UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2018 David Novák

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Optimalizace využití databázových serverů

David Novák

Bakalářská práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**David Novák**
Osobní číslo:	**I13289**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Optimalizace využití databázových serverů**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Silné postupy pro vybrané databázové servery zaměřené na jejich technické specifikace. Full table scan, index access, partitioning a další postupy pro zajištění optimálního výkonu. V praktické části student provede nejprve srovnání univerzálních dotazů na vybraných databázových systémech a následně provede optimalizace na základě technické specifikace databázového systému.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

Microsoft TechNet, "Which is Faster: Index Access or Table Scan".
Oracle 2011, "Avoiding Table Scans".
Aaron Newman, Marlene Theriault, Bezpečnost v Oracle, ISBN: 80-7226-979-8

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

Datum zadání bakalářské práce: **31. října 2016**
Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci 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., autorský zákon, 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.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 4. 5. 2018

David Novák

PODĚKOVÁNÍ

Rád bych věnoval mé velké poděkování paní Ing. Monice Borkovcové, Ph.D. za velmi dlouhodobou podporu, odborné vedení, cenné rady a obrovskou trpělivost, kterou mi v průběhu zpracování bakalářské práce věnovala. Děkuji své celé rodině za podporu, motivaci a čas během studia.

ANOTACE

Práce se zabývá technickým popisem jednotlivých databázových serverů z hlediska optimalizace a bezpečnosti. Pro každý databázový systém jsou detailně popsány jednotlivé možnosti optimalizace. Na začátku praktické části byly vytvořeny univerzální SQL dotazy, které byly následně otestovány pro jednotlivé databázové servery. Dále byla provedena optimalizace univerzálních SQL dotazů pro každý databázový server. Na závěr byly vyhodnoceny výsledky univerzálních a optimalizovaných SQL dotazů pro jednotlivé databázové servery.

KLÍČOVÁ SLOVA

databáze, databázové systémy, SQL optimalizace

TITLE

Optimalization of use database servers

ANNOTATION

This thesis engage in the technical description of individual database servers in terms of optimization and security. For each database system are the optimization options described in detail. At the beginning of the practical part universal SQL queries were created, which were subsequently tested for individual database servers. In addition the optimization of universal SQL queries for each database server was performed. Finally, the results of universal and optimized SQL queries for individual database servers were evaluated.

KEYWORDS

database, database systems, SQL optimalization

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

AAA	Authentication, Authorization and Accounting
ACID	Atomicity, Consistency, Isolation, Durability
AWS	Amazon Web Services
BI	Business Intelligence
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
CRM	Customer Relationship Management
DDL	Data Definition Language
ERP	Enterprise Resource Planning
GPL	General Public License
GSSAPI	Generic Security Service Application Interface
IaaS	Infrastructure as a Service
IMDBMS	In-Memory Database Management System
IoT	Internet of Things
KDC	Kerberos Key Distribution Center
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MD5	Message-Digest algorithm
NAS	Network Access Server
OLTP	Online Transaction Processing
OSS	Open-Source Software
PaaS	Platform as a Service
PAM	Pluggable Authentication Modules

PGA	Program Global Area
PKI	Public Key Infrastructure
RADIUS	Remote Authentication Dial-In User Service
SaaS	Software as a Service
SGA	System Global Area
SQL	Structured Query Language
SSL	Secure Socket Layer
SSO	Single Sign-On
SSPI	Security Support Provider Interface
TCP/IP	Transmission Control Protocol/Internet Protocol
WAN	Wide Area Network

ÚVOD

Pro lidskou činnost a poznání je důležité zaznamenávat a ukládat jakékoliv informace o reálném světě. Zaznamenávání a ukládání informací provázejí lidské pokolení od nejstarší doby v podobě jeskynních maleb přes vynález papíru, knihtisku až po dnešní počítače a chytré telefony.

Práce se zabývá velmi podstatnou částí informačních technologií, kterými jsou relační databázové systémy. Databázový systém představuje aplikaci, která uchovává databáze společně s různými podpůrnými nástroji. Databázové systémy mohou používat nejrůznější organizace či podniky, ať už se zabývají vývojem v oblasti informačních technologií či jiným odvětvím. Databáze, které společnosti využívají, umožňují uložení velkého množství jakýkoliv dat. Dále databáze dovoluje jednoduchý přístup, manipulaci a aktualizaci uložených dat.

Velký důraz na relační databázové systémy je kladen z hlediska nejaktuálnějších a nejpřesnějších informací. Dalším podstatným hlediskem u relačních databázových systémů je rychlé získání informací po zadání požadavku uživatelem přes aplikaci. Databázový systém musí svým řešením poskytnout vysoký výkon, čímž se rozumí převážně rychlé zpracování dotazu. Jednou z možností, jak zlepšit zpracování dotazu na databázovém systému je optimalizace výkonu databázového systému. Optimalizace cíli na nalezení nejslabšího místa systému. Optimalizace databázového systému může být provedena změnou SQL dotazu, změnou struktury databáze, využitím proprietárních funkcí atd. Optimalizace databázového systému znamená tedy nejen nastavení relačního databázového systému pro konkrétní účely řešení, ale i nalezení slabého místa v samotném SQL dotazu. Velká část této práce se věnuje právě optimalizaci SQL dotazů do databází různých databázových systémů, a to Oracle Database, MS SQL Server, MySQL, PostgreSQL.

Cílem práce je vytvořit na každém databázovém serveru databázový model se stejnou normalizovanou strukturou. Pro testování SQL dotazů na různých databázových systémech bude potřeba vytvořit univerzální SQL dotazy, které budou spustitelné na každém databázovém serveru. Dalším logickým krokem bude následná optimalizace univerzálních SQL dotazů využitím vlastních funkcí jednotlivých databázových systémů. Kroky optimalizovaných SQL dotazů budou posouzeny na základě výstupu exekučních plánů. Na závěr bude použita metoda vícekriteriálních hodnotících variant k analýze závěru o univerzálních a optimalizovaných SQL dotazech pro každý databázový systém.

