Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Testování závislosti fyzického návrhu databáze na hardware

Testing Dependency of a Physical Database Design on a Hardware

2019 **Tomáš Krása**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Krása**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Testování závislosti fyzického návrhu databáze na hardware
Testing Dependency of a Physical Database Design on a Hardware

Jazyk vypracování: čeština

Zásady pro vypracování:

Výkon jednotlivých SQL příkazů může ovlivňovat celá řada faktorů. Mezi nejdůležitější faktory patří fyzický návrh databáze (tzn. především vytvořené indexy), konfigurace databázového systému a hardware počítače. Náplní této práce je provést testy na běžně dostupných testovacích databázích s cílem ukázat zda-li volba hardware může významným způsobem ovlivnit volbu fyzického návrhu. Jinými slovy, cílem bude na netriviálním množství SQL příkazů, fyzických návrhů databází a několika dostupných hardware otestovat, zda-li nastane situace kdy je určitý index pro SQL příkaz na jednom fyzickém stroji nevýhodný, zatímco na druhém může být prospěšný.

Práce bude probíhat v následujících krocích:
1. Příprava aplikace, která bude generovat fyzické návrhy pro danou databázi.
2. Příprava aplikace, která bude spouštět SQL příkazy a provádět měření doby běhu. Důraz bude kladen na dosažení co nejmenší chyby při měření.
3. Příprava databází a databázových systémů, přičemž důraz bude kladen na eliminaci nežádoucích vlivů jako je nedostatek paměti.
4. Spouštění testů na dostupném hardware, přičemž důraz bude kladen na vyzkoušení různých typů disků a CPU.

Seznam doporučené odborné literatury:

[1] Difallah, Djellel Eddine, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. "Oltp-bench: An extensible testbed for benchmarking relational databases." Proceedings of the VLDB Endowment 7, no. 4 (2013): 277-288.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *30. dubna 2019*

.....

podpis studenta

Poděkování

Rád bych na tomto místě poděkoval panu doc. Ing. Radimu Bačovi, Ph.D., za odbornou pomoc, konzultace a rady při vytváření této bakalářské práce.

Abstrakt

Bakalářská práce se zabývá prováděním testů na dostupné testovací databázi, konkrétně na databázi TPC-H. Jednotlivé úkoly této práce probíhaly přípravou aplikace, která generuje různé fyzické návrhy databáze použitím indexů nad konkrétním SQL příkazem. Výsledkem je tak identifikovat odlišné plány vykonání dotazu. Každý takto vybraný plán vykonání je stejný při spuštění na různém hardware (HW), přičemž čas vykonání může být ovlivněn konfigurací HW. Cílem je tento čas doby běhu SQL dotazu naměřit a vyhodnotit, zda je určitý index na jednom fyzickém stroji výhodný, zatímco na druhém může být nevýhodný.

Klíčová slova: Databáze; Fyzický návrh databáze; index; SQL;

Abstract

Bachelor thesis focuses on database testing, more specifically on TPC-H database testing. The individual tasks of this thesis develop an application that generates miscellaneous physical database designs by using SQL command indexes in order to identify differences in command‘s execution. All of these designs are executed in the same way on various hardware configurations. The goal is to measure the time of the command execution and evaluate if it is beneficial to use specific set of indexes on one machine whereas it might not be beneficial on other machine.

Key words: Database; Physical database design; index; SQL;

Obsah

Seznam použitých zkratek a symbolů

Zkratka	Význam
CPU	Central Processing Unit
DBS	Databázový systém
HDD	Hard Disk Drive
SQL	Structured Query Language
SŘBD (angl. DBMS)	Systém řízení báze dat (Database management system)
SSD	Solid-state drive
RAM	Random Access Memory
HW	Hardware
LRU	Least Recently Used
XML	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Úvod

Existuje mnoho způsobů uložení a vyhledávání dat (záznamů) v počítači. Mezi nejpoužívanější a nejefektivnější patří bezesporu relační databáze. Použitím standardizovaného strukturovacího dotazovacího jazyku SQL v relačních databázích můžeme s těmito informacemi pohodlně a rychle pracovat. Mezi nejdůležitější faktory, které ovlivňují výkon jednotlivých SQL příkazů patří fyzický návrh databáze, konfigurace databázového systému a hardware počítače. Fyzický návrh databáze si můžeme představit jako nízkou úroveň abstrakce, která vychází z logického modelu, jenž zahrnuje implementaci ve specifických technických podmínkách. Zabývá se optimalizací výkonu SŘBD a uložení dat takovým způsobem, aby manipulace s nimi byla rychlá, zabezpečená a škálovatelná. Z pohledu fyzického návrhu databáze na rychlost doby běhu jednotlivých SQL příkazů mají významný vliv především vytvořené indexy. Pojem index se rozumí datová struktura, která danému SŘBD umožňuje v souboru rychleji lokalizovat konkrétní záznamy a tím zefektivnit rychlost odezvy na uživatelské dotazy. Motivací mé bakalářské práce je na netriviálním množství SQL příkazů, fyzických návrhů databází a několika dostupných hardware otestovat, zda-li volba hardware může významným způsobem ovlivnit volbu fyzického návrhu databáze.

1 Fáze databázového modelování

Fáze databázového modelování se člení na jednotlivé úrovně abstrakce pro odstínění nepatřičných hledisek při jeho tvorbě. Rozčlenit právě zkoumanou problematiku při návrhu na jednotlivé, lehce představitelné části nám může do určité míry pomoci tzv. princip tří architektur (P3A). [1]

Proto proces správného návrhu databáze prochází třemi fázemi a to:

- **Konceptuálním návrhem**, jenž poskytuje nejvyšší úroveň abstrakce, kdy modelujeme datové entity jako objekty reálného světa, přičemž neřešíme implementaci ani architekturu.
- **logickým návrhem**, jenž poskytuje střední úroveň abstrakce a za úkol má modelovat entity jako struktury v konkrétním logickém modelu.
- **fyzickým návrhem**, který leží na nejnižší úrovni abstrakce a zabývá se konkrétní implementací ve specifických technických podmínkách. [1,2]

Obecněji lze říci, že výsledkem konceptuálního návrhu je určit, co je obsahem systému. Jak je obsah systémů v dané technologii realizován popisuje logický návrh. A nakonec fyzický návrh databáze určuje, čím je technologické řešení realizováno.

S otázkou, jak získat data z databáze rychleji se pojí řada způsobů. Můžeme vyjmenovat např. využití parametrizovaných dotazů, hromadných operací nebo vhodnou technikou nastavit úroveň izolace transakce. Nesmírnou zásluhu na rychlost vykonání dotazu má vliv již zmíněný fyzický návrh databáze. V podstatě je fáze fyzického návrhu databáze založená na převodu logického návrhu (tabulky a jejich sloupce, integritní omezení) na fyzický návrh, který je použitelný v rámci vybraného databázového systému a provozován na vybraném hardware. [2, 3]

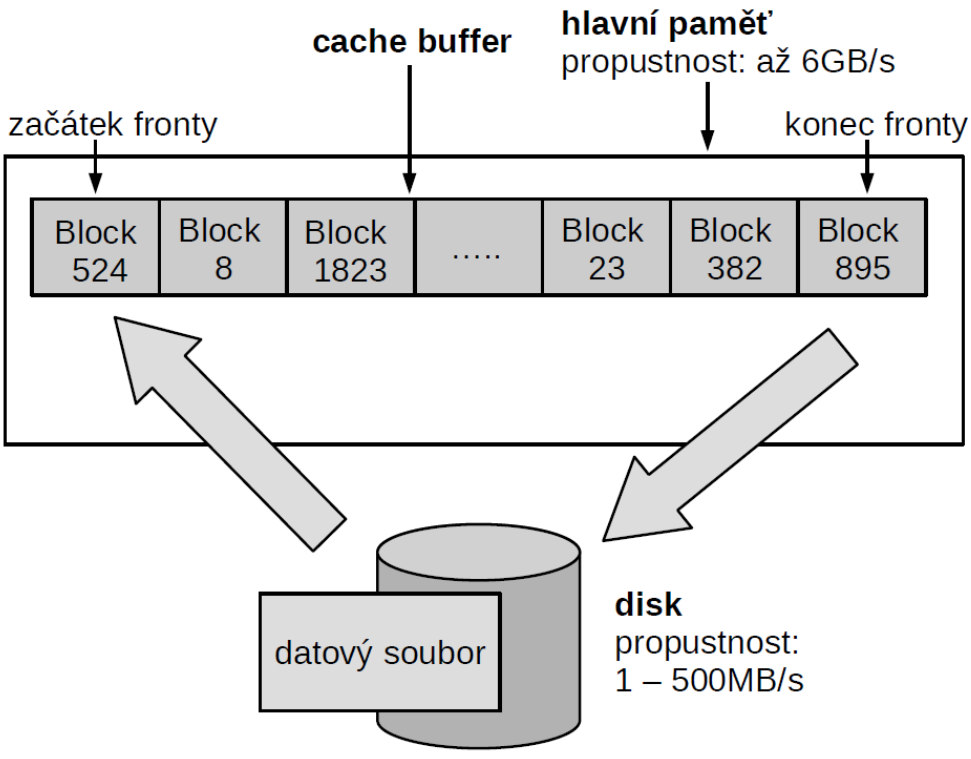
Metodologie fyzického návrhu databáze se skládá z postupů, které se mohou lišit. Vždy je třeba mít na paměti zohlednění specifik a omezení jednotlivých databázových systémů. Metodologie fyzického návrhu databáze se skládají z těchto činností:

- Přenos logického návrhu databáze do cílového databázového systému,
- volba indexů a organizace souborů,
- zabezpečení systému a jejich dat,
- konečné nasazení systému do provozu na vybraném stroji.

Z pohledu provedení rychlosti dotazu musíme nejdříve analyzovat datové struktury, které se ve většině databázových systémech objevují a ujasnit si, jakým způsobem se databázový systém chová při své práci na konkrétním hardware. [2, 3, 4]

1.1 Princip ukládání a načítání dat

Abychom zaručili výše zmíněné výhody databázových systémů a pochopili jejich princip, musíme nahlédnout pod pokličku problematiky ukládání a načítání dat z databáze, a především operace prováděné počítačem. Veškeré datové struktury v DBS jsou složeny z tzv. stránek (angl. pages), které jsou kvůli trvalosti dat uloženy na disku. Stránka na disku nejčastěji zabírá 8kB prostoru. Databázový systém při svém běhu však nemusí načítat data jen z disku počítače, to by bylo vzhledem k rychlosti i nežádoucí, ale může stránky s daty přemístit (alokovat) do hlavní paměti, která může být i 1000x rychlejší než diskové operace. Hlavní paměť se nazývá cache buffer, kterou si můžeme představit implementačně jako frontu (angl. queue). [2, 5]



Obrázek 1.1: Uložení stránek s daty v hlavní paměti na disku.[2]

Postup vyhledáváni stránky DBS začíná nejprve v cache buffer a pokud požadovanou stránku nenajde, požádá o načtení stránky disk. Pokud se daná stránka načte z disku, cache buffer ji označí jako poslední použitá stránka a načte si ji na začátek fronty (blok s pořadovým číslem 524 viz obrázek 1.1). Může nastat situace, kdy je cache buffer plný a v takovém případě označí nejméně používanou stránku, uloží ji na disk (blok s pořadovým číslem 895 viz obrázek 1.1) a na uvolněné místo na začátku fronty načte požadovanou stránku. Tento princip popisuje algoritmus LRU (Least Recently Used).

Na základě těchto poznatků rozlišujeme dva typy přístupů ke stránkám. Logické přístupy (angl. logical access), jenž popisují přístup ke stránce libovolné datové struktury a fyzické přístupy (angl. physical accesses), jenž popisují přístup ke stránce z disku počítače. Pokud rozlišíme operaci vyhledání stránky výhradně z cache buffer, hovoříme o tzv. cache hit. V opačném případě hovoříme o tzv. cache miss. Zavedením těchto pojmů můžeme vytvořit procentuální ukazatel míry úspěšnosti použití cache buffer jako cache hit rate = (cache hits / počet logických čtení) * 100.

