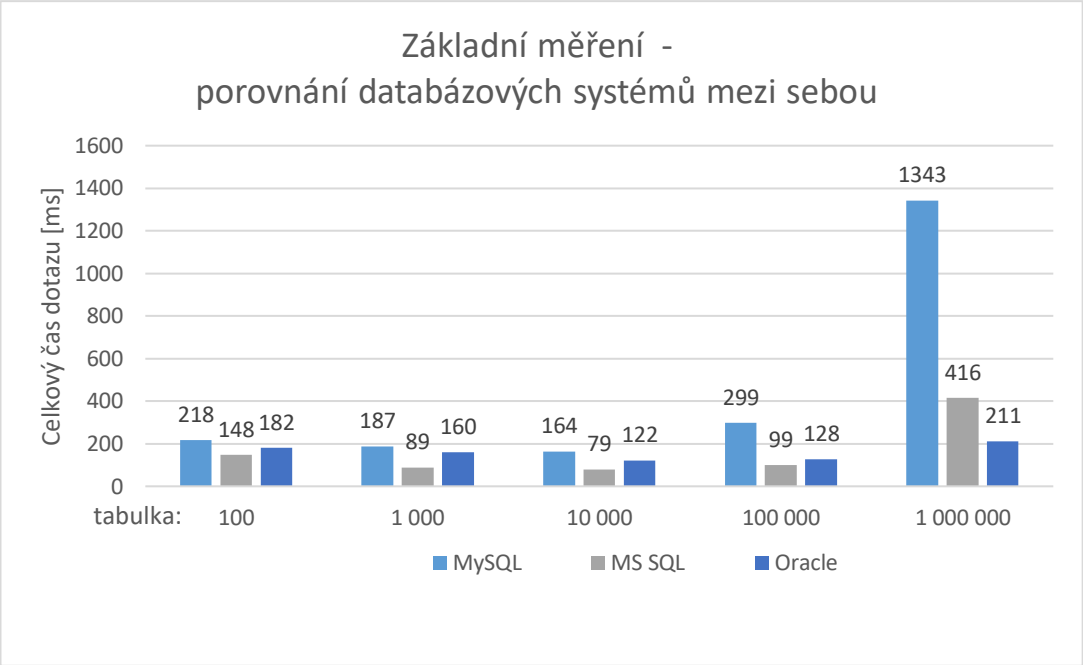
Všechny dotazy byly spuštěné v aplikaci DataGrip nad instancemi jednotlivých databázových systémů. Byl dodržen scénář testování. V průběhu měření běžel v hostitelském OS pouze DataGrip a instance databáze v Dockeru, na které probíhalo měření. V samotném průběhu měření se neobjevily žádné problémy.

Nejprve bude porovnání výsledků základního měření a poté bude porovnání výsledků jednotlivých databázových systémů, kde jsou porovnávány dotazy bez použití indexu a s ním. Zároveň bude možné sledovat, jak výkon ovlivňuje selektivita a jaký vliv na ni má index. Po jednotlivých databázových systémech bude následovat srovnání výsledků relačních databázových systémů mezi sebou. Ze závěrečného srovnání půjde vyvodit, který databázový systém bude vhodnější pro konkrétní příklad užití. Je však nutné brát v potaz, že testování neodpovídá přesně podmínkám databázového systému v reálném provozu. Není zde například bráno v potaz zatížení více dotazy najednou, vliv vyrovnávací paměti, podrobnější konfigurace databáze a vliv přidělených hardwarových prostředků.

V databázi Oracle bude navíc testován bitmapový index, který by měl dosahovat lepších výkonů u dotazů s malou selektivitou. MySQL a MS SQL Server bitmapový index nenabízí.

7.1 Základní měření

Základní měření porovnává jednotlivé databázové systémy mezi sebou po instalaci ve výchozím nastavení. Jedná se o základní full table scan, tedy prohledávání celé tabulky. Po celou dobu testu bylo pracováno s jednou tabulkou, do které byly postupně přidávány další záznamy. Výsledky mohly být ovlivněné ukládáním dat do vyrovnávací paměti jednotlivých databázových systémů. Z výsledků je patrné, že u menšího počtu dat je znatelně výkonnější databáze od společnosti Microsoft, kterou ale výrazně předčí na milionu dat databáze od společnosti Oracle. S přibývajícím počtem dat se začala velmi výrazně propadat databáze MySQL, která byla na milionu dat několikanásobně pomalejší oproti konkurenci, což je zapříčiněno faktem, že na Oracle a Microsoftu dochází k optimalizaci dotazů, která je minimálně závislá na objemu dat a u MySQL je tomu naopak.

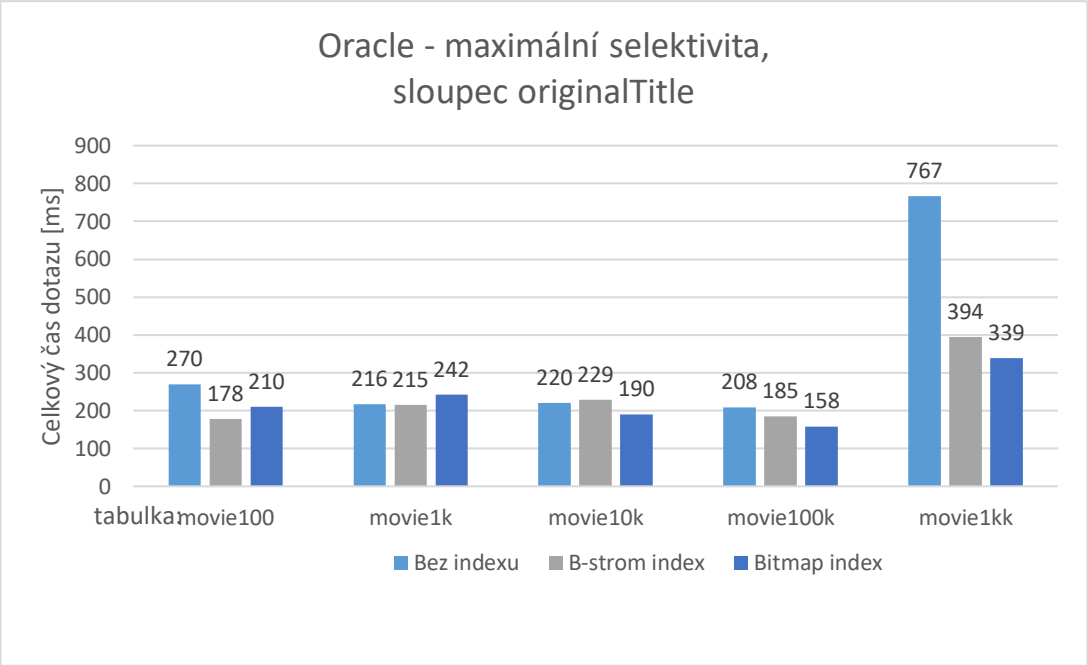


Graf 1 – Základní měření – porovnání databázových systémů mezi sebou

7.2 Oracle Database

7.2.1 Maximální selektivita

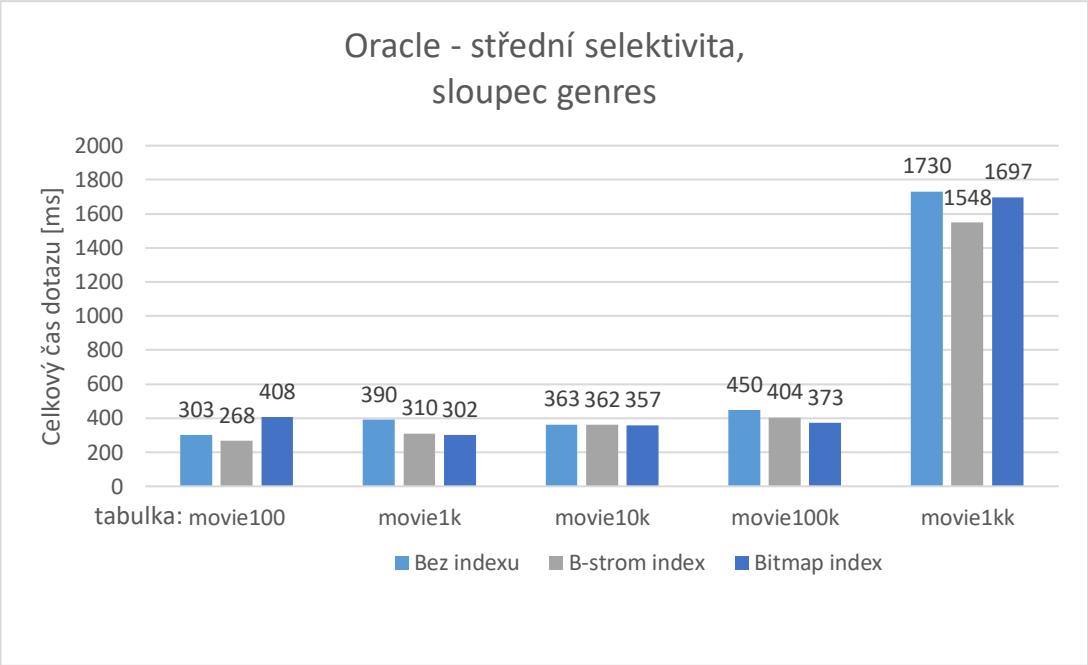
První byl otestován dotaz s maximální selektivitou, kdy se ve všech pěti tabulkách hledal jeden konkrétní záznam. V následujícím grafu je vidět, že větší rozdíl je patrný až u největší tabulky. Rychlost dotazu používající index byla skoro dvojnásobná oproti dotazu bez indexu. Nicméně v tabulkách o velikosti 100 000 dat a méně je vidět, že byly časy velmi vyrovnané a použití indexu nepřidalo žádný velký výkonnostní benefit. Jeho použití může být tedy spíše přítěží v podobě zabraného místa na disku navíc. Příčinou mohlo být to, že samotné načtení souboru s indexem je zátěží navíc, která se poté projeví právě u „rychlejšího“ dotazu, kde není zapotřebí prohledávat velké množství dat. Malým překvapením byl výkon bitmapového dotazu, který mírně předčil výkon indexu založeného na B-stromě, který by měl v případě maximální selektivity být výkonnější.