1 DATABÁZOVÉ SYSTÉMY

Data jsou údaje, která se uchovávají v databázi. Data zůstávají neměnná až do doby, kdy jsou změněna. Mohou být modifikována ručně nebo nějakým automatickým procesem. Z toho vyplývá, že jsou data statická do doby změny vnějším vlivem. [1]

Tabulka 1 – Uložená data v databázi (zdroj vlastní)

JMENO	PRIJMENI	DATUM_NAROZENI	RODNE_CISLO	POHLAVI
David	Novák	22.06.1990	900622/1111	M

Jiná definice uvádí: *„Za data považujeme jakékoliv fyzicky (materiálně) zaznamenané znalosti (vědomosti), poznatky, zkušenosti nebo výsledky pozorování procesu, projevu, činností a prvků reálného světa (reality). Data mohou být zaznamenávána jak klasickým („nepočítačovým“) způsobem, tak na počítačových médiích.“* [2]

Uchovávaná data jsou uložena v databázovém systému, kde se data nachází v tabulkách konkrétní databáze s jasnou organizací do jednotlivých sloupců, kde každý řádek tabulky představuje konkrétní záznam.

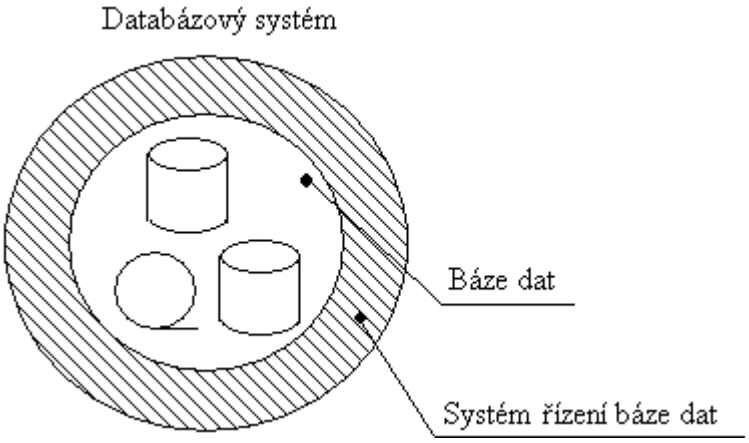
Informace jsou data, která jsou zpracována, aby byla srozumitelná a použitelná pro další práci při zobrazování. Na rozdíl od dat jsou informace dynamické. To znamená, že se dají neustále zpracovávat a prezentovat neomezeným počtem způsobů. [1]

Data jsou údaje, která jsou ukládána. Informace jsou data, která získáváme. [1]

Databáze uchovává velké množství dat, která jsou organizována ve strukturovaném formátu na paměťovém médiu. Lze říci, že databáze také umožňuje jednoduchý přístup, manipulaci a aktualizaci dat.

Architektura klient/server vznikla během 80. až 90. let, kdy byl kladen velký důraz na sdílení dat mezi uživateli. Princip architektury je založen na myšlence uložení dat v centrální databázi, která je umístěna na serveru. Uživatelé neboli klienti mají zajištěn přístup k datům prostřednictvím spuštěných aplikací na vlastních zařízeních (počítačích). Jeden nebo více klientů posílají dotazy na server a ten jim odpovídá na základě jejich požadavků. Komunikace mezi klientem a serverem může být zprostředkována například po síti. [1]

Vnitřní struktura databázového systému je složena z databáze a systému řízení báze dat.



Obrázek 1 – Struktura databázového systému (zdroj [3])

1.1 Databázový systém

Je program poskytující širokou škálu nástrojů jedné nebo více databázím. Mezi nejdůležitější prostředky patří práce s daty, která je umožněna klientům. U databázového systému je kladen vysoký důraz na výkon a také na jeho optimalizaci. V současné době jsou nejcennějším materiálem informace, které je nutné rozsáhle zabezpečit. Principiálně lze začít od přístupu klientů k databázi, manipulaci s daty až po samotné zapojení fyzického stroje, na kterém běží databázový systém.

1.2 Systém řízení báze dat (SŘBD)

Systém řízení báze dat představuje skupinu programů, která umožní řízený přístup do databáze. Mimo jiné tato sada programů umožňuje definovat, vytvořit a udržovat databázi. To znamená, že je možné definovat, manipulovat, obnovovat a spravovat data. V obecné rovině by měl SŘBD poskytnout uživatelům pomocné funkce a nástroje k vývoji databázových aplikací. Jedná se o prostředníka mezi databázovou aplikací a databází. [4]

1.2.1 Funkce SŘBD

Uložení, vyvolání a aktualizace dat

Základní funkce SŘBD, které nejčastěji pracují s daty, jsou uložení, vyvolání a aktualizace dat. [4]

Uživatelsky dostupný katalog

Důležitou součástí SŘBD je zpřístupnění systémového katalogu. Je vhodné, aby byl katalog dostupný jak SŘBD, tak i uživatelům. Každý výrobce se od sebe liší množstvím informací a způsobem jejich použití. Systémový katalog většinou zpřístupňuje název, druh a velikost

datových položek, integritu dat, jména ověřených uživatelů, kteří mají povolený přístup k datům. [4]

Podpora transakcí

Transakce je operace složená z jedné nebo více akcí. Akce nebo série akcí má přístup k obsahu databáze, kdy má možnost i tento obsah měnit. Důležitou podmínkou ke správnému fungování transakce je, aby operace buď provedla všechny změny nebo žádné. Transakce zadává jeden nebo více uživatelů prostřednictvím databázové aplikace, která komunikuje s databází pomocí SŘBD. Každá transakce by měla dodržovat čtyři základní vlastnosti, které se uvádějí pod slovem ACID. Zkratka ACID znamená:

- **Atomičnost** (nedělitelnost) je vlastnost nařizující transakci, aby se provedla celá nebo vůbec.
- **Konzistentnost** zaručuje po zpracování transakce konzistentní stav databáze.
- **Izolace** je vlastnost, kdy každá transakce probíhá nezávisle na ostatních transakcí, které jsou ve stejný okamžik spuštěny.
- **Trvanlivost** udává, že po úspěšném dokončení transakce jsou změny trvalé ve smyslu perzistentně uložených dat v databázi. Neznamená to, že by s daty nešlo jakýmkoliv způsobem pracovat.