Je tedy jasné, že cache buffer má velkou zásluhu na rychlosti vyhledání stránek a pokud počet fyzických přístupů se bude blížit nule, poroste tak úspěšnost použití cache buffer ke 100%, což je žádoucí z pohledu výkonu. [2, 5]

1.2 Důsledek výkonu diskových operací na databázový systém

Abychom mohli porovnávat výkon diskových operací, musíme se zaměřit především na práci, kterou provádějí a odklonit se tak od mnoha údajů, které nám sděluje výrobce. Disk ve své podstatě provádí čtení nebo zápis ve spojitosti se sekvenčním nebo náhodným přístupem. Sekvenčním přístupem identifikujeme takovou operaci, která v souboru zpracovává po sobě jdoucí jeden záznam za druhým. Naopak náhodný přístup je charakterizován zpracováváním souboru po částech s přeskakováním na různé pozice.

Tabulka 1.1: Výkon sekvenčních a náhodných operací pro různé disky [MB/s].[2]

Operace	SATA RAID1	SAS RAID10	Samsung SSD 840 Pro Series	SDHC
Sekvenční čtení	159.3	416.8	516.4	12.1
Sekvenční zápis	177.9	249.6	494.3	11.0
Náhodné čtení 512KB	173.5	239.0	465.5	11.8
Náhodný zápis 512KB	213.1	237.3	475.7	1.5
Náhodné čtení 4KB	2.5	4.2	30.3	3.7
IOPS	613.3	1 020.8	7 392.4	902.6
Náhodný zápis 4KB	12.1	15.9	63.8	1.4
IOPS	2 946.4	3 868.8	15 574.8	344.3

V tabulce 1.1 jsou porovnány odlišné disky různých výrobců a typů, kde jasně vidíme, že nejvyšší výkon z hlediska sekvenčního čtení i zápisu dosahuje disk Samsung SSD. Pokud však srovnáme náhodné operace po blocích o velikosti 4kB, což je řádově stejná velikost využívaná v databázových systémech, vidíme prudký pokles výkonu.

„Co to znamená? Zatímco sekvenční načtení 10 GB souboru na SSD disku bude trvat 19,8s, načtení souboru náhodnými operacemi bude trvat 338s.“ [2, s. 136]

Přímým důsledkem výkonu diskových operací na databázový systém je skutečnost, že se může databázový systém rozhodnout pro sekvenční čtení většího počtu stránek, než aby náhodně načetl menší počet stránek. A to z toho důvodu, že celkový čas vykonání dané operace bude nižší, což je pro databázový systém prioritní. [2, 5]

1.3 Datové struktury v databázových systémech

Všechny záznamy neboli položky jsou uloženy v databázových systémech v různých datových strukturách a to proto, že svými vlastnostmi charakterizují různé způsoby využití. Jedná se především o datové struktury skládající se ze stránek nebo z uzlů v případě, že jsou data uložena do stromové struktury. „*Důležité je, že neexistuje datová struktura, která by měla pro všechny situace optimální vlastnosti, volba vhodné datové struktury tedy vždy závisí na konkrétních datech a dotazech*.“ [2, s. 136]

Za základní rozdělení můžeme považovat datové struktury typu:

- Tabulka typu halda (angl. heap table),
- index.

Toto rozdělení má za důsledek různé operace při vkládání záznamů nebo dotazu uživatele, které mají přímý dopad na jejich složitost a ovlivňují tak výsledný čas odezvy. Pokud zavedeme *n* jako počet záznamů v datové struktuře a *N* jako počet stránek v datové struktuře, můžeme tak popsat složitost počtu operací ovlivněných řádků, kde uvažovaným parametrem bude zmíněné *n* a počtu operací ovlivněných stránek, kde uvažovaným parametrem bude *N*. [2]

1.3.1 **Tabulka typu halda**

Tato datová struktura je označována jako stránkované sekvenční pole, jehož velkou výhodou je efektivní vkládání záznamů. Při vložení záznamu do této datové struktury je postupováno tak, že daný záznam je vložen na první volné místo, které bývá v poslední stránce tabulky. V případě, že v poslední stránce již není místo, je alokována nová stránka hned za původně poslední stránku, do níž je záznam vložen. Pro všechny jednotlivé takto uložené záznamy jsou přiděleny jedinečná čísla, tzv. ROWID, které reprezentuje nejčastěji 32bitové číslo obsahující důležitou informaci o pořadí záznamu na stránce a číslo stránky.

Při dotazu uživatele na záznamy se provádí tzv. metoda sekvenčního průchodu tabulkou (angl. table scan). Pokud by nastal případ, kdy výsledkem uživatelského dotazu bude pouze jeden záznam, systém přistoupí ke všem stránkám a zobrazí výsledek. Složitost operace ovlivněných záznamů je přímo úměrná jejich počtu, tedy *O(n)*, v případě ovlivněných stránek *O(N)*. [2]

1.3.2 **Index**

Požadovaná data z databáze lze načítat i jiným způsobem než sekvenčním průchodem tabulky, a to za pomocí implementace indexů. Pojem index se rozumí taková datová struktura uložená na disku, která je spojená s tabulkou a používá pro načítání a ukládání dat stromovou strukturu tzv. B-strom. Strom je speciálním případem grafu, je tvořen kořenem bez cyklů na jejímž vrcholku jsou uloženy ukazatele na skupinu dat, jenž nazýváme uzly. Index obsahuje klíče vytvořené z jednoho nebo více sloupců v tabulce. Umožňuje tak danému SŘBD rychleji v souboru lokalizovat konkrétní záznamy a tím zefektivnit odezvy na uživatelské dotazy. Jednoduše řečeno, poskytují nám zkrácenou cestu k datům.

Pokud se zamyslíme nad principem vkládáním záznamů do tabulky využívající indexy, znamená to pro databázový systém průchod stromem od kořene k listu, do kterého záznam

zatřídí. Daný databázový systém musí vynaložit výpočetní prostředky na přeskupování kořene stromu vzhledem k vkládaným položkám, aby nedocházelo k degradaci stromu do jedné větve. Vložená položka do B-stromu tak obsahuje klíč a ROWID záznamu v tabulce. Je jasné, že vkládání záznamů do tabulky je tak dražší než v případě tabulky typu heap.

Při požadavku uživatele o položky musíme rozlišit, zda se jedná o jeden konkrétní záznam tzv. bodový dotaz nebo o více záznamů tzv. rozsahový dotaz. Bodový dotaz (angl. point query nebo unique scan) je z hlediska vyhledávání požadavek o konkrétní klíč, kdy při průchodu stromem se vyhledává shoda klíče s uzlem. Je jasné, že pokud označíme výšku stromu h, tak počty přístupů ke stránkám se rovnají h + 1. Naproti tomu rozsahový dotaz v B-stromu (angl. range query nebo range scan) musí vrátit všechny klíče z požadovaného intervalu. Pokud je interval rozsáhlý vzhledem k počtu klíčů z tabulky, výhody rychlosti vyhledávání ze stromové struktury uložených dat degradují z důvodu velkého počtu náhodných přístupů ke stránkám. [2, 5, 6]

1.3.3 **Závislost databázového optimalizátoru na datové struktury**

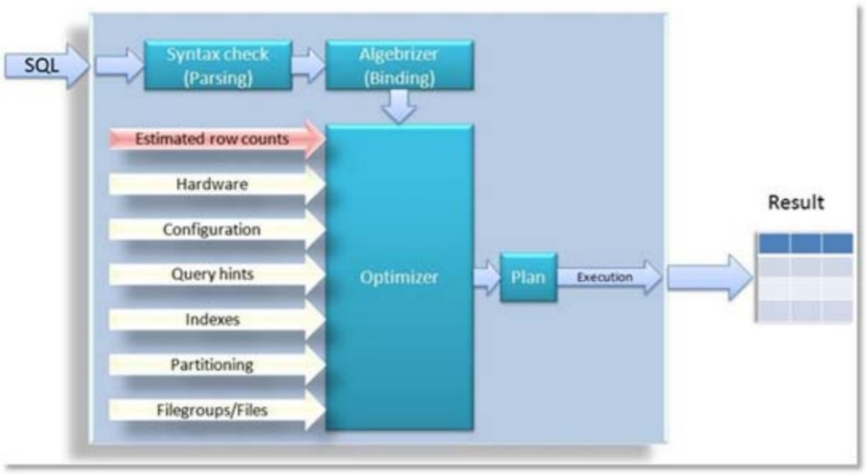
Databázový optimalizátor je jeden z důležitých komponent SŘBD, který má implementován rozhodovací algoritmy pro sestavení exekučního plánu jednotlivých SQL dotazů. Exekučním plánem je myšlena taková posloupnost kroků, kterou musí konkrétní SŘBD vykonat, aby byla data korektně načtena nebo uložena. Optimalizátor může vygenerovat více exekučních plánů a vybere jediný s nejnižšími náklady, které si vypočítá z dostupných statistik dle odhadu velikosti výsledku. Mezi faktory ovlivňující vybraný exekuční plán patří např. systémové prostředky, rychlost diskových operací, využití a rychlost procesoru, dostupná paměť počítače, fyzický návrh databáze atd.

Pokud při realizaci rozsahového dotazu selekce ve spojení s náhodnými přístupy optimalizátor na základě výše popsaných faktorů zjistí, že k vykonání výsledku je zapotřebí načtení většího množství klíčů z B-stromu, rozhodne se raději pro sekvenční průchod tabulkou. [2]

1.3.4 **Plán vykonání dotazu**

Je součástí každého vyhodnocení dotazu (angl. Statistics query). Popisuje složitý proces sestavený z logického plánu vykonání dotazu a fyzického plánu vykonání dotazu. Produktem logického plánu je strom pro vykonání dotazu s použitím relační algebry a výsledkem

fyzického plánu je konkrétní výběr algoritmu implementující jednotlivé operátory logického plánu. Na obrázku 1.2 můžeme vidět proces optimalizace dotazu a shlédnout, že sestavení plánu vykonání dotazu je netriviální záležitostí pro procesor dotazu (angl. Query processor) a každý jednotlivý parametr označený šipkou je relevantní pro optimalizaci ovlivnit celkový výkon dotazů. [2, 3]



Obrázek 1.2: Proces optimalizace dotazu. [3]

2 Pohled na indexy v databázích

Různé databázové systémy např. Oracle, Microsoft SQL Server nebo MySQL nám umožňují nastavovat a vytvářet různé typy indexů. Nastavovat především, zda má být daný sloupec nebo sloupce seřazeny vzestupným či sestupným způsobem.

V případě, kdy je počet atributů v indexu větší než jeden, hovoříme o tzv. složeném indexu (angl. composite index). Při takto vytvořeném indexu z pohledu rychlosti vykonání dotazu záleží i na pořadí vybraných atributů. Například pokud sestavíme index nad tabulkou v pořadí *atribut1*, *atribut2* bude vyhledávání podle *atributu2* náročnější než podle *atribut1*, protože bude docházet k sekvenčnímu průchodu.

Dalším typem indexu je pokrývající index (angl. covering index), jenž je charakterizován tím, že obsahuje veškeré informace, aby pokryl dotaz a zamezil tak přístup do datového souboru relace.

Dále můžeme vybírat pro danou tabulku klastrovaný nebo neklastrovaný index, přičemž musíme brát v úvahu skutečnost, že klastrovaný index ve většině databázových systémech může být nad danou tabulkou jen jeden.

Můžeme také používat filtr rozsahu, který nám dovoluje omezit, na základě nějaké podmínky, pro které řádky se index bude vytvářet. Toto řešení je vhodné pro řešení otázky optimalizace velikosti indexu na disku a rychlosti ukládání záznamů.