Graf 2 – Oracle – maximální selektivita, sloupec originalTitle (vlastní zpracování)

7.2.2 Střední selektivita

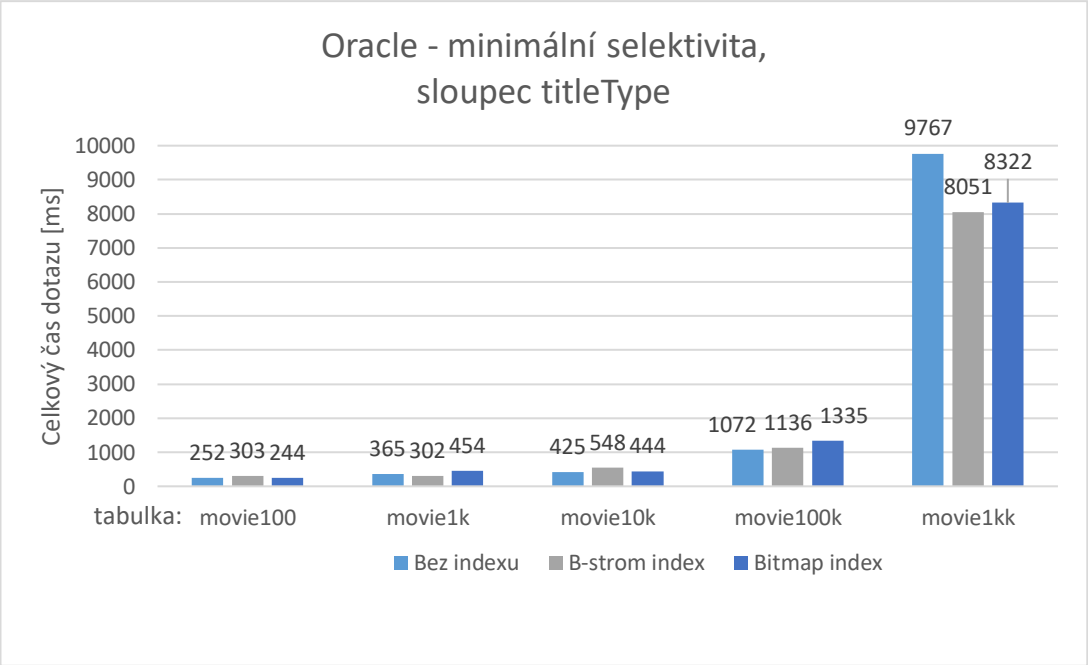
U střední selektivity byly výsledky velmi vyrovnané a nepotvrdilo se očekávání, že bitmapový index bude mít lepší výsledky než index založený na B-stromě. Zároveň se ukázalo, že ani jeden index výrazně výkonnostně nepředčil full table scan. Je zde tedy možné pozorovat, že při hledání větší množiny výsledků ztrácel index založený na B-stromě svoji rychlost a full table scan dosahoval srovnatelných výsledků. Bitmap index držel srovnatelné výsledky s indexem založeným na B-stromě, ale vzhledem k nevýhodám tohoto typu indexu jako je zamčení většího množství záznamů a celkově velké režii při DML operacích se ukázal spíše jako nevhodný pro střední selektivitu. Také je nutné zmínit, že při hledání větší množiny dat byl dotaz výrazně pomalejší oproti dotazu s vysokou selektivitou.



Graf 3 – Oracle – střední selektivita, sloupec genres (vlastní zpracování)

7.2.3 Minimální selektivita

Zajímavé výsledky se objevily také u minimální selektivity, a to opět díky bitmapovému indexu. Ten měl v tomto testu výkonově předčit index založený na B-stromě i full table scan. Z výsledků vyplývá, že bitmapový index je pravděpodobně hlavně dominantní u hvězdicových schémat tabulek. Testování na hvězdicovém schématu tabulek však není předmětem této bakalářské práce. I když je bitmapový index společností Oracle označen jako lepší pro dotazy s nízkou selektivitou, než index založený na B-stromě, tak měření v této bakalářské práci to nepotvrdila. Index založený na B-stromě byl u tabulky s nejvyšším počtem dat mírně rychlejší než full table scan. Dáno to bylo pravděpodobně tím, že jsou data uložena v indexu za sebou a při tak velkém množství dat je skenování celé tabulky pomalejší. Při menším počtu dat jsou, ale výsledky velmi vyrovnané a použití indexů se nejevilo jako výkonnostní benefit. Zároveň je vidět obrovský výkonnostní rozdíl mezi dotazem na sloupec s minimální selektivitou oproti sloupci s maximální. Také si lze všimnout, že hledání velké množiny dat je násobně pomalejší než hledání jednoho konkrétního záznamu.

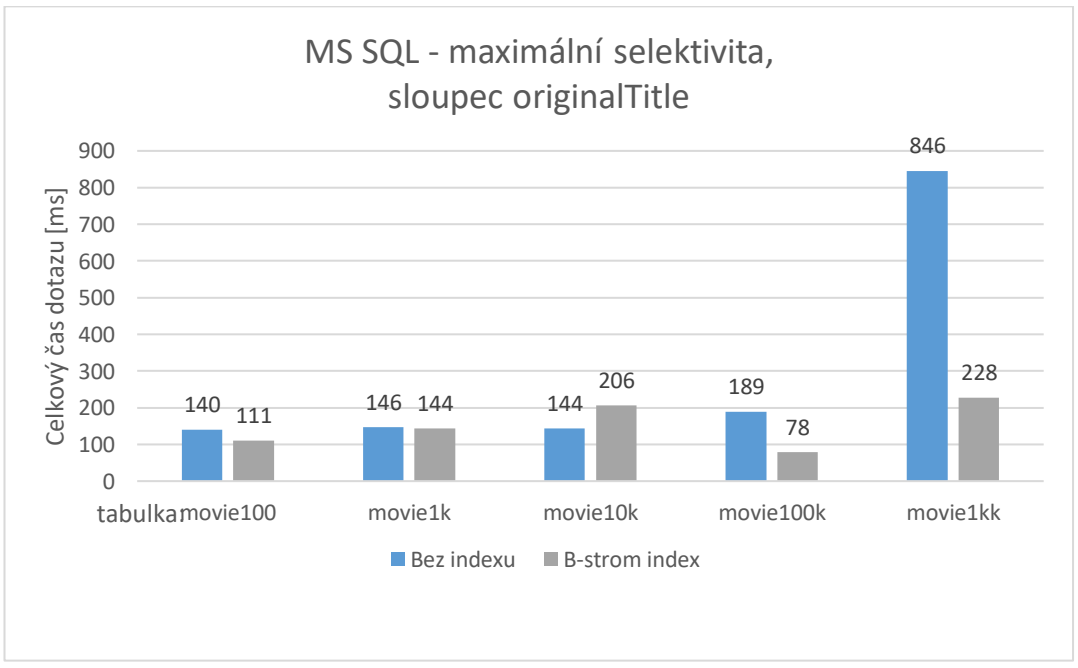


Graf 4 – Oracle – minimální selektivita, sloupec titleType (vlastní zpracování)

7.3 Microsoft SQL Server

7.3.1 Maximální selektivita

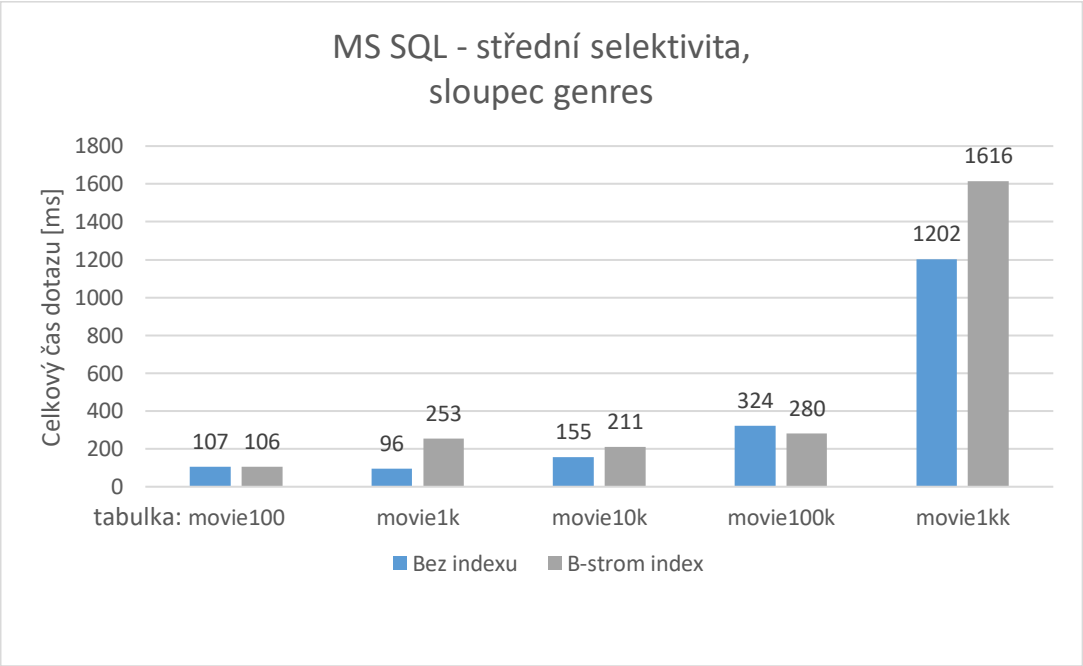
U SQL Serveru se při hledání jednoho záznamu a množstvím dat větším než 100 000 jeví použití indexu jako výkonnostní benefit. Rychlost u největšího počtu dat byla téměř čtyřnásobně vyšší než u dotazu bez indexu. U menších tabulek byly výsledky velmi vyrovnané a vytvoření indexu není příliš výhodné.