Pokud SŘBD podporuje všechny zmíněné vlastnosti transakce, zabrání vzniku nekonzistentních dat při provádění jakékoli transakce. Transakce může být jednoduchá nebo složitá. Příkladem, kdy mohou vzniknout nekonzistentní data je transakce vymazání manažera z databáze a nahrazení jeho místa jiným zaměstnancem. V tomto případě se nejedná pouze o jednu změnu. Pokud při provádění změn transakce selže, například z důvodu havárie zařízení, vznikají tak nekonzistentní data, v důsledku toho, že veškeré změny nejsou uloženy v databázi. Právě proto mechanismus musí zajistit v transakci provedení veškerých změn nebo žádné. [4] [5]

Souběžný přístup

Souběžný přístup je definován jako možnost mnoha uživatelů současně přistupovat ke sdíleným datům. Pokud všichni uživatelé data pouze čtou, tak nehrozí žádná ztráta dat. Nekonzistence dat vzniká, když současně dva nebo více uživatelů čte stejná data a alespoň jeden z nich data aktualizuje. Z tohoto důvodu SŘBD musí poskytnout druh ochrany, která zabrání výskytu nekonzistence dat z důvodu souběžného přístupu. [4]

Služby zotavení

Je nástroj SŘBD, který poskytuje vrácení databáze do konzistentního stavu při selhání transakce. To znamená, že vždy bude existovat možnost vrácení stavu databáze, při jakékoliv havárii, chybě nebo případně i útoku. [4]

Autorizační služby

Služba musí zajistit ochranu před neautorizovaným přístupem do databáze, bez rozdílu, zda se jedná o útok nebo náhodný pokus. Autorizační služby poskytují ochranu citlivých informací mezi uživateli. [4]

Podpora datové komunikace

Služba slouží k zajištění komunikace s databázovým serverem i mimo místní síť (LAN). Jde o to, že SŘBD umožní uživateli komunikovat s databázovým serverem přes síťové rozhraní. Do této služby se dá započítat podpora autentizace prostřednictvím třetích stran, která v podstatě databázovému serveru dává roli klienta. [4]

Služby integrity

Integrita dat je pojem správnosti a konzistence dat [6]. Integrita na úrovni databáze je správnost a konzistence uložených dat. Je to další fáze ochrany dat databáze. V podstatě se jedná o zabezpečení kvality dat, která provádíme omezením. Příklad databáze knihovny umožňuje uživateli si vypůjčit pouze pět knih, a právě toto omezení bude aplikováno a chápáno jako omezení na úrovni databáze. [4]

Je doporučeno kvalitně navrhnout strukturu databáze, která sníží riziko vzniku duplicity dat. Výskyt redundance dat vede k nekonzistenci dat. Nekonzistence dat je změna hodnot, která se nepromítne všude, kde se data opakují. Mezi nejčastější důvody vzniku nekonzistencí patří:

- nevhodně zvolená aktualizace dat,
- zadání chybných údajů,
- slabá provázanost tabulek.

Integritu dat je zaručena vytvořením omezení. Omezení je pravidlo, které je aplikováno nad databázový objekt, obvykle nad tabulkou nebo sloupcem. Omezení slouží k restrikci dovolených hodnot dat vybraného databázového objektu. [5]

Pokud je v databázi zajištěna přesnost dat a jejich konzistentní uložení, lze je správně získat a porovnávat. [6]

Služby podpory nezávislosti dat

Podpora nezávislosti dat umožňuje přenášení modelu dat na různé platformy. Jedná se o to, že je možná modifikace struktury modelu bez ovlivnění dat a aplikací postavených na konkrétním modelu. [4]

Utility

Utility jsou to pomocné programy, které pomáhají správci databáze efektivně spravovat databázi. Mezi utility patří například monitorovací utility, které slouží pro sledování, používání a provoz databáze. [4]

Shrnutí

SŘBD je na každé databázové platformě jinak implementován, takže nabízí různou podporu těchto obecných služeb. Přesněji každý databázový server umožňuje tyto služby, ale má je implementované v odlišné úrovní. [4]

1.3 Databázové modely

Model dat je abstraktní pohled na objekty z reálného světa. Jedná se o proces, který umožňuje přebudovat reálný svět do strukturované kolekce dat. Samotný model představuje prostředek poskytující nejpřesnější popis dat, jejich vzájemné souvislosti, omezení jich samotných nebo omezení, které mezi s sebou mají. [2] [5]

Databázový model je založen na určitém typu modelu dat [2]. Model může a nemusí být definován souborem logických pravidel. V obecné řeči jsou vymodelovány požadavky organizace do popsatelné formy, která umožňuje vyhovět všem požadavkům pomocí vhodně vybraného konkrétního databázového modelu.[4]

1.3.1 Relační databázový model

Byl představen v roce 1969. Zakladatelem tohoto modelu je Dr. E. F. Codd. V současnosti se jedná o nejrozšířenější databázový model. Codd definuje relační datovou strukturu, která chrání data a umožňuje s nimi manipulaci takovým způsobem, který je předvídatelný a odolný vůči chybám. Tento model je založen především na matematických principech teorie množiny a predikátové logice. Relační databázový model zajišťuje čtení dat, konzistenci dat a jejich přesnost. [7]

1.3.2 Objektově orientovaný model

Vznikl už v sedmdesátých letech minulého století. Důsledkem vyššího využívání internetu a nástupu služby World Wide Web se tento model prosadil až v devadesátých letech. Dalším důvodem prosazení objektově orientovaného modelu byl velký tlak na přenos složitějších dat, které relační databázové systémy nepodporovaly. Objektově orientovaný model využívá myšlenky objektově orientovaného přístupu. V odborné literatuře se často uvádí, že tento druh modelu degradoval relační databázi na skladiště dat. Velkou nevýhodou modelu byla nemožnost používání jednoduchých dotazů. [1] [5]

1.3.3 Objektově relační model

Objektově relační model je kombinace objektového a relačního modelu. V přesném znění se relační model rozšířil o objektové prvky z objektově orientovaného modelu. Model má k dispozici vlastnosti obou směrů. Díky tomu výrobci objektově relačních databází mohli například implementovat složitější datové typy. Díky implementaci složitějších datových typů došlo k podpoře obrázků, audio záznamů a videoklipů. Při vysoké popularitě se zdálo, že se tento směr významně prosadí. Nakonec se tomu tak nestalo, protože toto řešení bylo pouze pro malý segment trhu. [1] [5]

2 BEZPEČNOST DATABÁZE

„Zabezpečení systému se týká přístupu a používání databáze na systémové úrovni, jako například uživatelských jmen a hesel. Zabezpečení dat se týká přístupu k objektům databáze (jako tabulkám a pohledům) a činností, které mohou uživatelé s těmito objekty vykonávat.“ [4]

Autentizace

Autentizace je proces ověřování identity uživatele. Ověřování identity probíhá na základě dvou parametrů. Prvním parametrem je většinou název uživatelského účtu, který jednoznačně určuje, kdo se přihlašuje do systému. Druhým parametrem je heslo přiřazené k uživatelskému účtu, díky kterému si systém ověří správnou totožnost uživatele. Uživatel po správném ověření identity vstupuje do informačního systému. [8]