Indexy můžeme označit přívlastkem unikátní (angl. unique) jenž znamená, že veškeré hodnoty v indexu jsou jedinečné z čehož vyplývá i jeho použití v případě zavedení primárních klíčů. Z hlediska různých fyzických implementací databáze se budu v příštích oddílech odkazovat na Microsoft SQL Server, jenž byl součástí testu. [2, 6, 12, 13]

2.1 Microsoft SQL Server

Je relační databázový a analytický systém vyvíjený společností Microsoft Corporation. Firma vyvíjí tyto systémy bezmála třicet let na jejichž počátku spolupracovala s firmou Sybase. Výsledkem této fůze měl být rychlý průnik a zavedení produktu na trh. Po několika letech firma Microsoft Corporation vydala bez cizí pomoci svou první verzi Microsoft SQL Server 6.0, která se datuje do roku 1995. Microsoft SQL server je dnes nabízen v několika verzích (edicích), a to:

- SQL Server 2017 Enterprise,
- SQL Server 2017 Standard,
- SQL Server 2017 Express,
- SQL Server 2017 Developer.

Microsoft SQL Server dále nabízí hned několik typů indexů:

- Clustered Index,
- Non-Clustered Index,
- Clustered Columnstore Index,
- Non-Clustered Columnstore Index,
- Spatial Index,
- Primary XML Index.

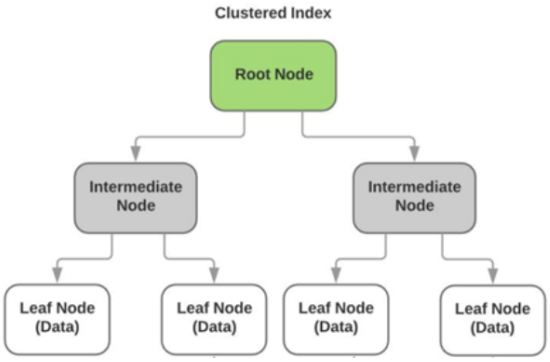
Pro správu databázových instancí nabízí firma Microsoft Corporation nástroj SQL Server Management Studio (SSMS). [14]

2.2 SQL Server Management Studio

Je softwarová aplikace, která byla poprvé spuštěna s verzí Microsoft SQL Server 2005. Poskytuje nám nástroje pro konfiguraci, monitorování a správu instancí serveru SQL Server a databází. Tento nástroj obsahuje skriptové editory a různé grafické nástroje, které pracují s objekty a funkcemi serveru. Příkladem může být grafické zpracování plánu vykonání dotazu optimalizátorem, měření doby vykonání dotazu (elapsed time) a mnoho dalších. [14]

2.3 Clustered Index

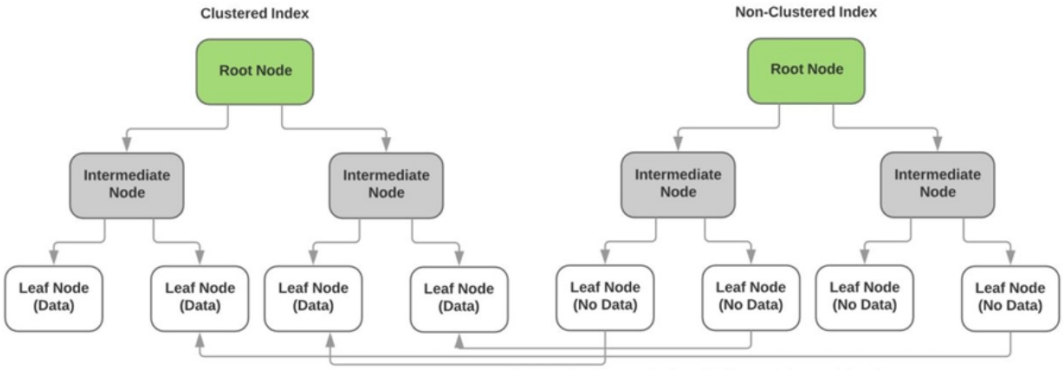
Jedná se o speciální typy indexů, jenž ukládají a třídí datové řádky v tabulce na základě jejich hodnot klíčů. Při implementaci indexů definujeme sloupce tabulky a zavedeme tak podmínky pro vytvoření indexu. Pro každou tabulku se může vytvořit pouze jeden Clustered Index, protože datové řádky mohou být uloženy pouze v jednom pořadí a index obsahuje hodnoty všech sloupců tabulky. Každá implementace Clustered Index vyvolá změnu fyzického uspořádání dat v tabulce. [12, 13]



Obrázek 2.1: Data uložená v Clustered Index.[13]

2.4 Non-Clustered Index

Na rozdíl od Clustered Index má Non-Clustered Index strukturu oddělenou od datových řádků. Takto vytvořená struktura obsahuje hodnoty klíčů zvolených sloupců a každá položka obsahuje ukazatel na řádek dat. Ukazatel řádku v Non-Clustered Index se nazývá lokátor řádku (angl. row locator), jehož struktura závisí na tom, zda jsou datové stránky (angl. pages) uloženy v Clustered Index nebo v tabulce typu Heap. Pokud jsou data uložena v datové struktuře tabulka typu Heap je lokátorem řádku ukazatel na příslušný řádek. Pokud jsou data uložena v Clustered Index je lokátorem řádku ukazatel na klastrovaný indexový klíč. [12, 13]



Obrázek 2.2: Listové uzly Non-Clustered Index obsahují lokátor řádku na klastrovaný indexový klíč. [13]

3 Testování závislosti fyzického návrhu na hardware

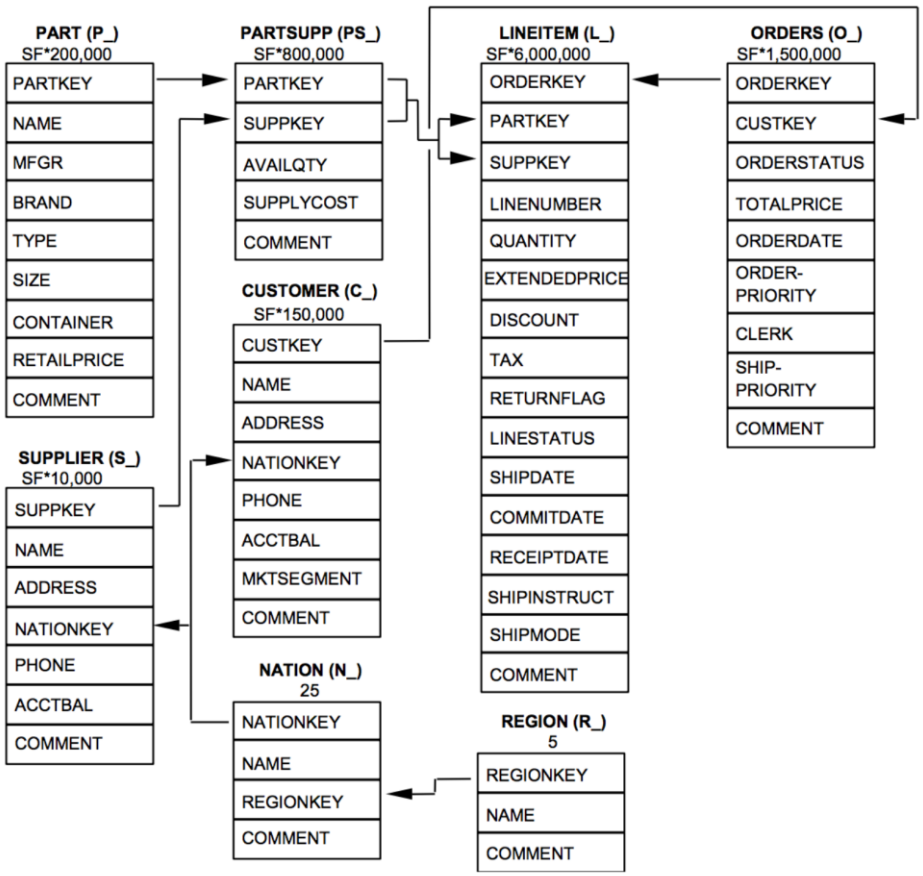
Cílem mé bakalářské práce je zjistit skutečnost, zda volba počítačového stroje, zejména jeho hardware, může výrazným způsobem ovlivnit konkrétní fyzický návrh databáze. Konkrétním fyzickým návrhem databáze je myšleno především jeho vytvořené indexy. To znamená, že budeme měnit datové struktury databáze a zkoumat, zda nastane situace, kdy pro konkrétní fyzický návrh databáze a konkrétní SQL příkaz se z hlediska časové prodlevy vykonání dotazu (angl. elapsed time) projeví vhodnost použít jeden fyzický stroj, kdežto na jiném stroji může být nevýhodný.

Má bakalářská práce probíhala v těchto krocích:

- Příprava databáze.
- Příprava aplikace, která generuje fyzické návrhy pro danou databázi.
- Příprava aplikace, která bude spouštět SQL příkazy a provádět měření doby běhu tzn. časovou prodlevu vykonání dotazu (elapsed time).
- Spouštění testů na dostupném hardware, přičemž je kladen důraz na vyzkoušení různých typů disků a CPU.

3.1 Příprava databáze

Jako referenční databázi pro uskutečnění testů jsem si vybral databázi TPC-H. Tato databáze se skládá ze sady obchodních SQL dotazů a souběžných modifikací dat a ilustruje systémy, které zkoumají velké objemy dat a provádějí dotazy s vysokou mírou složitosti. Významem tak splňují široký průmyslový záběr použití. Pro představu o vztazích a základních vlastnostech databáze uvádím schéma viz obrázek 3.6. Více se lze dočíst v technické dokumentaci k vybrané databázi [7, 8, 9, 10].



Obrázek 3.1: Schéma TPC-H databáze [10].

Popis schématu:

- Závorky za každým názvem tabulky obsahují předponu názvů sloupců pro tyto tabulky.
- Šipky směřují ve směru vzájemných vztahů mezi tabulkami.
- Číslo pod každým názvem tabulky uvádí její mohutnost (počet řádků).
- Označení SF pod některými názvy tabulek označují měřítko (angl. scale factor), které udává verzi databáze TPC-H.

Verze TPC-H

- TPCH_SF1 obsahuje základní velikost řádků tzn. řádově milióny prvků.
- TPCH_SF10 obsahuje základní velikost řádků * 10.
- TPCH_SF100 obsahuje základní velikost řádků * 100.
- TPCH_SF1000 obsahuje základní velikost řádků * 1000.

Dalším krokem je určit výchozí stav databáze tak, aby se od tohoto bodu mohly generovat další, odlišné verze databáze z pohledu fyzického návrhu. Proto zde uvádím výpis s nastavenými parametry (indexy) viz tabulka 3.1.

Tabulka 3.1: Výpis výchozího stavu testované databáze s indexy.