Graf 5 – MS SQL – maximální selektivita, sloupec originalTitle (vlastní zpracování)

7.3.2 Střední selektivita

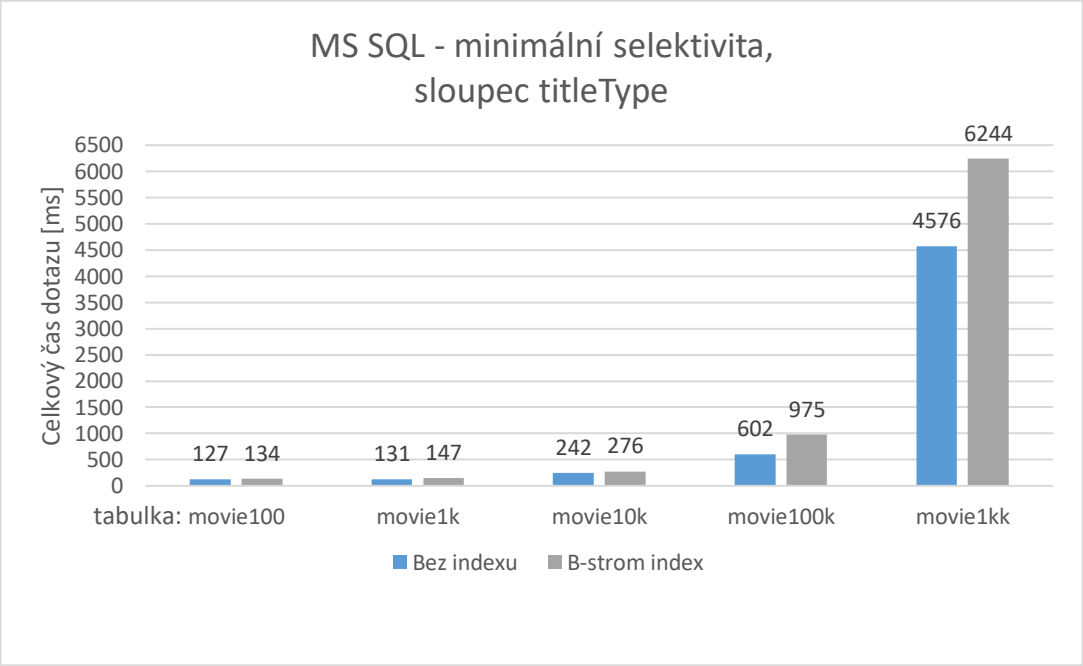
Dotaz na sloupec se střední selektivitou přinesl vyrovnané výsledky a u největší tabulky dokonce větší časovou ztrátu dotazu s indexem. Ztráta nebyla velmi výrazná, ale potvrdila to, že index založený na B-stromě je určený primárně pro zrychlení dotazů na sloupcích s vysokou selektivitou. V malých rozdílech poté mohl hrát roli rychlejší full table scan daného databázového systému, načítání souboru indexu, konkrétní dotaz a množina dat, která byla hledána. Také se zde potvrdilo, že s úměrně klesající selektivitou se prodlužuje celková doba dotazu.



Graf 6 – MS SQL – střední selektivita, sloupec genres (vlastní zpracování)

7.3.3 Minimální selektivita

Minimální selektivita potvrdila výsledky měření u střední selektivity. Dotazování s indexem bylo pomalejší a tento rozdíl se zvětšoval s přibývajícím množstvím dat v tabulce. Použití indexu má tak své opodstatnění hlavně u sloupců s maximální selektivitou.

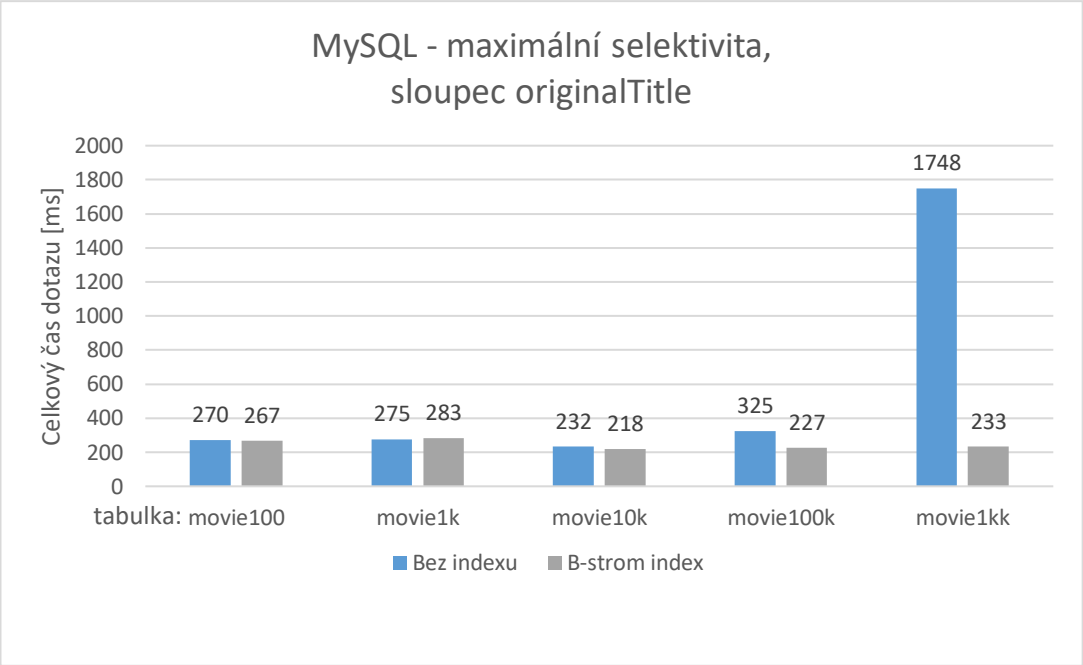


Graf 7 – MS SQL – minimální selektivita, sloupec titleType (vlastní zpracování)

7.4 MySQL

7.4.1 Maximální selektivita

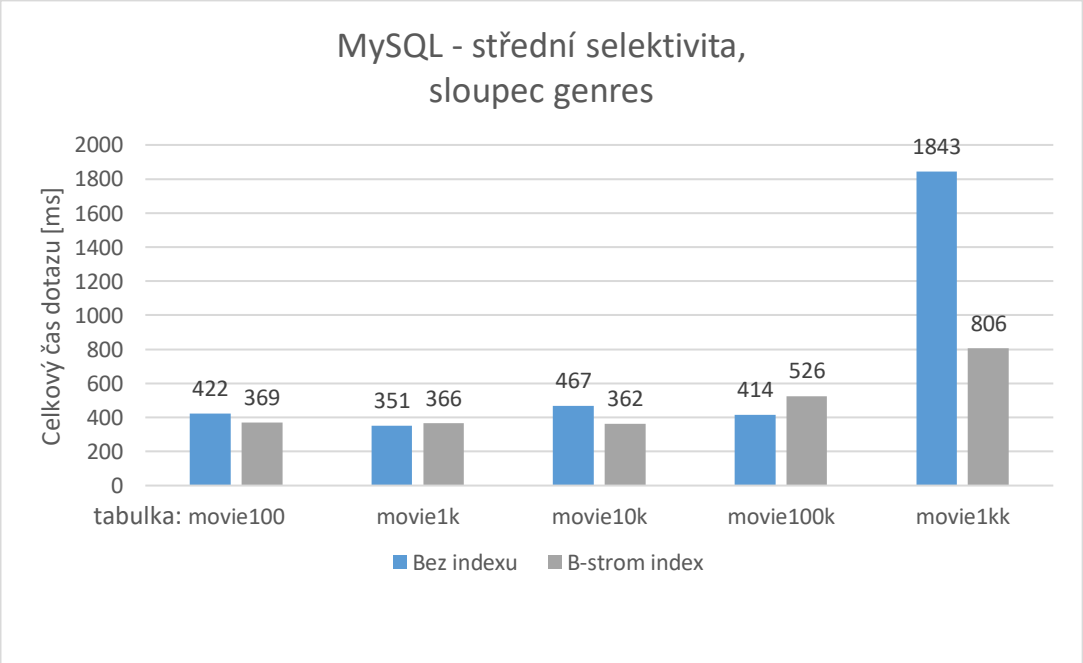
MySQL má celkově pomalejší full table scan. Je to dáno mimo jiné architekturou samotného databázového systému, který byl navržen spíš pro webové aplikace s menším množstvím dat. Jak je vidět z následujícího grafu při dotazu na největší tabulku začal full table scan násobně zpomalovat. Po použití indexu ovšem nemělo MySQL problém ani s velkým množstvím dat. U menšího počtu dat nebyl rozdíl tak patrný a výsledky byly velmi vyrovnané.



Graf 8 – MySQL – maximální selektivita, sloupec originalTitle (vlastní zpracování)

7.4.2 Střední selektivita

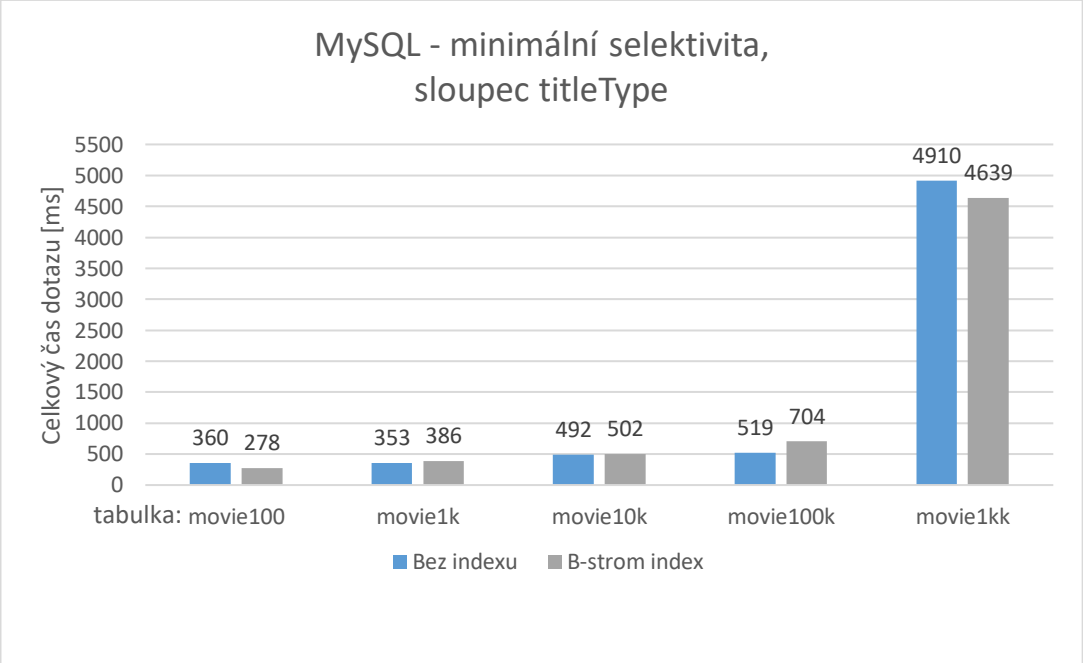
Výsledky u střední selektivity byly při prohledávání celé tabulky mírně pomalejší oproti sloupci s vysokou selektivitou a nejsou nijak překvapující. Střední selektivita měla větší dopad na indexované dotazy, které zpomalily výrazně, ale v případě největší tabulky byly pořád víc jak dvojnásobně rychlejší oproti použití bez indexu.