Autorizace

Po autentizaci následuje další stupeň ochrany systému nazývaným autorizace. Autorizace je mechanismus, který umožňuje uživatelům přístup do informačního systému na základě přístupových práv. V odborné literatuře se autorizace označuje jako řízení přístupu [4]. Autorizace též specifikuje oprávnění uživatelů k objektům, ale také jak s nimi může zacházet. Přístupová práva mohou mít několik úrovní přístupu. [4]

2.1 Autentizace operačním systémem

Autentizace operačním systémem je založena na principu ověřování identity databázového uživatele na základě uživatelského účtu v OS. To znamená, že uživatel se přihlásí do operačního systému a účet operačního systému je spárovaný s účtem uloženým v databázi. Výhodou této metody autentizace je pro koncového uživatele jednorázové zadávání hesla, kdy může dále přistupovat k databázi bez zadávání opětovného hesla. Dále toto řešení usnadňuje správu uživatelských účtů, protože databáze si nemusí uchovávat důvěrné informace zejména zašifrované heslo. Metoda přispívá ke snižování nákladů na zabezpečení databáze.

2.2 Autentizace databází

Autentizace databází je proces ověření uživatele, která se provádí pouze za přítomnosti databáze. Databáze ověřuje uživatele pomocí uživatelských účtů, které jsou uloženy v úložišti.

2.3 Síťová autentizace

V dnešní době se uživatelé připojují k databázovému serveru výhradně přes síť WAN. To znamená, že z pohledu ověření uživatele jsou kladeny větší nároky na autentizaci. Z tohoto

důvodu vznikl druh síťové autentizace, který využívá prostředky třetích stran. Třetí strany nabízejí široké spektrum sofistikovanějších autentizačních metod a případně i jejich kombinací.

2.3.1 Kerberos

Kerberos je protokol poskytující autentizační a autorizační služby na nezabezpečené síti prostřednictvím třetí strany. Třetí strana představuje server figurující jako silný centrální ověřovací mechanismus (KDC). Mechanismus má implementovaný silný algoritmus pro ověřování identity a využívá k tomu šifrování pomocí veřejného klíče. Pro klienta jde o to, že pokud chce používat nějakou síťovou službu, musí se nejprve ověřit u důvěryhodného serveru. Na základě správného ověření mu třetí strana vydá lístek, s kterým může klient využívat jeho dostupné služby a zdroje. Kerberos ještě používá technologii podporující jediné přihlášení, která se označuje SSO. SSO poskytne uživateli jediné přihlášení k účtu a díky tomu bude moci uživatel přistupovat kamkoliv prostřednictvím dalších účtů bez nutnosti opětovného zadávání hesla. [8]

2.3.2 PKI

PKI je bezpečnostní infrastruktura, která tvoří síť vzájemně propojených bodů nabízejících nejrůznější způsoby zabezpečení informačních systémů. Mezi tři obecné služby, které může infrastruktura nabízet patří autentizace, integrita a důvěrnost. Integrita zabrání možnosti modifikace zprávy během přenosu. Důvěrnost dává jasné stanovení, kdo může číst zprávu, tj. pouze určený příjemce zprávy. Mezi prostředky, které zajišťují obecné služby a jsou součástí této složité infrastruktury mohou být:

- digitální podpisy,
- certifikáty,
- kryptografie.

PKI je systém založený především na správě a distribuci veřejného klíče prostřednictvím asymetrické kryptografie. Asymetrická kryptografie využívá existence dvou klíčů, které tvoří pár. První zmiňovaný klíč se nazývá veřejný a je libovolně přístupný. Oproti tomu soukromý nebo také privátní klíč musí být před světem utajený a je dostupný pouze vlastníkovi. Princip asymetrické kryptografie funguje na bázi šifrování veřejným klíčem a dešifrování soukromým klíčem. [9]

2.3.3 RADIUS

RADIUS je síťový protokol, který nabízí svým typem protokolu AAA vyšší ochranu přístupu k síti. AAA ukrývá pod svým názvem tři důležité bezpečností služby:

- autentizace,
- autorizace,
- účtování.

Autentizaci zahajuje uživatel zasláním svého požadavku na klienta. Klient je v tomto směru označován jako „*přístupový server*“, kterým může být například NAS. NAS požádá nazpět uživatele o jeho identifikaci. Uživatel pošle identifikační údaje, které NAS dál přepošle jako požadavek přístupu na RADIUS server. Server zpracovává požadavek, který ověřuje v lokální databázi nebo u jiné autentizační služby dostupné v síti (většinou LDAP). Server odpoví jednou ze tří možných zpráv:

- odmítnutí autentizačního požadavku,
- přijetí autentizačního požadavku,
- požadavku na další komunikaci.

Pokud RADIUS server úspěšně ověří klienta posílá přijatý požadavek na NAS. NAS odešle informaci o úspěšně přijatém požadavku, který oznámí uživateli, že je přístup povolen. Jak Doseděl uvádí, [8] protokol RADIUS můžeme chápat jako „*nadprotokol*“ na který je možno dosadit další vhodný autentizační protokol viz LDAP. [10]

2.3.4 LDAP

LDAP je protokol, který je reprezentován adresářem uloženým na serveru využívající architekturu klient-server. Slouží pro uložení, čtení a zejména vyhledávání dat, která jsou uložena ve stromové struktuře. Protokol je otevřený internetový standard vytvořený na základě protokolu X.500, který byl svou složitostí těžko implementovatelný. Právě proto vznikl protokol LDAP, který je odlehčenou verzí předchozího protokolu a poskytuje následující možnosti:

- pracuje přímo přes TCP/IP,
- nabízí využití jednoduchého řetězcového formátu pro datové elementy,
- jednoduchou verzi pravidel pro přenos zašifrovaných dat.

LDAP je složen ze čtyř modelů (informační model, model jmen, funkční model, model zabezpečení), které podrobně popisují operace, uložení dat a používání LDAP. Nejzajímavějším modelem je bezpečnostní model, který podle [8] definuje, jak jsou citlivé informace chráněny před neautorizovanými uživateli a činnostmi. To znamená, že adresář LDAP nabízí možnost obecné bezpečnosti, autentizace a autorizace. Například uložení seznamu uživatelů a jejich oprávnění, jak mohou pracovat s informačním systémem. Jednotliví

výrobci databázového systému si můžou naimplementovat libovolně model zabezpečení, který jim zajistí zmíněné funkce (služby). [8]

2.4 Způsoby databázové autorizace

Databázová autorizace stanovuje uživateli rozsah pracovního prostoru, v kterém se může pohybovat. Znamená to, že systém umožní uživateli přístup k vlastnímu prostoru, ale i k prostoru jiných uživatelů. Pracovní prostor v databázi může představovat schéma nebo databázi, jelikož každý výrobce má tuto terminologii nazvanou odlišně. Prostor je složen z kolekce objektů, ke kterým bude mít uživatel odlišná přístupová práva. Odlišná přístupová práva udávají uživateli rozdílné možnosti práce s objekty.