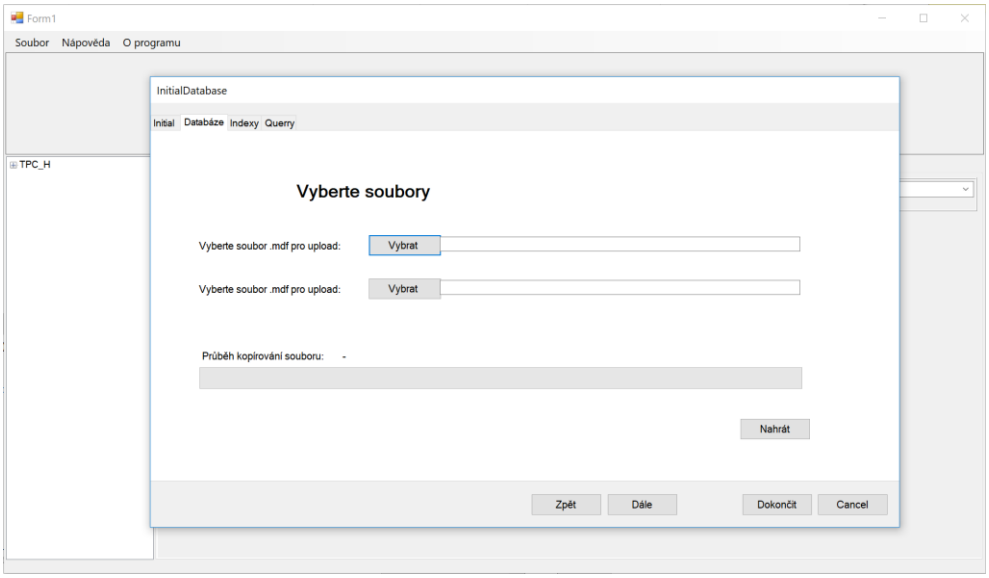
Tabulka	Jméno indexu	Typ indexu	Název sloupce	Primární klíč
customer	customer_pk	CLUSTERED	c_custkey	true
nation	nation_pk	CLUSTERED	n_nationkey	true
part	part_pk	CLUSTERED	p_partkey	true
partsupp	partsupp_pk	CLUSTERED	ps_partkey	true
partsupp	partsupp_pk	CLUSTERED	ps_suppkey	true
region	region_pk	CLUSTERED	r_regionkey	true
supplier	supplier_pk	CLUSTERED	s_suppkey	true
order	order_pk	NONCLUSTERED	o_orderkey	true

Součástí testu jsem vybral TPC-H ve verzi TPCH_SF1, velikost na disku je přibližně 2,367 GB. Následně pro test jsem zvolil tři konkrétní SQL dotazy z celkového počtu 22 SQL dotazů, které dokumentace TPC-H nabízí [7].

3.2 Příprava aplikace

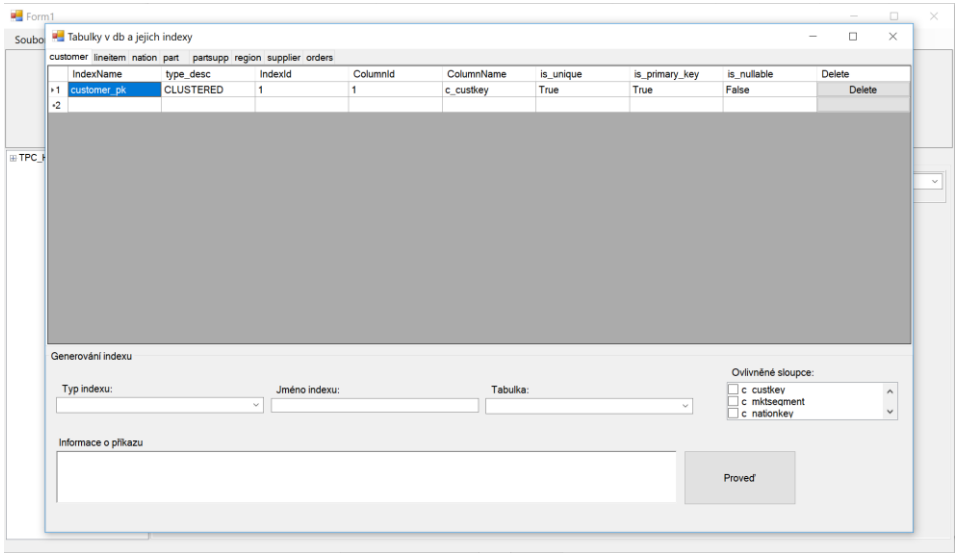
Jako programovací jazyk pro výrobu aplikace jsem zvolil C#. Aplikace je určena pro operační systém Microsoft Windows platformy .NET Framework ve verzi 4.6.1.

Aplikace je určena pro testování databází Microsoft SQL Server, jenž se skládají ze dvou souborů s příponou .mdf (datový soubor) a .ldf (transakční log). Po spuštění aplikace máme možnost nahrát tyto soubory. Pokud se datový soubor bude jmenovat např. TPC_H.mdf, tak nalezneme soubory ve složce v aplikační doméně *Database->TPC_H* viz obrázek 3.1.



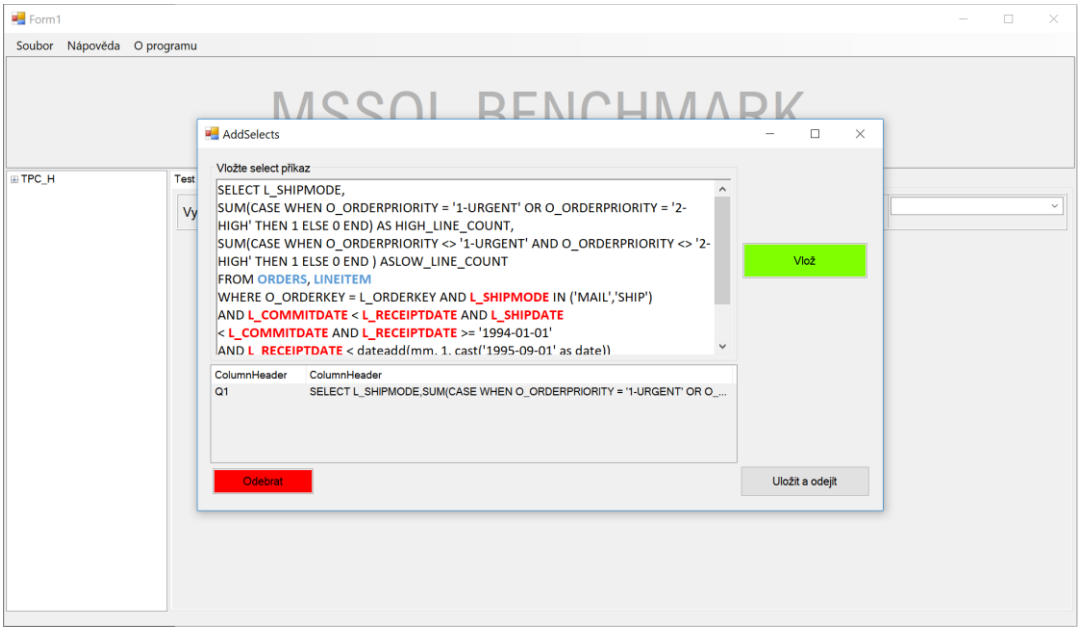
Obrázek 3.2: Spuštění aplikace s možností nahrání databázových souborů

Po úspěšném zkopírování souborů máme možnost si zobrazit všechny tabulky v databázi, především jejich indexy, se kterými máme možnost dále pracovat ve smyslu je mazat nebo vytvořit nové. Cílem je dostat databázi do takového stavu, který bude odrážet námi myšlený výchozí fyzický návrh databáze pro vykonávání testů.



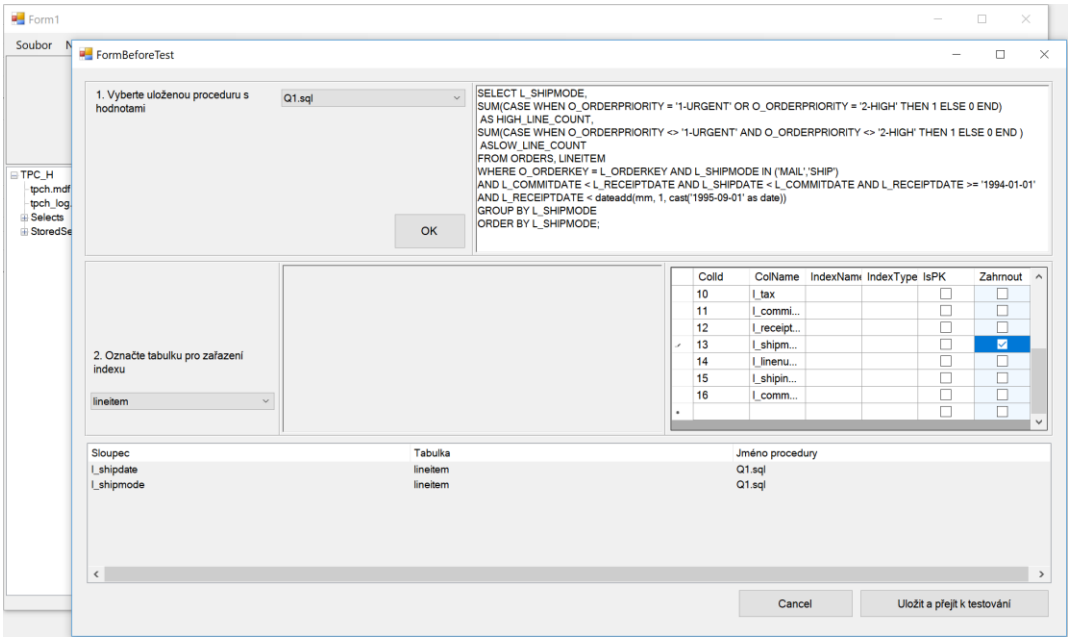
Obrázek 3.3: Tabulky v databázi a jejich indexy

V dalším kroku přidáme SQL příkazy viz obrázek 3.4, které se následně uloží do složky *Database/TPC_H/Selects*, pokud pokračujeme v práci dále se stejnou databází.



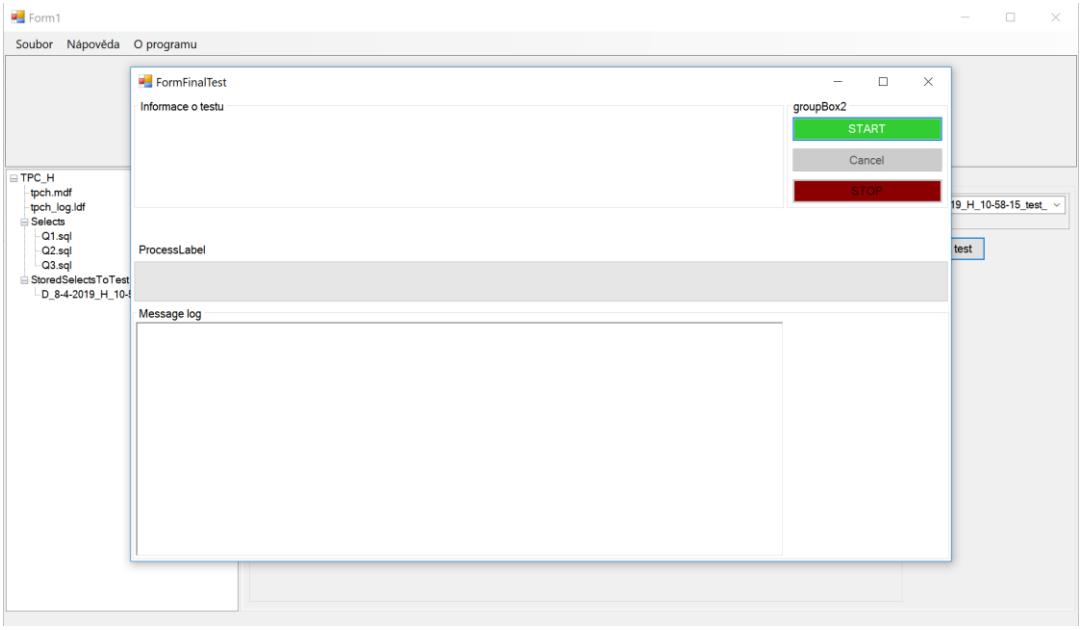
Obrázek 3.4: Průvodce přidáním SQL select příkazů

Jakmile máme přidány SQL SELECT příkazy, následuje výběr sloupců nad tabulkou viz obrázek 3.5, pro které budou vytvářeny Non-Clustered Index. Jsou to především ty sloupce, jejichž název se objevuje v predikátu daného SQL příkazu.



Obrázek 3.5: Výběr sloupců pro klastrované indexy

Tyto kroky vedou k uložení konfiguračního .xml souboru s časovým razítkem v jeho názvu do složky *Database/TPC_H/StoredSelectsToTest,* pokud pokračujeme v analogii práce se stejnou databází. Stromovou strukturu připraveného testu můžeme shlédnout na obrázku 3.6.