Graf 9 – MySQL – střední selektivita, sloupec genres (vlastní zpracování)

7.4.3 Minimální selektivita

Výsledky u sloupce s minimální selektivitou smazaly rozdíly mezi použitím dotazu s indexem a bez něj. Výsledky byly velmi vyrovnané s minimálními rozdíly a použití indexu, tak bylo spíše na škodu z důvodu místa na disku, který zabral. Dotazy na velkou množinu dat byly opět výrazně pomalejší oproti dotazům na sloupce se střední a maximální selektivitou.



Graf 10 – MySQL – minimální selektivita, sloupec titleType (vlastní zpracování)

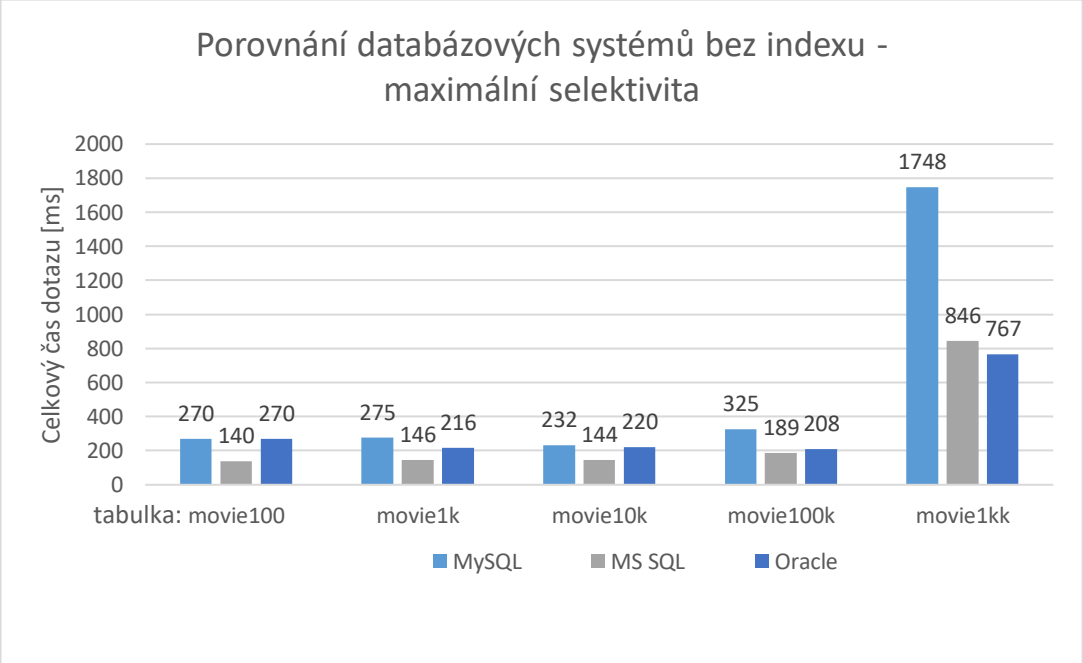
7.5 Porovnání

Porovnání mezi jednotlivými databázemi je důležité. Může přispět u rozhodování, který databázový systém zvolit. Výsledky dotazů bez indexů nebudou pro použití v praxi tak podstatné, protože všechny tři databázové systémy používají optimalizátor dotazů. Ten sám vytvoří index na často dotazované sloupce, pokud algoritmus vyhodnotí, že to bude výkonnostně výhodné. Ovšem malá šance, že chybovat může i algoritmus zde je, a také je třeba brát v potaz to, že pro nějaký velmi specifický příklad užití nemusí algoritmus optimalizátoru navrhnout optimalizace správně.

7.5.1 Bez indexu

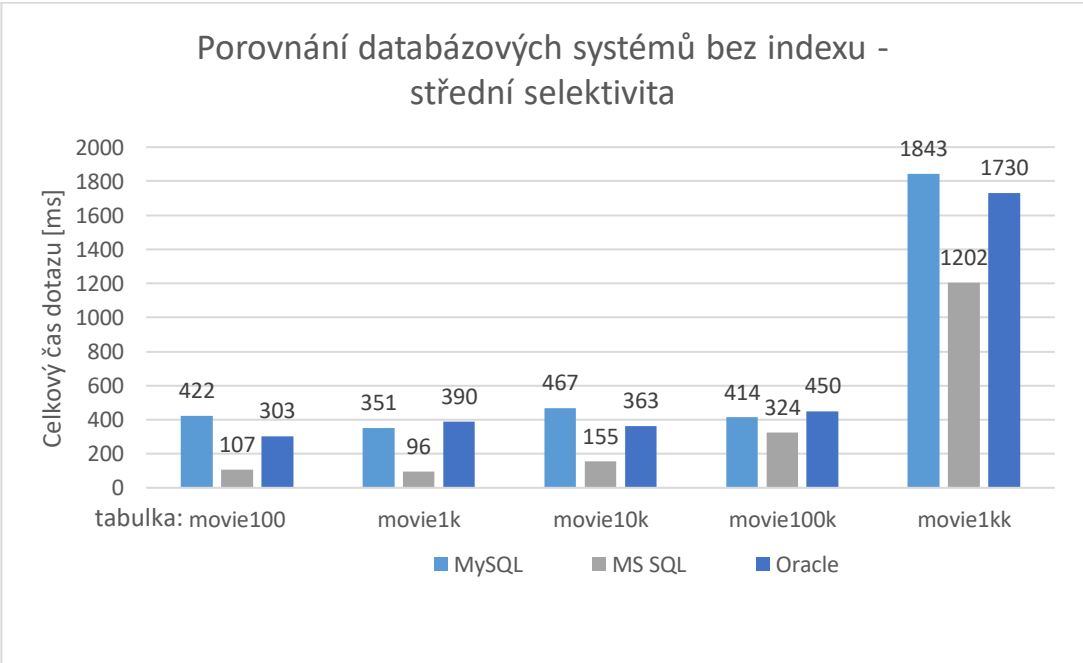
Porovnání dotazů bez indexu na maximální selektivitu u jednotlivých databázových systémů ukázalo, že MS SQL Server dosahuje oproti svým dvěma konkurentům až dvojnásobně lepších rychlostí u malých tabulek. Naproti tomu u největší tabulky se výrazně projevil více než

dvojnásobně pomalejší databázový systém MySQL. Jeho konkurenti si byli vyrovnaní, zatímco MySQL zaostával.



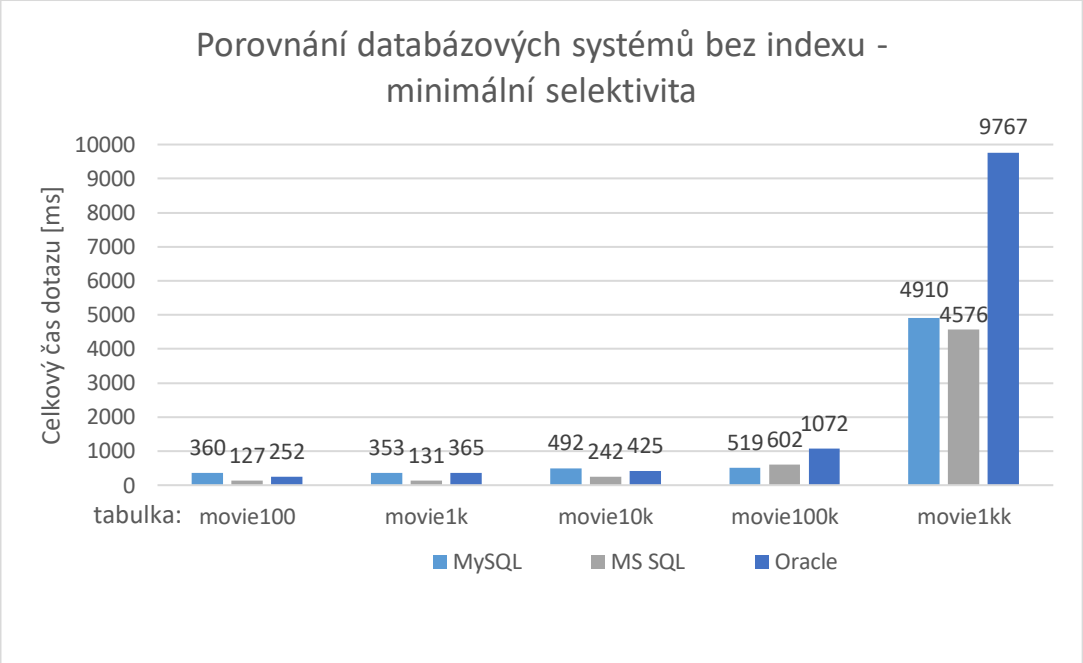
Graf 11 – Porovnání RDBMS bez indexu – maximální selektivita (vlastní zpracování)

U dotazů bez indexu na sloupec se střední selektivitou odskočil svým konkurentům databázový systém od Microsoftu, a to na všech tabulkách. Zatímco MySQL a Oracle měly velmi vyrovnané výsledky, MS SQL Server odskočil, a to velmi výrazně zejména na malém počtu dat, kde byly rozdíly tří až čtyřnásobné. Na milionu dat byl rozdíl v řádu desítek procent.



Graf 12 – Porovnání RDBMS bez indexu – střední selektivita (vlastní zpracování)

Na dalším grafu, kde byl testován dotaz na sloupec s nízkou selektivitou, zaujme hlavně pomalé dotazování databázového systému Oracle. A to zejména při velkém počtu dat, kde byl celkový čas dotazu až dvojnásobný. Naproti tomu za nejlepší v této kategorii by se dal považovat MS SQL Server, který hlavně při menších tabulkách dosahuje velkých výkonnostních rozdílů.