V předchozím odstavci jsou zmíněny tři hlavní prvky využívající se k zabezpečení databázového systému:

- uživatelé,
- objekty,
- práva.

Každý databázový systém tyto tři hlavní prvky zabezpečení používá, ale každý z nich je má jinak naimplementované. Jazyk SQL je proto hlavním prostředkem, který nám umožňuje zabezpečit databázový systém. Příkazy SQL jsou aplikovány pro bezpečnostní omezení. Pod bezpečnostním omezením si lze představit například přístup uživatelů k uloženým datům v databázi. [11]

2.4.1 Uživatelský účet

Uživatelský účet je identifikátor, podle kterého databáze ověřuje, jaký uživatel se přihlašuje. Každý identifikátor je v databázi unikátní neboli jedinečný. K uživatelskému účtu se většinou uživatel přihlásí na základě uživatelského jména a hesla. Uživatel má možnost si nastavit autentizační metodu ve svém uživatelském účtu. Další nastavení uživatelského účtu slouží k autorizaci neboli jaká oprávnění má uživatel v databázi přidělené. Při vytváření uživatelských účtů si lze nastavit další parametry, které se liší databázovou platformou. Například po vytvoření uživatelského účtu má účet zpravidla přidělen svůj úložný prostor, který se může lišit velikostí. Jedná se o to, že každá databáze nabízí odlišné autentizační metody, odlišný řízený přístup v databázi atd. Databáze si uchovává seznam uživatelských účtů. [13]

2.4.2 Skupiny uživatelů

Skupina je stejně jako uživatel rozpoznán podle identifikátoru. Skupina sdružuje pod jedním identifikátorem několik uživatelských účtů (uživatelů). Identifikátoru skupiny jsou nastavena přístupová práva. Existují dva způsoby přidělování:

- **Jediný identifikátor** – uživatel má přiřazen jeden identifikátor skupiny.
- **Více identifikátorů** – uživatel má přiřazen více než jeden identifikátor skupiny.

Kombinace předchozích dvou je nejčastěji implementována databázovými platformami. To znamená, že uživatelé mají svůj vlastní identifikátor a pak můžou mít identifikátor skupiny. Uživatel tím získává oprávnění podle uživatelského účtu a skupiny. [11]

2.4.3 Oprávnění

Oprávnění se někdy v databázovém světě označuje pojmem přístupová práva. Oprávnění uděluje uživateli, uživatelům nebo skupině přistupovat k databázovému systému. Přístup může být rozdělen do jednotlivých úrovní, jelikož mezi uživatele patří i správce a vývojář. Dále je v databázovém systému řešen přístup k objektům. Každý uživatel bude moci přistupovat ke svým objektům. Úroveň přístupu umožní uživateli případně uživatelům přistupovat k objektům, které jim nepatří neboli nejsou jejími vlastníky. Z tohoto důvodu je možné oprávnění rozšířit na to, co budeme moci s objekty dělat, takže oprávnění s operacemi nad objekty vlastními případně cizími. Shrnutím je detailně popsáno, objektové oprávnění, ale v databázi ještě existuje systémové oprávnění, které dává uživateli možnost používat databázové prostředky. Prostředky jsou myšleny procesorový čas, operační paměť, velikost disku. [5]

2.4.4 Role

Role představuje databázový objekt, který podle [5] „*je pojmenovaná skupina oprávnění.“*, která lze přidělovat jednomu nebo více uživatelům. Díky seskupení, role usnadňuje přiřazení a správu oprávnění. Role jsou dobrým mechanismem, který zajistí konzistentní sadu oprávnění v databázi. Role poskytují následující výhody:

- Při vytvoření nebo po vytvoření uživatelského účtu lze přiřadit role. Tento postup umožní jediným příkazem uživateli přiřadit danou kolekci oprávnění, kterou potřebuje k výkonu práce.
- Efektivnější práce administrátora – role umožnuje administrátorovi zkrátit dobu práce s přiřazováním nebo odebíráním oprávnění.

- Při odstranění uživatelských účtů zůstávají role v databázi – například při smazání a následném vytvoření stejného uživatele, role dovoluje administrátorovi přiřadit stejná oprávnění.

Jde o to, že při odstranění objektu nemusíte každému uživateli odebírat oprávnění jednotlivě, jelikož uživatel získává oprávnění přes role. [5]

2.4.5 Pohledy

Popis pohledů vychází z dokumentu viz [5] Pohledy jsou uložené databázové dotazy, které uživateli zobrazují upravený výběr dat z jedné nebo více tabulek. Jak uvádí Oppel [5], pohled představuje *„virtuální tabulku“*, která má stejný vzhled jako tabulka a do jisté míry se i tak chová. Pohled oproti tabulce nedisponuje fyzicky uloženými daty, protože v pohledu je uložena pouze definice dotazu. V mnoha databázových systémech jsou vytvořeny pohledy pomocí jazyka SQL.

Pohledy poskytují bezpečnostní prvek, který uživatelům omezí přístup k datům. Uživateli je zajištěn přístup k určitým sloupcům a řádkům tabulky, kdežto k jiným sloupcům a řádkům lze přístup zablokovat. Mezi užitečné vlastnosti pohledů z hlediska bezpečnosti patří:

- **Maskování sloupce** je provedeno vynecháním sloupců v pohledu, které uživatel nepotřebuje vidět nebo mu nejsou dovoleny.
- **Maskování řádků tabulky** je provedeno prostřednictvím klauzule **WHERE**, která provádí tzv. restrikci záznamů neboli vrácení omezeného počtu řádků. Uživatel má možnost si zobrazit pouze důležité řádky, které nutně potřebuje k práci.
- **Utajení složitějších databázových operací** například. pohled umožňuje zobrazení dat i z více než jedné tabulky prostřednictvím databázové operace spojení, která je uživateli skryta.