Obrázek 3.6: Náhled připraveného testu

3.3 Výběr SQL příkazů pro test

Po rozboru jednotlivých SQL dotazů z TPC Benchmark™ H Standard Specification [7] jsem vybral tři kandidáty. Pro každý SQL dotaz jsem vytvořil pomocí programu jeden XML testovací konfigurační soubor s názvem *Test_1.xml, Test_2.xml, Test_3.xml* jenž jsou součástí elektronické přílohy. Mezi hlavní důvody pro volbu SQL dotazů byl počet různých ovlivněných sloupců v predikátu, což v konečném důsledku mělo přímý dopad na počet různých fyzických návrhů databáze. Vybrané sloupce tak tvoří množinu unikátních prvků nad danou tabulkou a počet různých fyzických návrhů lze tedy matematicky odvodit jako permutace bez opakování vzorcem:

$$P(n) = n!$$

kde n označuje počet vybraných sloupců dané tabulky.

3.3.1 Konfigurační soubor Test_1.xml

Součástí souboru je SQL dotaz, který odpovídá v dokumentaci [7] funkční definici dotazu Discounted Revenue Query(Q19). (TPC Benchmark™ H, c1993 – 2014, str. 60). Vybrané sloupce v predikátu dotazu jsou uvedeny v tabulce 3.1.

Výpis použitého SQL dotazu:

```
SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM LINEITEM, PART
WHERE (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12' AND P_CONTAINER
 IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM
PKG') AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10 AND P_SIZE BETWEEN
 1 AND 5
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN
PERSON')
OR (P_PARTKEY = L_PARTKEY AND P_BRAND ='Brand#23' AND P_CONTAINER IN
('MED BAG', 'MED BOX', 'MED PKG', 'MED
PACK') AND L_QUANTITY >=10 AND L_QUANTITY <=10 + 10 AND P_SIZE BETWEE
N 1 AND 10
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN
PERSON')
OR (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34' AND P_CONTAINER IN
 ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG
PKG') AND L_QUANTITY >=20 AND L_QUANTITY <= 20 + 10 AND P_SIZE BETWEE
N 1 AND 15
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN
PERSON');
```

Tabulka 3.1: Výběr sloupců pro test dotazu ze souboru Test_1.xml

Tabulka v databázi	Sloupec	Zařadit do testu
PART	P_BRAND	ANO
PART	P_CONTAINER	ANO
PART	P_SIZE	ANO
LINEITEM	L_SHIPMODE	ANO
LINEITEM	L_SHIPINSTRUCT	ANO
LINEITEM	L_QUANTITY	ANO

3.3.2 **Konfigurační soubor Test_2.xml**

Odpovídá v dokumentaci [7] funkční definici dotazu Shipping Priority and Order Priority Querry(Q3). (TPC Benchmark™ H, c1993 – 2014, str. 33). Vybrané sloupce v predikátu dotazu jsou uvedeny v tabulce 3.2.

Výpis použitého SQL dotazu:

```
SELECT TOP 10 L_ORDERKEY, SUM(L_EXTENDEDPRICE*(1-
L_DISCOUNT)) AS REVENUE, O_ORDERDATE, O_SHIPPRIORITY
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING' AND C_CUSTKEY = O_CUSTKEY AND L_ORDE
RKEY = O_ORDERKEY AND
O_ORDERDATE < '1995-03-15' AND L_SHIPDATE > '1995-03-15'
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE;
```

Tabulka 3.2: Výběr sloupců pro test dotazu ze souboru Test_2.xml

Tabulka v databázi	**Sloupec**	**Zařadit do testu**

CUSTOMER	C_MKTSEGMENT	ANO
LINEITEM	L_ORDERKEY	ANO
LINEITEM	L_SHIPDATE	ANO
LINEITEM	L_DISCOUNT	ANO
ORDERS	O_CUSTKEY	ANO
ORDERS	O_ORDERDATE	ANO

3.3.3 **Konfigurační soubor Test_3.xml**

Odpovídá v dokumentaci [7] funkční definici dotazu Suppliers Who Kept Orders Waiting Query(Q21). (TPC Benchmark™ H, c1993 – 2014, str. 64). Vybrané sloupce v predikátu dotazu jsou uvedeny v tabulce 3.3.

Výpis použitého SQL dotazu:

```
SELECT TOP 100 S_NAME, COUNT(*) AS NUMWAIT
FROM SUPPLIER, LINEITEM
L1, ORDERS, NATION WHERE S_SUPPKEY = L1.L_SUPPKEY AND
O_ORDERKEY = L1.L_ORDERKEY AND O_ORDERSTATUS = 'F' AND L1.L_RECEIPTDA
TE> L1.L_COMMITDATE
AND EXISTS (SELECT * FROM LINEITEM
L2 WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
 AND L2.L_SUPPKEY <> L1.L_SUPPKEY) AND
NOT EXISTS (SELECT * FROM LINEITEM
L3 WHERE L3.L_ORDERKEY = L1.L_ORDERKEY AND
 L3.L_SUPPKEY <> L1.L_SUPPKEY AND L3.L_RECEIPTDATE > L3.L_COMMITDATE)
 AND
S_NATIONKEY = N_NATIONKEY AND N_NAME = 'SAUDI ARABIA'
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME;
```

Tabulka 3.3: Výběr sloupců pro test dotazu ze souboru Test_3.xml

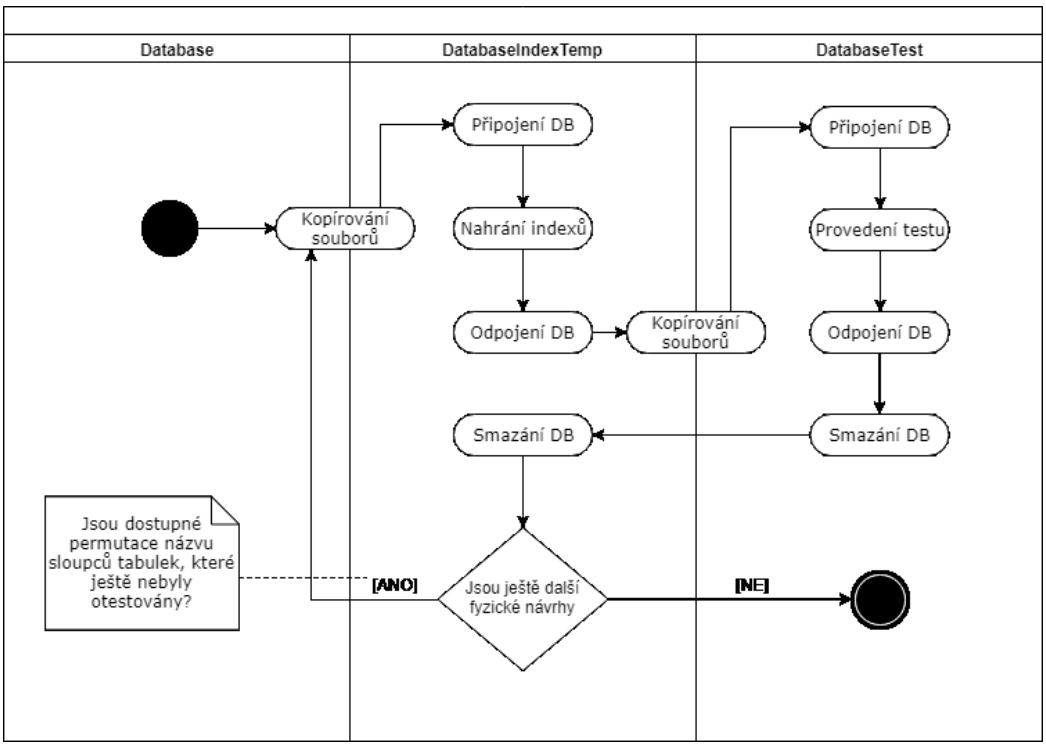
Tabulka v databázi	Sloupec	Zařadit do testu
ORDERS	O_ORDERSTATUS	ANO
LINEITEM	L_RECEIPTDATE	ANO
LINEITEM	L_ORDERKEY	ANO
LINEITEM	L_COMMITDATE	ANO
LINEITEM	L_SUPPKEY	ANO
NATION	N_NAME	ANO

3.4 **Výstup aplikace**

Popsal jsem do této chvíle veškeré potřebné aspekty a vstupní parametry pro vykonání testu, jako jsou:

- Výběr databáze,
- Volba výchozího fyzického návrhu databáze,
- Výběr SQL příkazů,
- Výběr sloupců v predikátu jednotlivých SQL dotazů,

Program má v rámci své aplikační domény vytvořené podsložky `Database`, `DatabaseIndexTemp`, `DatabaseTest`. Výchozí stav fyzického návrhu databáze je uložen ve složce `Database`, odkud se při spuštění testu začnou kopírovat databázové soubory do složky `DatabaseIndexTemp`. Aplikace si připojí právě zkopírovanou databázi, aby mohla změnit její fyzický návrh aplikováním Non-Clustered Index na vybrané sloupce v predikátu jednotlivých SQL dotazů za pomocí již zmíněných permutací. Jiným fyzickým návrhem je považována i změna pořadí sloupců v implementaci Non-Clustered Index. Dále se aplikace odpojí od databáze a kopíruje její soubory do složky `DatabaseTest`, kde spustí test jednoho konkrétního SQL dotazu. Program v prvním dotazu na instanci databáze získá plán vykonání dotazu pomocí příkazu `set showplan_text on` a v druhém dotazu výsledek časové odezvy (elapsed time) nad SQL dotazem. Hodnoty uloží a odpojí se od testované instance. Vytvořené databáze ve složkách `DatabaseIndexTemp` a `DatabaseTest` poté smaže. Algoritmus pokračuje, dokud nevyčerpá všechny možnosti fyzických návrhů databáze nad konkrétním SQL dotazem viz obrázek 3.1. Smyslem průběhu testu je identifikovat různé plány vykonání dotazu pro konkrétní fyzický návrh databáze.



Obrázek 3.7: Průběh kopírování souborů, nahrání indexů a provedení testu nad jedním SQL dotazem.

Součástí elektronické přílohy ve složce `vytvorenyProgram` je `.pdf` soubor s informacemi o spuštění testů s odkazem ke stažení databáze ve svém výchozím stavu fyzického návrhu společně s konfiguračními XML soubory jednotlivých testů.

Výstupem programu je složka, která obsahuje informaci o vykonaném SQL příkazu, konfigurační soubor testu a dokument Microsoft Excel, který je rozdělen na dva sešity, přičemž první obsahuje test všech fyzických návrhů databáze nad konkrétním dotazem a obsahem druhého jsou pouze ty fyzické návrhy, u kterých došlo ke změně plánu vykonání dotazu optimalizátorem společně s plánem vykonání dotazu v textové podobě.