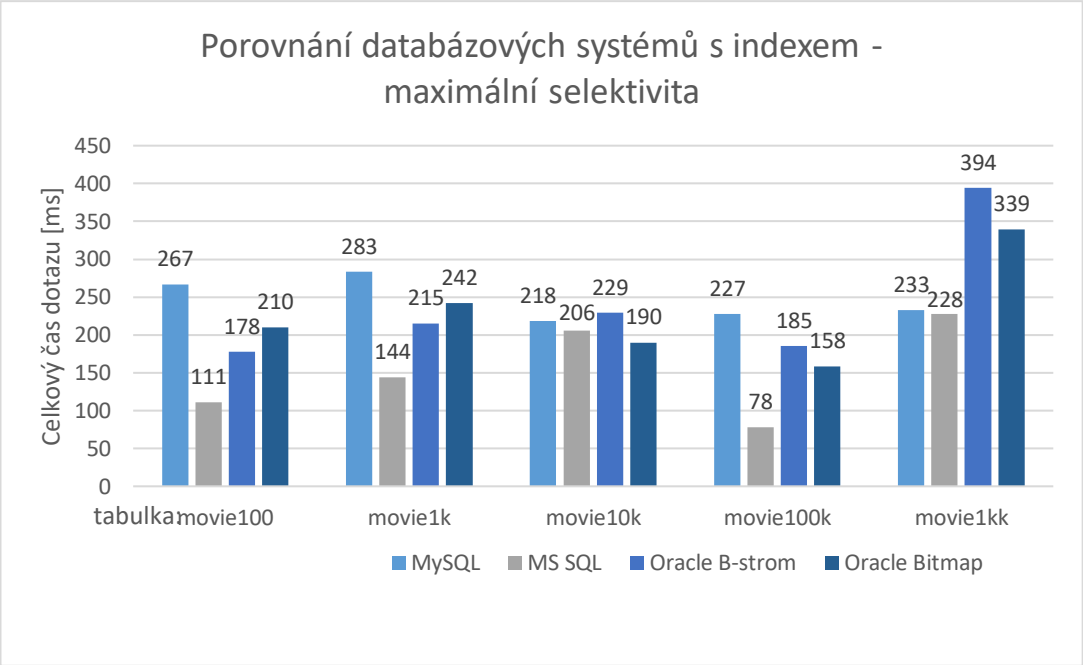


Graf 13 – Porovnání RDBMS bez indexu – minimální selektivita (vlastní zpracování)

7.5.2 S indexem

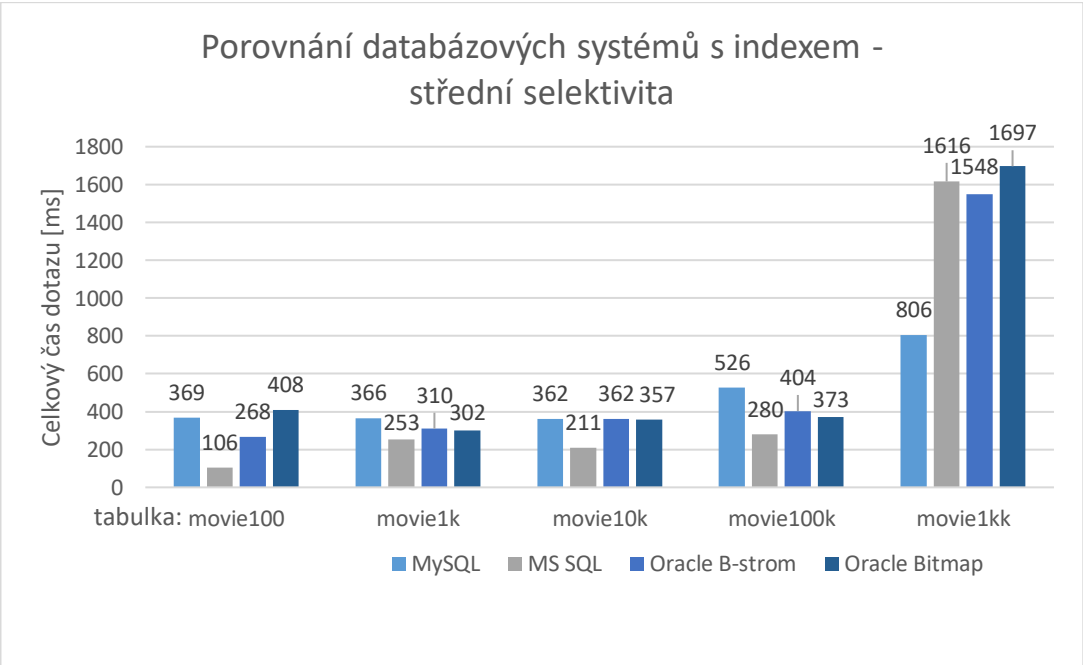
Srovnání jednotlivých databázových systémů na dotazech s indexem je pravděpodobně jedním z nejdůležitějších bodů této práce. Často dotazované sloupce budou používat indexy, ať už ručně vytvořené nebo vytvořené optimalizátorem. Stále je však nutné brát ohled na to, že toto testování nereflektuje databázový systém nasazený přímo v praxi, kde bude například zatížen více uživateli najednou. Může však posloužit jako pomůcka a dát určitý náhled do problematiky indexování.

Další graf tedy srovnává dotazy na konkrétní jeden záznam. Místy těsné výsledky, z kterých vychází nejlépe MS SQL Server. Ten zejména opět na malých počtech dat dosáhnul až dvojnásobné rychlosti oproti systému MySQL, který naopak na malých tabulkách ztrácel. Na velkém počtu dat byly výsledky MySQL a MS SQL Serveru velmi vyrovnané, ale databázový systém Oracle byl téměř dvojnásobně pomalejší.



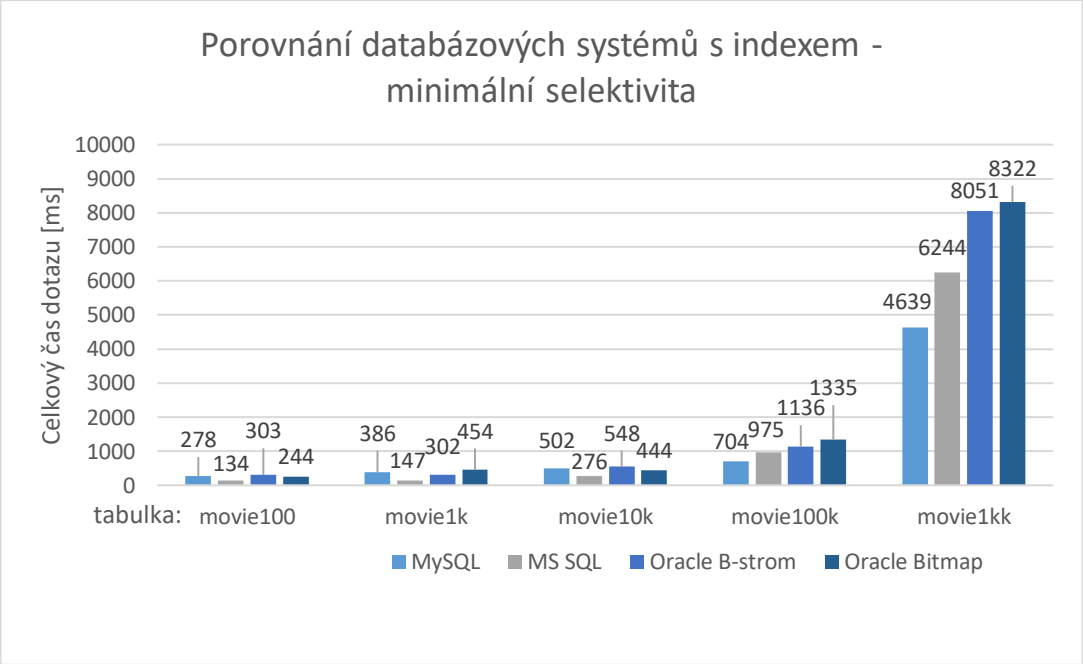
Graf 14 – Porovnání RDBMS s indexem – maximální selektivita (vlastní zpracování)

Při pohledu na graf zabývající se střední selektivitou opět zaujme MS SQL Server, který odskočil při dotazování na tabulkách s malým počtem dat. Jinak byly výsledky velmi vyrovnané až na databázový systém MySQL. Ten u dotazu na tabulce s milionem dat byl dvojnásobně rychlejší než jeho konkurenti, kteří byli mezi sebou téměř vyrovnaní.



Graf 15 – Porovnání RDBMS s indexem – střední selektivita (vlastní zpracování)

Výsledky posledního grafu zobrazují výsledky dotazů s indexem na sloupec s minimální selektivitou, tedy minimálním počtem unikátních hodnot ve sloupci na daný počet dat. Na malém počtu dat byl opět výrazný MS SQL Server, který by se tak dal v této kategorii označit celkově za nejrychlejší databázový systém s použitím indexů. Naproti tomu na počtu dat 100 000 a více byl nejefektivnější databázový systém MySQL, který u maximální selektivity držel krok s ostatními databázovými systémy a na střední a minimální selektivitě měl jednoznačně nejlepší čas. Na malém počtu dat, by se tak daly označit za nejefektivnější indexy MS SQL Serveru a na větší ty od MySQL naopak oba typy indexů od společnosti Oracle v tomto měření spíše selhaly.



Graf 16 – Porovnání RDBMS s indexem – minimální selektivita (vlastní zpracování)

ZÁVĚR

První část práce podrobně seznámila čtenáře s problematikou relačních databází a testovanými relačními databázemi, tedy Microsoft SQL Server, Oracle a MySQL. Pátá kapitola „Index“ zmiňuje možnosti využití indexů u testovaných relačních databází. Cílem této práce bylo porovnání rychlosti `SELECT` dotazů různých databázových systémů bez indexu a s indexem. Poté srovnat výsledky vybraných databázových systémů mezi sebou, a určit tak nejvhodnější relační databázový systém. Je nutné zdůraznit, že měření plně nereflektují reálné zatížení databázového systému v praxi.

Z výsledků je patrné, že indexování má značný a pozitivní vliv na výkon dotazů, a to především na dotazy s vysokou selektivitou. U dotazů s nízkou selektivitou je výkonnostní rozdíl poměrně malý a v případech s minimální selektivitou může být až negativní. Jeho použití v těchto případech je tedy spíše kontraproduktivní. Z výsledků také vyplývá, že s klesající selektivitou a vyšším počtem dotazovaných dat se celková doba provedení dotazu značně navyšuje. Bitmapový index od Oracle nemá žádné výkonnostní benefity pro dotazy na sloupce s nízkou selektivitou a celkově oba typy indexů od Oracle zůstaly spíše za očekáváními.

Z porovnání dotazů jednotlivých databázových systémů mezi sebou vyšlo najevo, že Microsoft SQL Server je s indexy znatelně výkonnější na tabulkách s menším počtem dat. Naproti tomu indexy databáze MySQL si velmi dobře poradily s dotazy na sloupce s menší selektivitou na tabulkách s velkým počtem dat a jeví se tak pro tento případ užití jako nejlepší. V této časti byl oproti konkurenci naopak velmi pomalý databázový systém Oracle. Je nutno také zmínit bezplatnou licenci MySQL, kterou konkurence v nabídce nemá.