Materializované pohledy

Materializovaný pohled má oproti normálnímu pohledu uloženy v sobě i fyzicky uložená data. Výhodou je rychlejší zpracování dotazu, jelikož má v sobě uložená data, která jsou ihned k dispozici. Nevýhoda vyplývá z uložených dat, která nemusejí být aktuální. [13]

3 DATABÁZOVÉ SYSTÉMY A BEZPEČNOST

Společnosti zabývající se vývojem nabízejí své produkty pro různý segment trhu, kde je nutné rozlišovat technické specifikace každého databázového serveru. Ty přinášejí některá omezení a limity a řadí tak každý databázový server pro použití v určitých oblastech.

V následující části budou popsány jednotlivé databázové servery. Nejprve budou popsány obecně a poté bude podrobně popsána optimalizace databázového serveru z hlediska zpracování dotazu, dále různé způsoby přístupu k databázovému serveru a s tím související řízení přístupu.

Optimalizace

Optimalizace je neustálý proces změn systému. Hlavním cílem optimalizace je dosažení maximálního výkonu se stávajícími prostředky. Alfou omegou celého procesu optimalizace je nalezení slabé části systému. Při nalezení slabé části následuje návrh možných řešení až po výběr nejlepšího způsobu řešení. Existuje několik úrovní modifikací systému. Vhodné je začít od nejvyšší úrovně až po tu nejnižší. [14]

Optimalizátor

Optimalizátor je rozhodující komponenta SŘBD, která analyzuje dotazy jazyka SQL a určuje jejich efektivní způsoby provádění. Pro každý dotaz vygeneruje optimalizátor jeden nebo více exekučních plánů. Všechny vygenerované plány dotazu po spuštění vracejí stejné výsledky. Optimalizátor vybere nejoptimálnější exekuční plán s nejnižšími náklady, které jsou vypočteny na základě dostupných údajů ze statistik. Náklady představují systémové prostředky jako je vstupně/výstupní operace, využití procesoru a paměť serveru. Uživatelé nemají možnost pracovat s optimalizátorem, protože běží na pozadí. Jedinou možností, jak lze řídit optimalizaci dotazu je použití hintů (hints), pokud je daný databázový systém podporuje. [15]

Exekuční plán

Exekuční plán je posloupnost kroků, kterou databáze musí vykonat, aby se provedl SQL příkaz, to znamená, že v databázi se provádí SQL dotaz podle exekučního plánu. Exekuční plán zobrazuje způsob, jakým budou data vyjmuta případně vložena. Exekuční plán specifikuje zejména čtyři oblasti, kterými jsou:

- Pořadí a nákladnost jednotlivých operací.
- Upřesňuje všechny databázové objekty, které se na dotazu podílejí.

- Specifikuje přístupové cesty k jednotlivým objektům.
- Metody spojení dvou tabulek.

Exekuční plán zobrazí slabá místa dotazu, která lze lehce určit. Nabízí cenný zdroj informací pro ladění výkonu jednotlivých dotazů. [15]

Indexy

Popis indexů vychází z tohoto dokumentu viz [13]. Index je datová struktura fyzického úložiště, která je v databázi reprezentována jako databázový objekt, umožňující rychlý přístup k záznamům v tabulce. Přístup závisí na hodnotách jednoho nebo více sloupců. Index ukládá hodnoty dat a ukazatele na řádky, ve kterých se tyto hodnoty vyskytují. Index v databázi je velice podobný rejstříku v knize, podle kterého lze najít požadovaný obsah. Většinou jsou hodnoty dat uloženy v seřazeném pořadí buď vzestupně nebo sestupně. Důvodem je možnost prohledat rychle index a nalézt odpovídající hodnoty. Na základě nalezení odpovídající hodnoty v indexu, lze pomocí ukazatele najít řádek, který obsahuje tuto hodnotu.

Hlavním smyslem indexu je, aby databáze prováděla dotazy co nejefektivněji, to znamená za krátký časový úsek s menšími nároky na systémové prostředky. Jedná se zejména o dotazy s vyhledávacími podmínkami.

Nevýhoda vzniká při aktualizaci indexu, když je do tabulky přidáván nový řádek nebo stávající řádek aktualizoval hodnotu indexovaného sloupce. Díky tomu databáze bude mít větší režii na příkazy INSERT a UPDATE.

Je doporučeno, vytvářet indexy nad sloupci, kde bude často používána vyhledávací podmínka. Zpravidla databáze vytváří implicitní indexy při tvorbě tabulky. Většinou se jedná o dva případy:

- Sloupec, nad kterým je definován primární klíč tabulky.
- Jakýkoliv sloupec anebo kombinace více sloupců s omezením jedinečnosti (unikátnosti).

U primárního klíče je to z důvodu předpokladu častého přístupu, který bude prováděn přes hodnoty primárního klíče. U druhého případu je cílem ulehčit kontrolu unikátní hodnoty, kterou databáze musí provádět, při vkládání nového řádku nebo při aktualizaci současného řádku. S použitím indexu databáze prohledá pouze hodnoty, které jsou v indexu a nemusí prohledávat celou tabulku kvůli kontrole omezení.

Existuje celá řada indexů, které nabízejí databázové systémy. Většinou každý databázový systém nabízí jeden nebo více druhů. Indexy jsou optimalizovány, jelikož nabízejí různé druhy přístupu do databáze:

- **B-strom** je základní datová struktura pro uložení dat**.**
- **B+strom** vychází z B-stromu a na rozdíl od něj má uložená data pouze v listech.
- **Bitmapový index** pracuje pouze s binárními hodnotami.

Clusterový index má v posledním uzlu (listu) uloženy veškeré sloupce záznamu. Tabulka může mít pouze jeden clusterový index.

Neclusterový index nemá v posledním uzlu uloženy konkrétní záznamy, ale má uložený odkaz na clusterový index. Pod odkazem si lze představit např. primární klíč. Použití neclusterového indexu má za následek vyšší režii. Důvodem je prohledání obou indexů k nalezení konkrétních záznamů.

Přístupová cesta

Popis přístupových cest vychází z dokumentu viz [16]. Přístupová cesta je způsob, jakým SŘBD získává hledaná data z datových souborů. Zdroj představuje tabulku, pohled, nebo výsledek spojení. Vyhledaná data slouží k provedení SQL dotazu. Existuje několik druhů přístupových metod, které lze použít pro odlišně napsané dotazy. Optimalizátor rozhoduje o metodě přístupu. Výběr přístupové metody může mít velký vliv na výkon databázového systému. Každý výrobce databázového systému nabízí odlišné přístupové cesty. Mezi nejčastější přístupové cesty patří Index Scan, který je efektivnější na načtení menšího počtu řádků z tabulky. Při načítání větší části tabulky je účinnější použít Full Table Scan.