3.5 Výběr testovaného hardware

PC1

- Processor: Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
- RAM: 16 GB
- System type: 64-bit Operating System, x64 based processor
- DISK: SSD Intel SSDPEKKF256G7H, capacity 256GB

PC2

- Processor: Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz 1.99 GHz
- RAM: 16 GB
- System type: 64-bit Operating System, x64 based processor
- DISK: HDD Toshiba MQ04ABF100, capacity 1TB, 5400 rpm

PC3

- Processor: Intel(R) Core (TM) i3-6100U CPU @ 2.3GHz
- RAM: 4 GB
- System type: 64-bit Operating System, x64 based processor
- DISK: SSD HFS128G3BMND-3210A, capacity 128GB

PC4

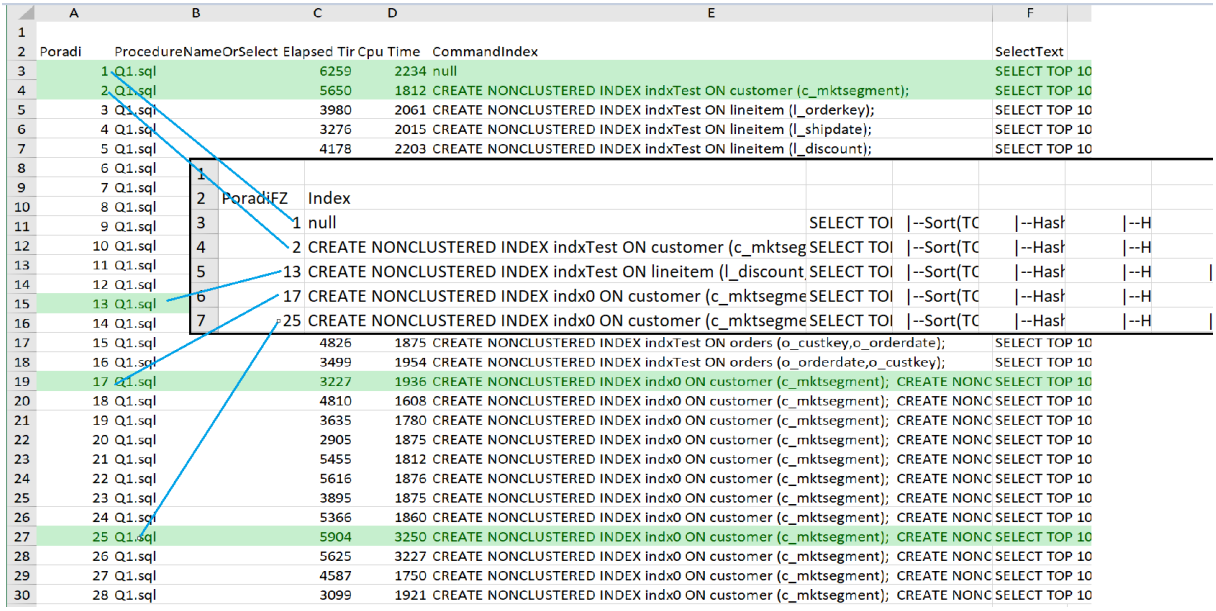
- Processor: Intel(R) Core (TM) i5-6400 CPU @ 2.7GHz
- RAM: 8 GB
- System type: 64-bit Operating System, x64 based processor
- DISK: SSD SanDisk SD7SB6S128G1001, capacity 128GB

PC5

- Processor: Intel(R) Core (TM) i5-6400 CPU @ 2.7GHz
- RAM: 8 GB
- System type: 64-bit Operating System, x64 based processor
- DISK: HDD WDC WD10EZEX-08WN4AO, capacity 1TB, 7200rpm

3.6 **Měření**

Po dokončení běhu aplikace nad konkrétním testem a vytvoření složky s výsledky jsem otevřel uložený soubor Microsoft Excel, přičemž jsem zkontroloval kolik různých plánů vykonání se vytvořilo nad konkrétním dotazem. Jednotlivým plánům jsem přiřadil hodnoty, které odpovídaly pořadí ze všech testovaných fyzických návrhů viz obrázek 3.8.

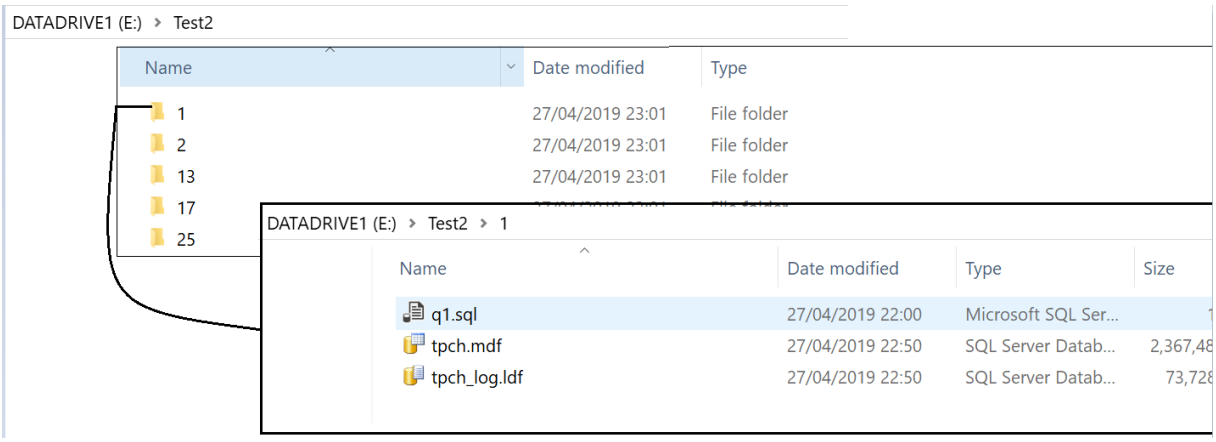


Poradi	ProcedureNameOrSelect	Elapsed Tir	Cpu Time	CommandIndex	SelectText
1	Q1.sql	6259	2234	null	SELECT TOP 10
2	Q1.sql	5650	1812	CREATE NONCLUSTERED INDEX indxTest ON customer (c_mktsegment);	SELECT TOP 10
3	Q1.sql	3980	2061	CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_orderkey);	SELECT TOP 10
4	Q1.sql	3276	2015	CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_shipdate);	SELECT TOP 10
5	Q1.sql	4178	2203	CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_discount);	SELECT TOP 10
6	Q1.sql				
7	Q1.sql				
8	Q1.sql				
9	Q1.sql				
10	Q1.sql				
11	Q1.sql				
12	Q1.sql				
13	Q1.sql				
14	Q1.sql				
15	Q1.sql				
16	Q1.sql				
17	Q1.sql	4826	1875	CREATE NONCLUSTERED INDEX indxTest ON orders (o_custkey,o_orderdate);	SELECT TOP 10
18	Q1.sql	3499	1954	CREATE NONCLUSTERED INDEX indxTest ON orders (o_orderdate,o_custkey);	SELECT TOP 10
19	Q1.sql	3227	1936	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
20	Q1.sql	4810	1608	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
21	Q1.sql	3635	1780	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
22	Q1.sql	2905	1875	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
23	Q1.sql	5455	1812	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
24	Q1.sql	5616	1876	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
25	Q1.sql	3895	1875	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
26	Q1.sql	5366	1860	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
27	Q1.sql	5904	3250	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
28	Q1.sql	5625	3227	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
29	Q1.sql	4587	1750	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10
30	Q1.sql	3099	1921	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);	CREATE NONC SELECT TOP 10

PoradiFZ	Index				
1	null	SELECT TOI	|--Sort(TC	|--Hash	|--H
2	CREATE NONCLUSTERED INDEX indxTest ON customer (c_mktseg	SELECT TOI	|--Sort(TC	|--Hash	|--H
13	CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_discount	SELECT TOI	|--Sort(TC	|--Hash	|--H
17	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegme	SELECT TOI	|--Sort(TC	|--Hash	|--H
25	CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegme	SELECT TOI	|--Sort(TC	|--Hash	|--H

Obrázek 3.8: Přiřazení hodnot k různým plánům vykonání dotazu.

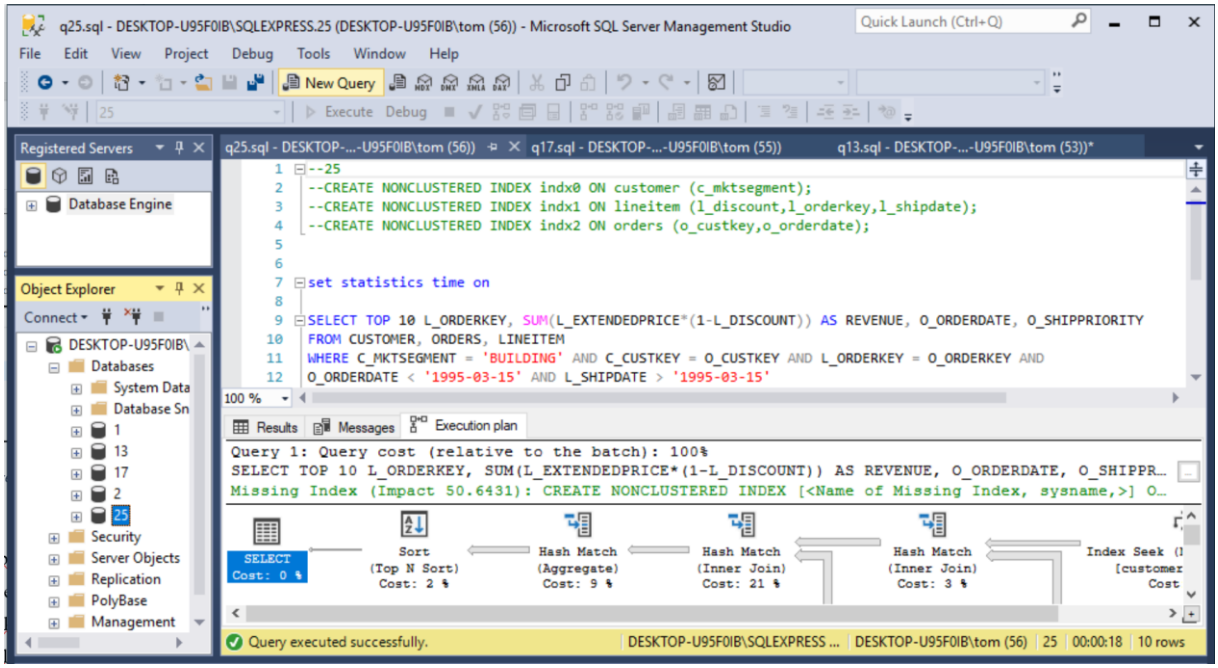
Dále jsem si připravil na disku v testovaném počítači složky s názvem odpovídajícím pořadí fyzických návrhů viz obr. 3.9. Do složek jsem rozkopíroval soubory výchozího stavu instance databáze.



Obrázek 3.9: Způsob uložení souborů daného testu na disku.

Za pomocí SQL Server Management Studio (SSMS) jsem připojil jednotlivé instance databáze a pokračoval vytvořením Non-clustered Index, abych jednotlivé instance uvedl do stavu odpovídajícímu fyzickému návrhu podle výstupu excel souboru aplikace, takže každá instance databáze ve složkách používala při vykonání stejného SQL dotazu jiný exekuční plán. Příslušné plány dotazu jsou uloženy v elektronické příloze ve složkách odpovídajícímu

testu (Test_1, Test_2, Test_3) a počítači (PC1, PC2, PC3, PC4, PC5). V SSMS byla na všech testovaných strojích nastavena maximální hodnota používané paměti serveru na 500MB (Properties->Memory->MaximumServerMemory).



Obrázek 3.10: Plán vykonání dotazu nad instancí označenou číslem konkrétního fyzického návrhu db.

Po tomto nastavení a restartu PC jsem otevřel SSMS a jednotlivé databázové instance pod daným číslem, tedy jiným fyzickým návrhem, jsem za pomocí příkazu `set statistics time on` proved sérii spuštění konkrétního dotazu, kdy první spuštění jsem označil jako *ms0* a následné jako *ms1*, *ms2*, *ms3*.

Postup měření se opakoval analogicky pro dané PC (PC1, PC2, PC3, PC4, PC5) a daný test (Test_1, Test_2, Test_3).

4 Shrnutí dosažených výsledků

Různé fyzické návrhy (FN) pro konkrétní testy jsou označeny čísly 1–52. každému číslu odpovídá jiný typ použitého Non-Clustered Index pro jednotlivý SQL příkaz. Výpis příkazů pro jejich tvorbu jsou součástí výpisu každého testu. Některá časová vyhodnocení jednotlivých FN na konkrétním PC jsou z hlediska rozdílů časů minimální a stanovil jsem tak míru odchylky, kdy se jednotlivá pořadí FN seskupené dle doby vykonání dotazu mohou s daným řádkem prohodit a vytvořit tak jiné pořadí. Časovou prodlevu odchylky jsem stanovil na 300ms.

4.1 Vyhodnocení Test_1 pro PC1 – PC5

Test byl proveden při spouštění SQL dotazu Discounted Revenue Query(Q19). (TPC Benchmark™ H, c1993 – 2014, str. 60).