V dnešní době všechny tři databázové systémy mají svůj optimalizátor dotazů. Ten většinou monitoruje dotazy dané instance databáze a navrhuje si a implementuje optimalizace sám včetně indexů. Ačkoliv je šance, že by optimalizátor vyvíjený množstvím vývojářů nevytvořil nejefektivnější index malá, existují speciální případy využití, kdy je vhodné nebo i nutné nastavit index manuálně.

V budoucnu by toto měření mohlo být rozšířeno o měření pod zátěží, tedy s více vlákny naráz. Tento test by se pak více blížil běhu databázového systému v praxi. Další možností by bylo otestování indexů na cizích klíčích.

POUŽITÁ LITERATURA

[1] KROENKE, David a David J. AUER. *Databáze*. Přeložil Jakub GONER. Brno: Computer Press, 2015. ISBN 978-80-251-4352-0.

[2] LAURENČÍK, Marek. *SQL: podrobný průvodce uživatele*. Praha: Grada Publishing, 2018. Průvodce (Grada). ISBN 978-80-271-0774-2.

[3] POKORNÝ, Jaroslav a Michal VALENTA. *Databázové systémy*. Praha: České vysoké učení technické v Praze, 2020. ISBN 978-80-01-05212-9.

[4] STRATE, Jason a Grant FRITCHEY. *Expert performance indexing in SQL server. Second edition*. New York: Apress, [2015].

[5] SQL Constraints. *W3Schools* [online]. © 2020 [cit. 2020-08-04]. Dostupné z: https://www.w3schools.com/sql/sql_constraints.asp

[6] MySQL What is DDL, DML and DCL? *W3Schools* [online]. [b. r.] [cit. 2020-08-04]. Dostupné z: https://www.w3schools.in/mysql/ddl-dml-dcl/

[7] ASHDOWN, Lance a Tom KYTE. Introduction to Oracle Database. *Oracle Help Center* [online]. 05-2015 [cit. 2020-08-04]. Dostupné z: https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT001

[8] Database New Features Guide: Oracle Database Release 19c New Features. *Oracle Help Center* [online]. © 2019 [cit. 2020-08-04]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/19/newft/new-features.html#GUID-5490FE65-562B-49DC-9246-661592C630F9

[9] Release Schedule of Current Database Releases. *My Oracle Support* [online]. Aktualizace 03-08-2020 [cit. 2020-08-04]. Dostupné z: https://support.oracle.com/knowledge/Oracle%20Cloud/742060_1.html

[10] SLEDGE, Orryn a Mark SPENIK. The Evolution of SQL Server. *Informit* [online]. 06-02-2002 [cit. 2020-08-04]. Dostupné z: https://www.informit.com/articles/article.aspx?p=30167

[11] SQL Server Data Types Tutorial. *EssentialSQL* [online]. © 2020 [cit. 2020-08-04]. Dostupné z: https://www.essentialsql.com/sql-tutorial/sql-server-data-types/

[12] Oracle Vs. SQL Server: Key Differences. *Guru99* [online]. Aktualizace 13-07-2020 [cit. 2020-08-04]. Dostupné z: https://www.guru99.com/oracle-vs-sql-server.html

[13] ROUSE, Margaret. T-SQL (Transact-SQL). *TechTarget SearchSQLServer* [online]. 09-2019 [cit. 2020-08-04]. Dostupné z: https://searchsqlserver.techtarget.com/definition/T-SQL

[14] STROHM, Richard. Oracle Data Types. *Oracle Help Center* [online]. 01-2011 [cit. 2020-08-12]. Dostupné z: https://docs.oracle.com/cd/B28359_01/server.111/b28318/datatype.htm#CNCPT012

[15] Database design basics. *Microsoft Support* [online]. © 2021 [cit. 2021-03-22]. Dostupné z: https://support.microsoft.com/en-gb/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5

[16] GRAVELLE, Robert. The NULL Value and its Purpose in Relational Database Systems. *Navicat: Navicat Blog* [online]. 03-03-2020 [cit. 2021-03-23]. Dostupné z: https://www.navicat.com/en/company/aboutus/blog/1312-the-null-value-and-its-purpose-in-relational-database-systems

[17] LACKO, Luboslav. *Oracle: Správa, programování a použití databázového systému*. 2. dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1490-2.

[18] Database New Features Guide: Oracle Database Release 12g New Features. *Oracle Help Center* [online]. 10-2019 [cit. 2021-05-01]. Dostupné z: https://docs.oracle.com/database/121/NEWFT/chapter12102.htm#NEWFT003

[19] Oracle Database Technologies. *Oracle* [online]. © 2021 [cit. 2021-03-23]. Dostupné z: https://www.oracle.com/database/technologies/

[20] PACHEV, Alexander. *Understanding MySQL Internals*. Gravenstein Highway North, Sebastopol, USA: O'Reilly Media, 2007. ISBN 978-0-596-00957-1.

[21] RAY, Mike. Ntext, text, and image. *Microsoft Docs* [online]. 22-07-2017 [cit. 2021-03-24]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/data-types/ntext-text-and-image-transact-sql?view=sql-server-ver15

[22] SINGH, HIMANSHI. A Guide to Data Types in MySQL for Data Science Beginners. *Analytics Vidhya* [online]. 10-11-2020 [cit. 2021-03-24]. Dostupné z: https://www.analyticsvidhya.com/blog/2020/11/guide-data-types-mysql-data-science-beginners

[23] ZEDNÍČEK, Jan. Optimalizace a zrychlení SQL dotazů – 10 tipů s ukázkou. *Biportal* [online]. 06-07-2018 [cit. 2021-04-04]. Dostupné z: https://biportal.cz/optimalizace-zrychleni-sql-dotazu-tipy/

[24] AKANKSHA, Rai. Query Optimization in Relational Algebra. *GeeksforGeeks* [online]. 20-08-2019 [cit. 2021-04-04]. Dostupné z: https://www.geeksforgeeks.org/query-optimization-in-relational-algebra/

[25] LORENTZ, Diana. Database SQL Reference: ROWID Pseudocolumn. *Oracle Help Center* [online]. 12-2005 [cit. 2021-04-05]. Dostupné z: https://docs.oracle.com/cd/B19306_01/server.102/b14200/pseudocolumns008.htm

[26] RAY, Mike. What's new in SQL Server 2019. *Microsoft Docs: SQL Docs* [online]. 11-04-2019 [cit. 2021-04-07]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-ver15?view=sql-server-ver15

[27] STOKES, Dave. MySQL 8 is coming. *Opensource.com* [online]. 28-02-2017 [cit. 2021-04-10]. Dostupné z: https://opensource.com/article/17/2/mysql-8-coming

[28] AXMARK, David a Michael WIDENIUS. What Is New in MySQL 8.0. *MySQL: MySQL 8.0 Reference Manual* [online]. Aktualizace 04-09-2021 [cit. 2021-04-10]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html

[29] NORTHCUTT, Corey. What's New In MySQL 8? *FUTURE HOSTING* [online]. 25-08-2018 [cit. 2021-04-17]. Dostupné z: https://www.futurehosting.com/blog/whats-new-in-mysql-8/

[30] ORACLE. Oracle Database 18c Technical Architecture. *Oracle Help Center* [online]. © 2019 [cit. 2021-04-17]. Dostupné z: https://www.oracle.com/webfolder/technetwork/tutorials/architecture-diagrams/18/technical-architecture/pdf/db-18c-architecture.pdf

[31] ASHDOWN, Lance, Donna KEESLING, Joe KYTE a Joe MCCORMACK. Oracle Database Architecture. *Oracle Help Center: Database Concepts* [online]. 01-2021 [cit. 2021-04-17]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/introduction-to-oracle-database.html#GUID-2B1BADE1-C36F-4555-9867-3B15B6CE858C

[32] MS SQL Server: SQL Server – Architecture. *Tutorialspoint* [online]. TutorialsPoint, © 2016 [cit. 2021-04-18]. Dostupné z: https://www.tutorialspoint.com/ms_sql_server/ms_sql_server_tutorial.pdf

[33] CHOUDHARY, Lalit. MySQL Architecture and Components. *Lalit's Blog* [online]. 03-11-2016 [cit. 2021-04-18]. Dostupné z: https://lalitvc.wordpress.com/2016/11/03/mysql-architecture-and-components/

[34] SRIVASTAVA, Shashwat. MySQL Logical Architecture: Internal working of mysql. *Shashwat Srivastava* [online]. 10-10-2019 [cit. 2021-04-18]. Dostupné z: https://shashwat-creator.medium.com/mysqls-logical-architecture-1-eaaa1f63ec2f

[35] SAXON, Chris. How to Create and Use Indexes in Oracle Database. *All Things SQL* [online]. Aktualizace 27-07-2020 [cit. 2021-04-21]. Dostupné z: https://blogs.oracle.com/sql/how-to-create-and-use-indexes-in-oracle-database

[36] RAY, Mike. Indexes: Available index types. *Microsoft Docs* [online]. 21-12-2016 [cit. 2021-04-23]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/indexes?view=sql-server-ver15

[37] VILEIKIS, Lukas. A Guide to MySQL Indexes. *Severalnines* [online]. 18-09-2020 [cit. 2021-04-23]. Dostupné z: https://severalnines.com/database-blog/guide-mysql-indexes

[38] BRADLEY, Steven. The Types of Indexes You Can Add To MySQL Tables. *Vanseo design* [online]. 10-02-2018 [cit. 2021-04-23]. Dostupné z: https://vanseodesign.com/web-design/the-types-of-indexes-you-can-add-to-mysql-tables

[39] CONOLLY, Thomas, Carolyn BEGG a Richard HOLOWCZAK. *Mistrovství – databáze: Profesionální průvodce tvorbou efektivních databází*. Brno: Computer Press, 2009. ISBN 978-80-251-2328-7.

[40] Database New Features Guide: Oracle Database Release 11g New Features. *Oracle Help Center* [online]. 12-2012 [cit. 2021-05-01]. Dostupné z: https://docs.oracle.com/cd/B28359_01/server.111/b28279/chapter1.htm#NEWFTCH1

[41] Database New Features Guide: Oracle Database Release 18c New Features. *Oracle Help Center* [online]. 02-2018 [cit. 2021-05-01]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/18/newft/new-features.html#GUID-04A4834D-848F-44D5-8C34-36237D40F194

[42] LACKO, Luboslav. *Jak vyzrát na Microsoft SQL Server 2008: Správa, konfigurace, programování*. Brno: Computer Press, 2009. ISBN 978-80-251-2101-6.