Každý výrobce databázového systému má své vlastní algoritmy pro optimalizaci volby přístupových cest a minimalizaci celkových nákladů na přístup. Optimalizace je měřena na základě vstupně/výstupních operací a využití CPU. Optimalizátor rozhodne o výběru možných přístupových cest k vyhledání dat a odhadne náklady na provedení dotazu využitím právě možných přístupových cest a jejich kombinací. Optimalizátor zkoumá klauzule **WHERE** a **FROM** pro výběr přístupové cesty.

Full Table Scan

Metoda prohledává všechny řádky z tabulky a filtruje řádky, které nevyhovují podmínce. Databázový systém zvolí tuto možnost v případě, když není k dispozici index nebo je potřeba vyhledat větší množství dat. Full Table Scan je pomalejší než Index Scan, jelikož musí

prohledat větší množství dat. Metoda používá více blokové čtení, které povede k obrovskému počtu vstupně/výstupních operací, což velmi zatíží databázový systém.

ROWID Scan

Jedná se o nejrychlejší způsob načtení jednoho záznamu prostřednictvím hodnoty ROWID. ROWID představuje přesné umístění řádku v databázi. Přesněji to znamená, že ROWID reprezentuje fyzickou adresu každého uloženého záznamu.

Index Scan

Indexové prohledávání vrací záznamy z indexu pomocí hodnot indexovaného sloupce tabulky, které jsou zadány v příkazu SQL. Jelikož existuje mnoho druhů indexů, tak se nabízí i mnoho typů metod k prohledávání:

- **Index Unique Scan,**
- **Index Range Scan,**
- **Index Full Scan,**
- **Index Fast Full Scan.**

Cluster Access Scan

Metoda slouží k načtení všech řádků, které mají stejnou hodnotu klíče. Řádky pocházejí z tabulky, kde se vyskytuje indexový klastr (cluster).

Hash Access Scan

Metoda umisťuje řádky v hash cluster na základě stejné hodnoty hash. Všechny řádky obsahující stejnou hodnotu hash jsou uloženy ve stejném datovém bloku.

Metody spojení

Výsledkem SQL operace spojení je tabulka, která může být složena ze dvou nebo z více množin. Množina představuje buď tabulku nebo pohled. Spojení je provedeno na základě odpovídajících sloupců, čímž vznikne vztah mezi tabulkami. Následující ukázka reprezentuje spojení prostřednictvím jazyka SQL:

Inner Join (vnitřní spojení) vrací záznamy, když se hodnoty z obou tabulek shodují.

Outer Join (vnější spojení)

- **Right Outer Join** (pravé vnější spojení) vrací všechny záznamy z pravé tabulky a shodné záznamy z levé tabulky.
- **Left Outer Join** (levé vnější spojení) vrací všechny záznamy z levé tabulky a shodné záznamy z pravé tabulky.
- **Full Outer Join** (úplné vnější spojení) vrací všechny záznamy, pokud je shoda v levé nebo pravé tabulce.

Každý databázový systém má implementované své vlastní algoritmy, které spojují tabulky pomocí zmíněných spojení z jazyka SQL.[13]

Partitioning

Jedná se o techniku, která umožňuje databázovému systému fyzicky rozdělit objemné datové tabulky do menších segmentů. Rozdělení tabulky je podle logického uspořádání dat. Dělené segmenty je možné umístit na různé disky, což poskytuje několik výhod:

- rychlejší přístup k datům,
- požadavky může zpracovávat více disků,
- lepší manipulace s tabulkami.

Vhodným příkladem rozdělení tabulky může být tabulka obsahující seznam všech lidí v obci. Tabulku rozdělíme podle měst. [17]

3.1 Oracle Database

Úvodní část oddílu vychází z dokumentu Oracle viz [18]. Oracle je moderní multiplatformní databázový systém s velice pokročilými možnostmi zpracování dat, vysokým výkonem a snadnou škálovatelností. Oracle Database je oficiální název databázové platformy firmy Oracle Corporation. Společnost je na trhu více než třicet pět let a patří mezi přední vývojářské firmy databázového systému.

Poslední vydaný produkt je nabízený ve verzi (12.2.0.1.0) ve dvou edicích:

- Oracle Database 12c **Enterprise Edition** (EE) nabízí výkon, dostupnost, škálovatelnost a zabezpečení potřebné pro kritické aplikace, které pracují s velkým množstvím zpracování online transakcí (OLTP). Edice obsahuje všechny součásti Oracle Database a je možné zakoupit další balíčky, které umožňují další varianty využití.

- Oracle Database 12c **Standard Edition 2** (SE2) je plnohodnotná databáze, poskytující snadné použití, sílu a výkon. Obsahuje všechny funkce nezbytné pro vytváření podnikových aplikací.

Písmeno **c** znamená cloud a odráží práci společnosti Oracle při rozšiřování svého relačního databázového systému. Firmám umožňuje konsolidaci a správu databáze jako cloudové služby prostřednictvím multitenántní (multifunkční) architektury Oracle a schopností zpracování dat v paměti (in-memory).

Novinky (nové funkce):

- cloud computing,
- multitenant architecture,
- automatic data optimalization,
- heat map,
- data redaction,
- database In-Memory,
- adaptive query optimalization.

3.1.1 Zpracování SQL dotazu

Tento pododdíl vychází z dokumentace Oracle viz [19]. Zpracování SQL příkazu probíhá v několika fázích:

- analýza,
- optimalizace,
- generování zdrojových řádků,
- provedení dotazu.

Analýza oddělí části příkazu do datové struktury, aby ji mohli dále jiné rutiny zpracovávat. Během analýzy dochází k syntaktické a sémantické kontrole. Fáze optimalizace spočívá v určování nejúčinnější metody přístupu k požadovaným datům, kterou provádí optimalizátor. Výstupem z optimalizátoru je optimální exekuční plán, který je nejvhodnější ze všech možných exekučních plánů. Generátor zdroje řádků zpracuje optimální plán, z kterého vytvoří iterativní exekuční plán, který je použitelný pro databázi. Iterativní plán je binární strom rozdělený na dílčí kroky, kdy každý krok stromu představuje zdroj řádků. Při spuštění dotazu databázový server provede každý krok stromu a tím je zaručen správný postup provedení dotazu.