Tabulka 4.1: Výpis použitých indexů asociovaných k číslu označení FN.

FN	*Index*
1	*Bez indexu*
5	`CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_quantity);`
8	`CREATE NONCLUSTERED INDEX indxTest ON part (p_size,p_brand,p_container);`
10	`CREATE NONCLUSTERED INDEX indxTest ON part (p_brand,p_size,p_container);`
20	`CREATE NONCLUSTERED INDEX indx0 ON part (p_size,p_brand,p_container);` `CREATE NONCLUSTERED INDEX indx1 ON lineitem (l_quantity,l_shipmode,l_shipinstruct);`
32	`CREATE NONCLUSTERED INDEX indx0 ON part (p_brand,p_size,p_container);` `CREATE NONCLUSTERED INDEX indx1 ON lineitem (l_quantity,l_shipmode,l_shipinstruct);`

Tabulka 4.2: Vyhodnocení testu č. 1 vzhledem k časovým prodlevám.

	PC1	PC2	PC3	PC4	PC5
1	9130	16791	9129	9254	14236
5	8957	17166	9586	9320	14136
8	8689	17334	8757	9399	14187
10	8821	17438	9059	8988	14787
20	8858	17202	9423	9409	14861
32	8752	17318	8798	9091	14925

Tabulka 4.3: Určení pořadí FN dle seskupení časových prodlev ms0 jednotlivých PC vzestupně.

SSD						HDD			
PC1		PC3		PC4		PC2		PC5	
FN	ms0	FN	ms0	FN	ms0	FN	ms0	FN	ms0
8	8689	8	8757	10	8988	1	16791	5	14136
32	8752	32	8798	32	9091	5	17166	8	14187
10	8821	10	9059	1	9254	20	17202	1	14236
20	8858	1	9129	5	9320	32	17318	10	14787
5	8957	20	9423	8	9399	8	17334	20	14861
1	9130	5	9586	20	9409	10	17438	32	14925

Všechny počítače (PC1, PC2, PC3, PC4, PC5) z hlediska časové prodlevy vykazuji poměrně stabilní výsledky a mezi různými FN, z čehož vyplývá, že neměly tak velký vliv na odlišnou rychlost na stejný SQL příkaz. Při porovnáni pořadí rychlosti od nejmenší po největší viz tabulka 4.3 si můžeme všimnout odskoku v pořadí pro FN číslo 8 mezi PC1, PC3, PC5 a PC2, PC4. Avšak z pohledu délky jednotlivých dotazů a přičtení, respektive odečtení pomyslné odchylky k hodnotám časové prodlevy FN 8 mezi jednotlivými PC se jedná o zanedbatelnou hodnotu.

4.2 Test 2 pro PC1 – PC5

Test byl proveden při spouštění SQL dotazu Shipping Priority and Order Priority Querry(Q3). (TPC Benchmark™ H, c1993 – 2014, s. 33).

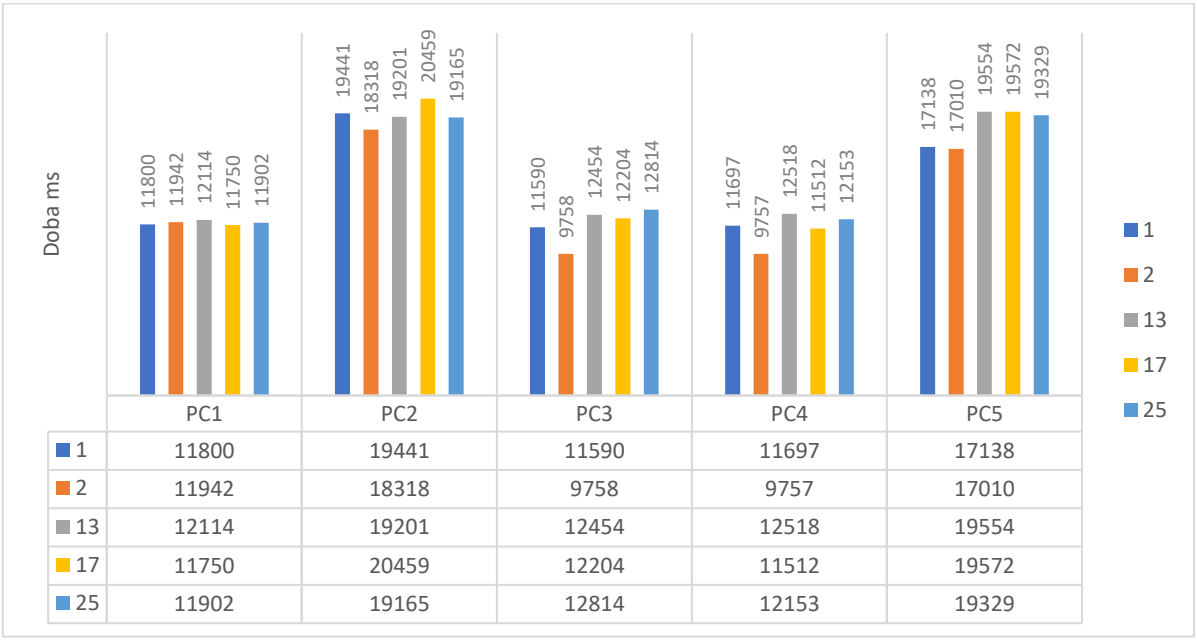
Tabulka 4.4: Výpis použitých indexů asociovaných k číslu označení FN.

FN	*Index*
1	`Bez indexu`
2	`CREATE NONCLUSTERED INDEX indxTest ON customer (c_mktsegment);`
13	`CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_discount,l_orderkey,l_shipdate);`

17	`CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);`
	`CREATE NONCLUSTERED INDEX indx1 ON lineitem (l_orderkey,l_shipdate,l_discount);`
	`CREATE NONCLUSTERED INDEX indx2 ON orders (o_custkey,o_orderdate);`

25	`CREATE NONCLUSTERED INDEX indx0 ON customer (c_mktsegment);`
	`CREATE NONCLUSTERED INDEX indx1 ON lineitem (l_discount,l_orderkey,l_shipdate);`
	`CREATE NONCLUSTERED INDEX indx2 ON orders (o_custkey,o_orderdate);`

Tabulka 4.5: Vyhodnocení testu č. 2 vzhledem k časovým prodlevám.



	PC1	PC2	PC3	PC4	PC5
1	11800	19441	11590	11697	17138
2	11942	18318	9758	9757	17010
13	12114	19201	12454	12518	19554
17	11750	20459	12204	11512	19572
25	11902	19165	12814	12153	19329

Tabulka 4.6: Určení pořadí FN dle seskupení časových prodlev ms0 jednotlivých PC vzestupně

SSD						HDD			
PC1		PC3		PC4		PC2		PC5	
FN	ms0	FN	ms0	FN	ms0	FN	ms0	FN	ms0
17	11750	2	9758	2	9757	2	18318	2	17010
1	11800	1	11590	17	11512	25	19165	1	17138
25	11902	17	12204	1	11697	13	19201	25	19329
2	11942	13	12454	25	12153	1	19441	13	19554
13	12114	25	12814	13	12518	17	20459	17	19572

Jedná se opět o poměrně stabilní test z pohledu porovnání časových prodlev jednotlivých FN na daných strojích. Můžeme opět shlédnout v tabulce 4.6 na FN číslo 17 mezi PC2, PC5 a PC1, PC3, PC4, kdy opět vidíme nesrovnalosti v pořadí, ale vzhledem k malé odchylce v porovnáni s celkovou délkou dotazů se jedna opět o zanedbatelnou hodnotu.

4.3 Test 3 pro PC1 – PC5

Test byl proveden při spouštění SQL dotazu Suppliers Who Kept Orders Waiting Query(Q21). (TPC Benchmark™ H, c1993 – 2014, s. 64).

Tabulka 4.7: Výpis použitých indexů asociovaných k číslu označení FN.

FN	*Index*
1	`Bez indexu`
7	`CREATE NONCLUSTERED INDEX indxTest ON nation (n_name);`
9	`CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_receiptdate,l_orderkey,l_commitdate,l_suppkey);`
15	`CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_orderkey,l_receiptdate,l_commitdate,l_suppkey);`
27	`CREATE NONCLUSTERED INDEX indxTest ON lineitem (l_suppkey,l_receiptdate,l_orderkey,l_commitdate);`

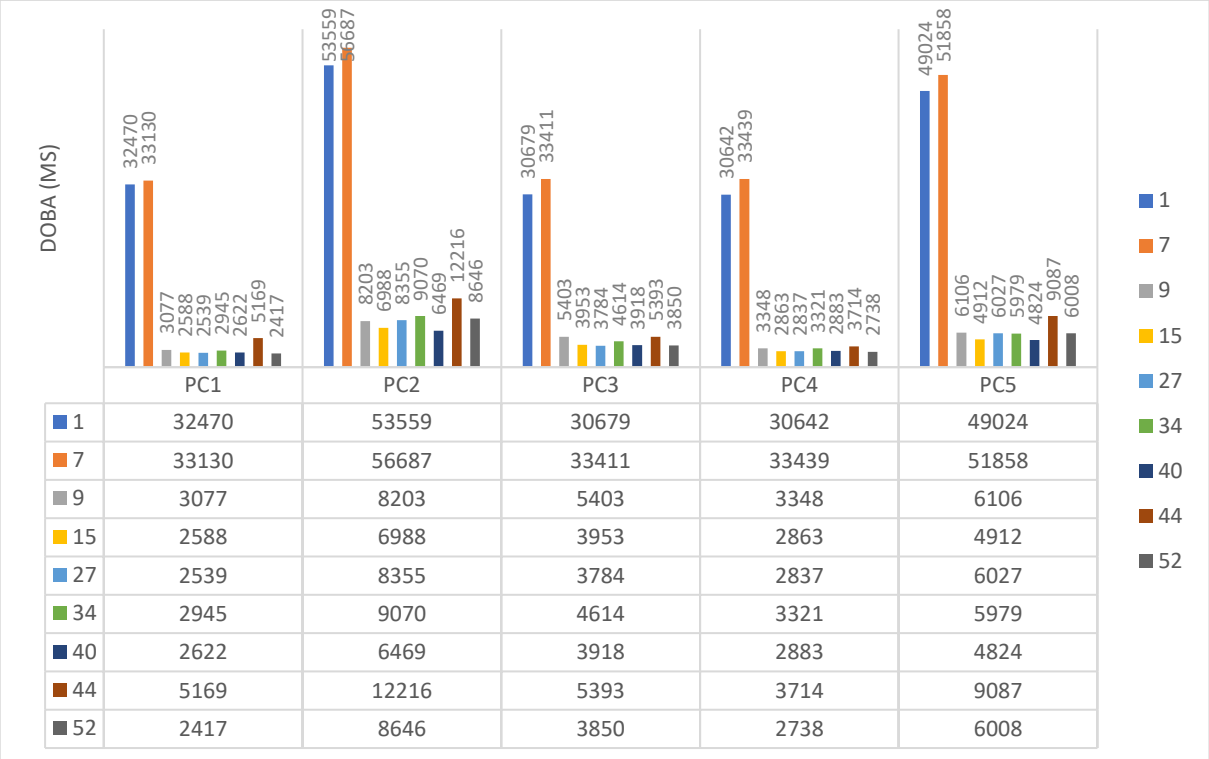
34	`CREATE NONCLUSTERED INDEX indx0 ON orders (o_orderstatus);`
	`CREATE NONCLUSTERED INDEX indx1 ON lineitem (l_receiptdate,l_orderkey,l_commitdate,l_suppkey);`
	`CREATE NONCLUSTERED INDEX indx2 ON nation (n_name);`

40	`CREATE NONCLUSTERED INDEX indx0 ON orders (o_orderstatus);`
	`CREATE NONCLUSTERED INDEX indx1 ON lineitem (l_orderkey,l_receiptdate,l_commitdate,l_suppkey);`
	`CREATE NONCLUSTERED INDEX indx2 ON nation (n_name);`

```
44  CREATE NONCLUSTERED INDEX indx0 ON orders (o_orderstatus);
    CREATE NONCLUSTERED INDEX indx1 ON lineitem
    (l_orderkey,l_suppkey,l_receiptdate,l_commitdate);
    CREATE NONCLUSTERED INDEX indx2 ON nation (n_name);
```

```
52  CREATE NONCLUSTERED INDEX indx0 ON orders (o_orderstatus);
    CREATE NONCLUSTERED INDEX indx1 ON lineitem
    (l_suppkey,l_receiptdate,l_orderkey,l_commitdate);
    CREATE NONCLUSTERED INDEX indx2 ON nation (n_name);
```

Tabulka 4.8: Vyhodnocení testu č. 3 vzhledem k časovým prodlevám.