[43] RAY, Mike. What's new in SQL Server 2017. *Microsoft Docs: SQL Docs* [online]. 20-10-2017 [cit. 2021-05-01]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2017?view=sql-server-ver15

[44] RAY, Mike. What's new in SQL Server 2016. *Microsoft Docs: SQL Docs* [online]. 20-10-2017 [cit. 2021-05-01]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2016?view=sql-server-ver15

[45] RAY, Mike. Columnstore indexes – Query performance. *Microsoft Docs: SQL Docs* [online]. 01-11-2019 [cit. 2021-05-01]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-query-performance?view=sql-server-ver15

[46] RABELER, Carl. What's New (Database Engine). *Microsoft Docs: SQL Docs* [online]. 22-06-2016 [cit. 2021-05-01]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/sql/2014/database-engine/whats-new-in-sql-server-2016?view=sql-server-2014

[47] HOYDALSVIK, Geir. What's New in MySQL 5.7? *MySQL Server Blog: News from the MySQL Server Team* [online]. 21-10-2015 [cit. 2021-05-01]. Dostupné z: https://mysqlserverteam.com/whats-new-in-mysql-5-7-generally-available/

[48] AXMARK, David a Michael WIDENIUS. MySQL 5.6 Reference Manual. *MySQL: What Is New in MySQL 5.6* [online]. Aktualizace 01-05-2021 [cit. 2021-05-02]. Dostupné z: https://mysqlserverteam.com/whats-new-in-mysql-5-7-generally-available/

[49] WAGNER, Bert. Should You Use Index Hints? *Data with Bert* [online]. 31-07-2018 [cit. 2021-05-02]. Dostupné z: https://bertwagner.com/posts/should-you-use-index-hints/

[50] RICHARDSON, Ben. What is the difference between Clustered and Non-Clustered Indexes in SQL Server? *SQLShack* [online]. 28-08-2017 [cit. 2021-05-02]. Dostupné z: https://www.sqlshack.com/what-is-the-difference-between-clustered-and-non-clustered-indexes-in-sql-server

[51] NADEL, Ben. The Not-So-Dark Art Of Designing Database Indexes: Reflections From An Average Software Engineer. *Ben Nadel* [online]. 04-07-2018 [cit. 2021-05-10]. Dostupné z: https://www.bennadel.com/blog/3467-the-not-so-dark-art-of-designing-database-indexes-reflections-from-an-average-software-engineer.htm

[52] ASHDOWN, Lance a Tom KYTE. Database Concepts: Indexes and Index-Organized Tables. *Oracle Help Center* [online]. 05-2015 [cit. 2021-07-06]. Dostupné z: https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CNCPT721

[53] AXMARK, David a Michael WIDENIUS. How MySQL Uses Indexes. *MySQL: MySQL 8.0 Reference Manual* [online]. Aktualizace 11-07-2021 [cit. 2021-07-18]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html

[54] AXMARK, David a Michael WIDENIUS. Comparison of B-Tree and Hash Indexes. *MySQL: MySQL 8.0 Reference Manual* [online]. Aktualizace 11-07-2021 [cit. 2021-07-18]. Dostupné z: https://dev.mysql.com/doc/refman/8.0/en/index-btree-hash.html

[55] ROTH, Jason aj. SQL Server Index Architecture and Design Guide. *Microsoft Docs: SQL Docs* [online]. 19-01-2019 [cit. 2021-07-18]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver15

[56] GRYGAŘÍKOVÁ, Michaela. Docker, Kubernetes a kontejnery. Jak fungují a proč je chtít. *MasterDC* [online]. 14-08-2019 [cit. 2021-08-03]. Dostupné z: https://www.master.cz/blog/docker-kubernetes-kontejnery-jak-funguji-proc-je-chtit/

[57] IMDb: IMDB Datasets [online]. [cca 2021] [cit. 2021-08-04]. Dostupné z: https://www.imdb.com/interfaces/

[58] SCHWARTZ, Baron. What is Database Index Selectivity. *Orangematter* [online]. 18-07-2018 [cit. 2021-08-04]. Dostupné z: https://orangematter.solarwinds.com/2018/07/18/what-is-database-index-selectivity/

PŘÍLOHY

PŘÍLOHA 1 – VÝSLEDKY MĚŘENÍ

Ve výchozím nastavení databáze bude dotazován záznam 'Carga de rurales' ve sloupci originalTitle a postupně budou do tabulky movie v průběhu měření přidávána další data

MySQL

Execution time [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	48	30	52	244	1320
2. měření	45	24	48	182	1216
3. měření	39	25	42	225	1294
Průměr:	**44.00**	**26.33**	**47.33**	**217.00**	**1276.67**

Fetching data [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	172	144	94	88	70
2. měření	169	200	137	56	72
3. měření	180	138	119	102	57
Průměr:	**173.67**	**160.67**	**116.67**	**82.00**	**66.33**

Celková doba dotazu (execution + fetching) [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	220	174	146	332	1390
2. měření	214	224	185	238	1288
3. měření	219	163	161	327	1351
Průměr:	**217.67**	**187.00**	**164.00**	**299.00**	**1343.00**

MS SQL

Execution time [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	53	20	22	62	343
2. měření	26	20	12	48	389
3. měření	15	10	30	54	421
Průměr:	**31.33**	**16.67**	**21.33**	**54.67**	**384.33**

Fetching data [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	158	83	71	48	39
2. měření	119	73	63	42	25
3. měření	72	60	40	44	32
Průměr:	**116.33**	**72.00**	**58.00**	**44.67**	**32.00**

Celková doba dotazu (execution + fetching) [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	211	103	93	110	382
2. měření	145	93	75	90	414
3. měření	87	70	70	98	453
Průměr:	**147.67**	**88.67**	**79.33**	**99.33**	**416.33**

Oracle

Execution time [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	37	12	16	32	156
2. měření	36	14	15	13	106
3. měření	35	11	26	18	92
Průměr:	**36.00**	**12.33**	**19.00**	**21.00**	**118.00**

Fetching data [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	210	152	120	132	86
2. měření	95	171	64	79	61
3. měření	133	119	126	111	131
Průměr:	**146.00**	**147.33**	**103.33**	**107.33**	**92.67**

Celková doba dotazu (execution + fetching) [ms]

Počet dat:	100	1 000	10 000	100 000	1 000 000
1. měření	247	164	136	164	242
2. měření	131	185	79	92	167
3. měření	168	130	152	129	223
Průměr:	**182.00**	**159.67**	**122.33**	**128.33**	**210.67**

Tabulka znázorňuje dotazovaný záznam v konkrétním sloupci konkrétní tabulky

	movie100	movie1k	movie10k	movie100k	movie1kk
originalTitle	Carga de rurales	Esmeralda	The Butterfly	She Devil	The Island
titleType	short	short	short	short	short
Genres	Comedy,Short	Comedy,Short	Comedy,Short	Comedy,Short	Comedy,Short

Tabulka znázorňuje přesnou selektivitu - počet unikátních hodnot ve sloupci

Počet dat:	100	1 000	10 000	100 000	1 000 000
originalTitle	99	983	9 648	92 122	726 031
titleType	1	2	2	10	10
genres	14	85	2 70	1 090	1 720

MySQL výsledky měření:

1. Bez použití indexů (full table scan)

Execution time [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	34	27	35	204	2071
2. měření	30	23	44	153	1438
3. měření	31	28	49	193	1268
Průměr:	**31.67**	**26.00**	**42.67**	**183.33**	**1592.33**
	Sloupec titleType - minimální selektivita				
1. měření	25	25	31	27	32
2. měření	16	30	15	25	23
3. měření	26	25	34	48	18
Průměr:	**22.33**	**26.67**	**26.67**	**33.33**	**24.33**
	Sloupec Genres - střední selektivita				
1. měření	22	15	9	32	30
2. měření	18	21	51	46	19
3. měření	16	24	26	22	19
Průměr:	**18.67**	**20.00**	**28.67**	**33.33**	**22.67**

Fetching data [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	172	216	206	153	215
2. měření	180	243	144	156	115
3. měření	364	289	218	116	138
Průměr:	**238.67**	**249.33**	**189.33**	**141.67**	**156.00**
	Sloupec titleType - minimální selektivita				
1. měření	262	381	395	401	6278
2. měření	409	222	513	523	4586
3. měření	343	375	487	533	3793
Průměr:	**338.00**	**326.00**	**465.00**	**485.67**	**4885.67**
	Sloupec Genres - střední selektivita				
1. měření	435	274	352	355	1814
2. měření	472	354	335	421	1773
3. měření	304	365	629	366	1875
Průměr:	**403.67**	**331.00**	**438.67**	**380.67**	**1820.67**

Celková doba dotazu (execution + fetching) [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	206	243	241	357	2286
2. měření	210	266	188	309	1553
3. měření	395	317	267	309	1406
Průměr:	**270.33**	**275.33**	**232.00**	**325.00**	**1748.33**
	Sloupec titleType - minimální selektivita				
1. měření	287	406	426	428	6310
2. měření	425	252	528	548	4609
3. měření	369	400	521	581	3811
Průměr:	**360.33**	**352.67**	**491.67**	**519.00**	**4910.00**
	Sloupec Genres - střední selektivita				
1. měření	457	289	361	387	1844
2. měření	490	375	386	467	1792
3. měření	320	389	655	388	1894
Průměr:	**422.33**	**351.00**	**467.33**	**414.00**	**1843.33**

2. S výchozím indexem (založený na B-stromě)

Execution time [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	39	43	33	64	67
2. měření	20	23	41	26	39
3. měření	21	35	27	32	35
Průměr:	**26.67**	**33.67**	**33.67**	**40.67**	**47.00**
	Sloupec titleType - minimální selektivita				
1. měření	20	17	23	36	50
2. měření	20	92	25	23	23
3. měření	17	22	38	30	26
Průměr:	**19.00**	**43.67**	**28.67**	**29.67**	**33.00**
	Sloupec Genres - střední selektivita				
1. měření	17	25	33	57	27
2. měření	25	28	36	26	28
3. měření	33	17	22	28	16
Průměr:	**25.00**	**23.33**	**30.33**	**37.00**	**23.67**