	PC1	PC2	PC3	PC4	PC5
1	32470	53559	30679	30642	49024
7	33130	56687	33411	33439	51858
9	3077	8203	5403	3348	6106
15	2588	6988	3953	2863	4912
27	2539	8355	3784	2837	6027
34	2945	9070	4614	3321	5979
40	2622	6469	3918	2883	4824
44	5169	12216	5393	3714	9087
52	2417	8646	3850	2738	6008

Tabulka 4.9: Určení pořadí FN dle seskupení časových prodlev ms0 jednotlivých PC vzestupně.

SSD						HDD			
PC1		PC3		PC4		PC2		PC5	
FN	ms0 ⬇	FN	ms0 ⬇	FN	ms0 ⬇	FN	ms0 ⬇	FN	ms0 ⬇
52	2417	27	3784	52	2738	40	6469	40	4824
27	2539	52	3850	27	2837	15	6988	15	4912
15	2588	40	3918	15	2863	9	8203	34	5979
40	2622	15	3953	40	2883	27	8355	52	6008
34	2945	34	4614	34	3321	52	8646	27	6027
9	3077	44	5393	9	3348	34	9070	9	6106
44	5169	9	5403	44	3714	44	12216	44	9087
1	32470	1	30679	1	30642	1	53559	1	49024
7	33130	7	33411	7	33439	7	56687	7	51858

U testu číslo tři vygenerovala aplikace nejvíce FN. Při vykonání dotazu nad různými FN pro konkrétní PC docházelo k obrovským časovým rozdílům viz tabulka 4.9. Pro FN1 a FN7 docházelo u všech PC k prodlevě v řádech desítek sekund. U ostatních FN došlo k razantní časové úspoře vykonaní dotazu za pomoci vytvořených indexů. Při porovnáni pořadí jednotlivých FN vzhledem k časové prodlevě pro konkrétní PC je opět viditelné obdobné pořadí. Po detailnějším průzkumu tabulky 4.9 se ještě můžeme ohlédnout na FN 15 a 52 počítače PC1, PC3, PC4 jenž vykazovaly opačné pořadí seřazení vzhledem k PC2 a PC5. Avšak časový rozdíl FN 15 a FN 52 u SSD disků nepřesáhl hodnotu 171ms, což tvoří opět zanedbatelnou hodnotu vůči odchylce 300ms.

5 Závěr

V této bakalářské práci jsem se zabýval měřením doby běhu (elapsed time) jednotlivých SQL dotazů a nepodařilo se mi vyvrátit fakt, že může volba hardware významně ovlivnit volbu fyzického návrhu databáze. Na třech SQL dotazech, velkém množství fyzických návrhů databází a pěti počítačích jsem nezaznamenal, že by nastala taková situace, kdy je určitý index pro SQL příkaz na jednom stroji nevýhodný, přičemž na jiném může být výhodný. Z výsledků je zřejmé, že toto tvrzení platí i v případech, kdy byly mezi testovanými počítači jejich různé konfigurace z pohledu použitých disků (SSD, HDD) a procesorů.

Díky této práci jsem hlouběji vnořil do problematiky výkonu databáze z mnoha úhlů pohledu, jako například její fyzické uložení a uspořádání na disku, přizpůsobení jejího fyzického návrhu vzhledem k jejímu následnému nasazení nebo využívání různých nástrojů pro konfiguraci, monitorovaní a správu databázových instancí.

Použitá literatura

[1] BRUCKNER, Tomáš. *Tvorba informačních systémů: principy, metodiky, architektury*. Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4153-6.

[2] KRÁTKÝ, Michal a Radim BAČA. *Databázové systémy* [online]. Ostrava, 2015 [cit. 2019-04-23]. Dostupné z: https://dbedu.cs.vsb.cz/files/book/dbcb.pdf

[3] SCHMELING, Holger. *SQL Server Statistics* [online]. Cambridge, UK: Simple-Talk Publishing, 2010 [cit. 2019-04-29]. ISBN 978-1-906434-61-8. Dostupné z: http://assets.red-gate.com/community/books/sql-server-statistics.pdf

[4] *Databázové systémy - Tomáš Skopal* [online]. [cit. 2019-04-30]. Dostupné z: http://www.ms.mff.cuni.cz/~kopecky/vyuka/dbs/lekce01_old.pdf

[5] POKORNÝ, Jaroslav a Ivan HALAŠKA. *Databázové systémy*. Praha: České vysoké učení technické, 1998. ISBN 80-01-01724-9.

[6] STEPHENS, Ryan K., Ronald R. PLEW a Arie JONES. *Naučte se SQL za 28 dní: [stačí hodina denně]*. Brno: Computer Press, 2010. ISBN 978-80-251-2700-1.

[7] *TPC Benchmark™H: Standard Specification* [online]. Revision 2.17.1. San Francisco, c1993-2014 [cit. 2019-04-23]. Dostupné z: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

[8] *TPC-H* [online]. c2001-2019 [cit. 2019-04-23]. Dostupné z: http://www.tpc.org/tpch/

[9] Difallah, Djellel Eddine, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. "Oltp-bench: An extensible testbed for benchmarking relational databases." Proceedings of the VLDB Endowment 7, no. 4 (2013): 277-288.

[10] *WELCOME TO SNOWFLAKE DOCUMENTATION* [online]. San Mateo, CA, United States, c2019 [cit. 2019-04-23]. Dostupné z: https://docs.snowflake.net/manuals/user-guide/sample-data-tpch.html

[11] BEN-GAN, Itzik, Dejan SARKA a Ron TALMAGE. *Querying Microsoft SQL Server 2012: exam 70-461 training kit*. Sebastopol, Calif.: Microsoft, c2012. ISBN 978-0735666054.

[12] *Microsoft: Clustered and Nonclustered Indexes Described* [online]. c2019, 11.2.2019 [cit. 2019-04-23]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-2016

[13] SNAIDERO, Ben. SQL Server non-clustered Indexes: Non-clustered indexes relation to clustered index. *[MSSQL tips]* [online]. 2018, 25/7/2018 [cit. 2019-04-23]. Dostupné z: https://www.mssqltips.com/sqlservertutorial/9133/sql-server-nonclustered-indexes/

[14] *Microsoft Corporation* [online]. [cit. 2019-04-30]. Dostupné z: https://www.microsoft.com/cs-cz/

Seznam příloh

Příloha A: *Naměřené hodnoty*

Tabulka A.1: *PC1 Test_1*

FN	ms0	ms1	ms2	ms3
1	9130	8671	9466	9206
5	8957	9362	8930	9410
8	8689	9043	8893	8926
10	8821	8643	8776	9476
20	8858	8666	9408	8835
32	8752	8822	8942	8933

Tabulka A.2: *PC2 Test_1*

FN	ms0	ms1	ms2	ms3
1	16791	17035	16855	16872
5	17166	17435	17448	17386
8	17334	17147	17113	17133
10	17438	17444	18583	17259
20	17202	17186	17396	17228
32	17318	17203	17315	17471

Tabulka A.3: *PC3 Test_1*

FN	ms0	ms1	ms2	ms3
1	9129	9472	9385	10258
5	9586	9535	9577	9574
8	8757	9090	9289	9224
10	9059	9506	9359	9156
20	9423	9469	9049	9573
32	8798	8927	9104	8894

Tabulka A.4: *PC4 Test_1*

FN	ms0	ms1	ms2	ms3
1	9254	8842	9124	9225
5	9320	9371	9386	9380
8	9399	9304	8849	8908
10	8988	9038	9251	9089
20	9409	9411	9418	9093
32	9091	9035	8925	9427

Tabulka A.5: *PC5 Test_1*

FN	ms0	ms1	ms2	ms3
1	14236	15773	15867	16011
5	14136	16020	16451	18371
8	14187	16334	16735	16743
10	14787	16360	16623	16368
20	14861	17131	15947	17408
32	14925	16360	16036	16726

Tabulka A.6: *PC1 Test_2*

FN	ms0	ms1	ms2	ms3
1	11800	11810	11524	11188
2	11942	11993	12260	12056
13	12114	11573	12257	11687
17	11750	11967	12307	12089
25	11902	12255	11556	12192

Tabulka A.7: *PC2 Test_2*

FN	ms0	ms1	ms2	ms3
1	19441	19502	19795	20019
2	18318	18962	19180	18591
13	19201	19300	19342	19517
17	20459	19089	19469	19128
25	19165	18839	19254	18992

Tabulka A.8: *PC3 Test_2*

FN	ms0	ms1	ms2	ms3
1	11590	11334	12048	11716
2	9758	12590	12205	12335
13	12454	11954	11931	12879
17	12204	11773	12432	12229
25	12814	12714	12919	12415

Tabulka A.9: *PC4 Test_2*

FN	ms0	ms1	ms2	ms3
1	11697	11838	11538	11583
2	9757	9614	11084	11586
13	12518	12560	12619	12443
17	11512	12051	12225	11849
25	12153	12395	12541	12587

Tabulka A.10: *PC5 Test_2*

FN	ms0	ms1	ms2	ms3
1	17138	18744	17847	18077
2	17010	18129	17563	23847
13	19554	21339	20035	20783
17	19572	19835	18874	18882
25	19329	19523	20889	19868

Tabulka A.11: *PC1 Test_3*

FN	ms0	ms1	ms2	ms3
1	32470	32754	33050	35945
7	33130	32438	33228	33607
9	3077	3159	3093	3050
15	2588	2595	2702	2663
27	2539	2320	2382	2446
34	2945	2966	2894	3072
40	2622	2649	2602	2469
44	5169	4613	4898	5150
52	2417	2370	2359	2378

Tabulka A.12: *PC2 Test_3*

FN	ms0	ms1	ms2	ms3
1	53559	54950	53315	52443
7	56687	53172	52774	52715
9	8203	6135	5487	6105
15	6988	5407	5276	5449
27	8355	6189	6232	6272
34	9070	6245	6027	6131
40	6469	5352	5510	5435
44	12216	10586	10724	10835
52	8646	5734	5662	5783

Tabulka A.13: *PC3 Test_3*

FN	ms0	ms1	ms2	ms3
1	30679	30454	30673	30680
7	33411	33439	33467	33014
9	5403	4669	4660	4753
15	3953	3965	3955	4003
27	3784	3808	3623	3706
34	4614	4536	4523	4645
40	3918	3952	3934	3866
44	5393	5370	5421	6029
52	3850	3859	3594	3774

Tabulka A.14: *PC4 Test_3*

FN	ms0	ms1	ms2	ms3
1	30642	30632	30635	30629
7	33439	33423	33428	33289
9	3348	3403	3347	3371
15	2863	2934	2788	2758
27	2837	2664	2422	2485
34	3321	3377	3601	3712
40	2883	2780	2815	2826
44	3714	5547	4707	4987
52	2738	2441	2453	2440

Tabulka A.15: *PC5 Test_3*

FN	ms0	ms1	ms2	ms3
1	49024	51341	50150	57059
7	51858	53457	54051	53812
9	6106	5177	5337	4988
15	4912	4405	4341	4179
27	6027	4512	4673	4191
34	5979	5032	5371	5396
40	4824	4693	4508	4296
44	9087	8114	8621	8575
52	6008	4548	4345	4374