Fetching data [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	299	253	166	168	218
2. měření	258	241	248	164	171
3. měření	164	255	139	228	169
Průměr:	**240.33**	**249.67**	**184.33**	**186.67**	**186.00**
	Sloupec titleType - minimální selektivita				
1. měření	318	402	457	629	7064
2. měření	233	361	358	743	3561
3. měření	225	265	606	650	3193
Průměr:	**258.67**	**342.67**	**473.67**	**674.00**	**4606.00**
	Sloupec Genres - střední selektivita				
1. měření	273	349	367	465	1011
2. měření	367	290	288	465	708
3. měření	391	388	340	537	627
Průměr:	**343.67**	**342.33**	**331.67**	**489.00**	**782.00**

Celková doba dotazu (execution + fetching) [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	338	296	199	232	285
2. měření	278	264	289	190	210
3. měření	185	290	166	260	204
Průměr:	**267.00**	**283.33**	**218.00**	**227.33**	**233.00**
	Sloupec titleType - minimální selektivita				
1. měření	338	419	480	665	7114
2. měření	253	453	383	766	3584
3. měření	242	287	644	680	3219
Průměr:	**277.67**	**386.33**	**502.33**	**703.67**	**4639.00**
	Sloupec Genres - střední selektivita				
1. měření	290	374	400	522	1038
2. měření	392	318	324	491	736
3. měření	424	405	362	565	643
Průměr:	**368.67**	**365.67**	**362.00**	**526.00**	**805.67**

MS SQL výsledky měření:

1. Bez použití indexů (full table scan)

Execution time [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	74	122	73	171	550
2. měření	79	93	99	126	847
3. měření	134	100	117	143	850
Průměr:	**95.67**	**105.00**	**96.33**	**146.67**	**749.00**
	Sloupec titleType - minimální selektivita				
1. měření	100	79	124	465	156
2. měření	102	94	106	91	150
3. měření	67	58	67	73	127
Průměr:	**89.67**	**77.00**	**99.00**	**209.67**	**144.33**
	Sloupec Genres - střední selektivita				
1. měření	67	81	165	284	646
2. měření	101	60	64	153	85
3. měření	58	73	61	97	62
Průměr:	**75.33**	**71.33**	**96.67**	**178.00**	**264.33**

Fetching data [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	57	55	50	53	172
2. měření	37	22	37	38	69
3. měření	38	47	55	35	49
Průměr:	**44.00**	**41.33**	**47.33**	**42.00**	**96.67**
	Sloupec titleType - minimální selektivita				
1. měření	55	75	184	341	5184
2. měření	21	47	123	511	3971
3. měření	37	40	122	326	4140
Průměr:	**37.67**	**54.00**	**143.00**	**392.67**	**4431.67**
	Sloupec Genres - střední selektivita				
1. měření	25	24	66	241	796
2. měření	21	25	50	87	1025
3. měření	48	25	60	109	993
Průměr:	**31.33**	**24.67**	**58.67**	**145.67**	**938.00**

Celková doba dotazu (execution + fetching) [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	131	177	123	224	722
2. měření	116	115	136	164	916
3. měření	172	147	172	178	899
Průměr:	**139.67**	**146.33**	**143.67**	**188.67**	**845.67**
	Sloupec titleType - minimální selektivita				
1. měření	155	154	308	806	5340
2. měření	123	141	229	602	4121
3. měření	104	98	189	399	4267
Průměr:	**127.33**	**131.00**	**242.00**	**602.33**	**4576.00**
	Sloupec Genres - střední selektivita				
1. měření	92	105	231	525	1442
2. měření	122	85	114	240	1110
3. měření	106	98	121	206	1055
Průměr:	**106.67**	**96.00**	**155.33**	**323.67**	**1202.33**

2. S výchozím indexem (založený na B-stromě)

Execution time [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	56	122	176	50	126
2. měření	67	83	204	28	27
3. měření	64	131	86	30	10
Průměr:	**62.33**	**112.00**	**155.33**	**36.00**	**54.33**
	Sloupec titleType - minimální selektivita				
1. měření	94	98	145	84	350
2. měření	63	90	104	118	225
3. měření	113	61	111	94	264
Průměr:	**90.00**	**83.00**	**120.00**	**98.67**	**279.67**
	Sloupec Genres - střední selektivita				
1. měření	81	294	70	69	216
2. měření	63	149	192	71	102
3. měření	93	96	140	117	101
Průměr:	**79.00**	**179.67**	**134.00**	**85.67**	**139.67**

Fetching data [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	45	37	71	39	222
2. měření	47	22	45	43	197
3. měření	53	37	36	45	101
Průměr:	**48.33**	**32.00**	**50.67**	**42.33**	**173.33**
	Sloupec titleType - minimální selektivita				
1. měření	55	75	200	1151	6713
2. měření	24	44	141	804	5928
3. měření	53	74	126	675	5252
Průměr:	**44.00**	**64.33**	**155.67**	**876.67**	**5964.33**
	Sloupec Genres - střední selektivita				
1. měření	36	105	105	160	1516
2. měření	16	67	63	171	1582
3. měření	28	48	62	252	1331
Průměr:	**26.67**	**73.33**	**76.67**	**194.33**	**1476.33**

Celková doba dotazu (execution + fetching) [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100k	movie1kk
1. měření	101	159	247	89	348
2. měření	114	105	249	71	224
3. měření	117	168	122	75	111
Průměr:	**110.67**	**144.00**	**206.00**	**78.33**	**227.67**
	Sloupec titleType - minimální selektivita				
1. měření	149	173	345	1235	7063
2. měření	87	134	245	922	6153
3. měření	166	135	237	769	5516
Průměr:	**134.00**	**147.33**	**275.67**	**975.33**	**6244.00**
	Sloupec Genres - střední selektivita				
1. měření	117	399	175	229	1732
2. měření	79	216	255	242	1684
3. měření	121	144	202	369	1432
Průměr:	**105.67**	**253.00**	**210.67**	**280.00**	**1616.00**

Oracle výsledky měření:

1. Bez použití indexů (full table scan)

Execution time [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100	movie1kk
1. měření	52	64	101	112	834
2. měření	62	73	40	56	686
3. měření	155	44	85	58	206
Průměr:	**89.67**	**60.33**	**75.33**	**75.33**	**575.33**
	Sloupec titleType - minimální selektivita				
1. měření	34	24	24	33	56
2. měření	43	36	24	33	160
3. měření	23	30	45	34	37
Průměr:	**33.33**	**30.00**	**31.00**	**33.33**	**84.33**
	Sloupec Genres - střední selektivita				
1. měření	29	34	36	29	52
2. měření	24	24	41	26	33
3. měření	33	19	24	42	104
Průměr:	**28.67**	**25.67**	**33.67**	**32.33**	**63.00**

Fetching data [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100	movie1kk
1. měření	223	137	124	129	290
2. měření	168	184	150	169	170
3. měření	149	147	161	101	116
Průměr:	**180.00**	**156.00**	**145.00**	**133.00**	**192.00**
	Sloupec titleType - minimální selektivita				
1. měření	217	511	367	1014	13989
2. měření	211	256	371	946	7760
3. měření	229	238	443	1155	7298
Průměr:	**219.00**	**335.00**	**393.67**	**1038.33**	**9682.33**
	Sloupec Genres - střední selektivita				
1. měření	239	442	389	380	1626
2. měření	238	372	279	468	1682
3. měření	346	279	319	404	1694
Průměr:	**274.33**	**364.33**	**329.00**	**417.33**	**1667.33**

Celková doba dotazu (execution + fetching) [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100	movie1kk
1. měření	275	201	225	241	1124
2. měření	230	257	190	225	856
3. měření	304	191	246	159	322
Průměr:	**269.67**	**216.33**	**220.33**	**208.33**	**767.33**
	Sloupec titleType - minimální selektivita				
1. měření	251	535	391	1047	14045
2. měření	254	292	395	979	7920
3. měření	252	268	488	1189	7335
Průměr:	**252.33**	**365.00**	**424.67**	**1071.67**	**9766.67**
	Sloupec Genres - střední selektivita				
1. měření	268	476	425	409	1678
2. měření	262	396	320	494	1715
3. měření	379	298	343	446	1798
Průměr:	**303.00**	**390.00**	**362.67**	**449.67**	**1730.33**

2. S výchozím indexem (založený na B-stromě)

Execution time [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100	movie1kk
1. měření	57	40	142	40	357
2. měření	28	42	47	49	30
3. měření	24	101	114	90	57
Průměr:	**36.33**	**61.00**	**101.00**	**59.67**	**148.00**
	Sloupec titleType - minimální selektivita				
1. měření	85	31	31	37	38
2. měření	59	32	50	28	44
3. měření	65	24	29	34	35
Průměr:	**69.67**	**29.00**	**36.67**	**33.00**	**39.00**
	Sloupec Genres - střední selektivita				
1. měření	28	64	61	83	36
2. měření	88	30	92	92	30
3. měření	74	46	28	38	24
Průměr:	**63.33**	**46.67**	**60.33**	**71.00**	**30.00**

Fetching data [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100	movie1kk
1. měření	133	162	129	105	512
2. měření	141	168	112	141	126
3. měření	150	133	143	131	101
Průměr:	**141.33**	**154.33**	**128.00**	**125.67**	**246.33**
	Sloupec titleType - minimální selektivita				
1. měření	180	283	701	1435	10313
2. měření	324	263	419	1007	7163
3. měření	196	272	413	868	6561
Průměr:	**233.33**	**272.67**	**511.00**	**1103.33**	**8012.33**
	Sloupec Genres - střední selektivita				
1. měření	191	260	277	335	1345
2. měření	193	266	278	337	1582
3. měření	231	264	349	326	1627
Průměr:	**205.00**	**263.33**	**301.33**	**332.67**	**1518.00**

Celková doba dotazu (execution + fetching) [ms]

	Sloupec originalTitle - maximální selektivita				
Tabulka:	movie100	movie1k	movie10k	movie100	movie1kk
1. měření	190	202	271	145	869
2. měření	169	210	159	190	156
3. měření	174	234	257	221	158
Průměr:	**177.67**	**215.33**	**229.00**	**185.33**	**394.33**
	Sloupec titleType - minimální selektivita				
1. měření	265	314	732	1472	10351
2. měření	383	295	469	1035	7207
3. měření	261	296	442	902	6596
Průměr:	**303.00**	**301.67**	**547.67**	**1136.33**	**8051.33**
	Sloupec Genres - střední selektivita				
1. měření	219	324	338	418	1381
2. měření	281	296	370	429	1612
3. měření	305	310	377	364	1651
Průměr:	**268.33**	**310.00**	**361.67**	**403.67**	**1548.00**