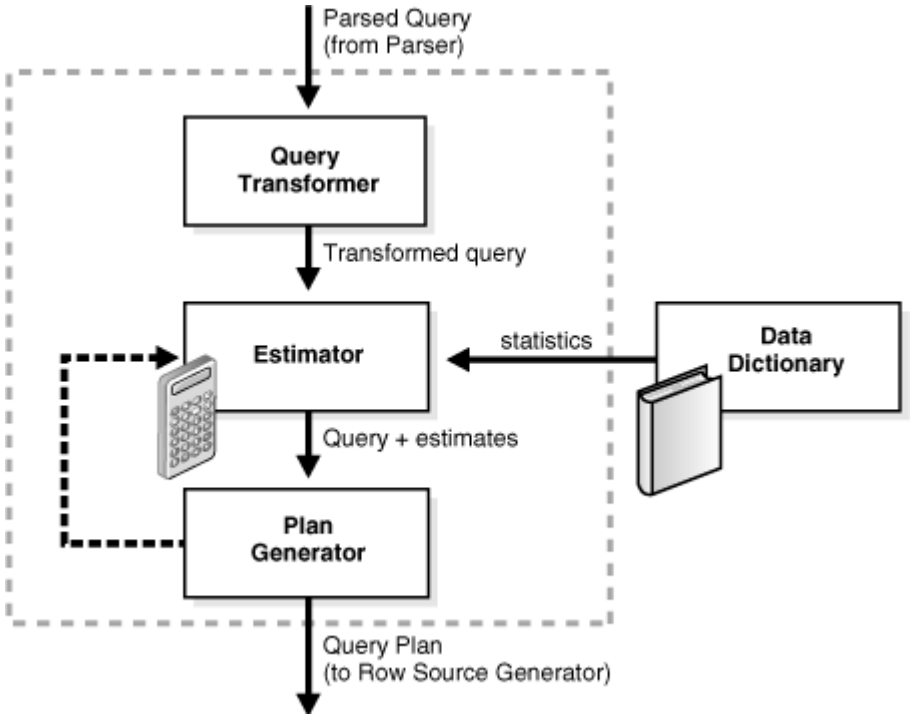
Fáze Optimalizace

Databázový systém Oracle podporuje dva druhy optimalizace:

- **rule-based** – plán se sestavuje pouze na základě tvaru dotazu, nezajímají ho data v tabulkách. Zastaralý způsob, který je udržován pouze pro zpětnou kompatibilitu.
- **cost-based** – optimalizace založená na využívání systémových prostředků v průběhu zpracování dotazu (CPU, paměť, vstupně/výstupní operace). Metoda využívá sesbírané statistiky o tabulkách a datech, které jsou uloženy v datovém slovníku (Data Dictionary).

Celou optimalizaci cost-based řídí optimalizátor (Query Optimizier). Optimalizátor je důležitou vestavěnou komponentou databázového systému Oracle. Určuje nejúčinnější metodu SQL dotazu pro přístup k požadovaným datům. Optimalizátor je složen ze tří částí:

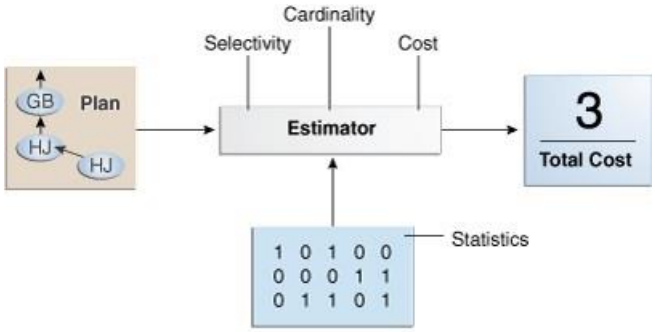
- Transformátor dotazu (Query Transformer)
- Odhadce (Estimator)
- Generátor plánů (Plan Generator)



Obrázek 2 – Optimalizátor (zdroj [19])

Transformátor dotazu rozhoduje, zda-li je výhodné přepsat originální příkaz, který by mohl mít nižší náklady. Optimalizátor transformuje dotazy pomocí technik, které má k dispozici.

Odhadce



Obrázek 3 – Odhadce (zdroj [19])

Odhadce odhaduje náklady každého exekučního plánu na základě statistik z datového slovníku. Odhadce je součástí optimalizátoru, který určuje celkové náklady exekučních plánů. K určení nákladů využívá tři odlišná měřítka:

- Selektivitu (Selectivity),
- Kardinalitu (Cardinality),
- Náklady (Cost).

Selektivita představuje procento řádků z množiny, které dotaz vybere. Množinou může být tabulka, pohled a výsledek spojení dvou a více tabulek. Selektivita úzce souvisí s predikátem dotazu **WHERE** last_name = ‘Novák‘, nebo kombinací predikátu (last_name = ‘Novák **AND** first_name = ‘David‘). Predikát vyfiltruje určitý počet řádků z množiny, které prošly predikátovou podmínkou. Výsledek selektivity je v rozmezí 0.0 až 1.0. Selektivita 0.0 znamená, že nejsou vybrány žádné řádky. Selektivita 1.0 znamená, že jsou vybrány všechny řádky. Hodnota selektivity ukazuje, jestli je predikát selektivní nebo méně selektivní (více neselektivní). Výpočet selektivity není viditelný v exekučních plánech. Optimalizátor odhaduje selektivitu na základě dostupnosti statistiky:

- Pokud jsou **statistiky nedostupné,** optimalizátor použije buď dynamické statistiky nebo interní výchozí hodnoty. Dynamické statistiky se použijí v závislosti na nastavené systémové hodnotě **OPTIMIZER_DYNAMIC_SAMPLING**. Databáze má nastavené pro typy predikátů různé výchozí hodnoty.
- Když jsou **dostupné statistiky,** odhadce použije k odhadu selektivity nasbírané statistické údaje.

Kardinalita udává počet řádků vrácených každou operací v exekučním plánu. V exekučním plánu je tato hodnota zobrazena ve sloupci řádky (Rows) viz Obrázek 4. Tento vstup je stěžejním bodem pro získání optimálního plánu, který je společný pro všechny nákladové funkce. Odhadce si může odvodit kardinalitu ze statistik tabulky, které lze získat při využití balíčku **DBMS_STATS**.

Náklady představují jednotku práce nebo použité zdroje. V exekučním plánu je hodnota nákladu zobrazena ve sloupci náklady (Cost) viz Obrázek 4. Optimalizátor využívá diskové vstupně/výstupní operace, procesor a paměť jako jednotky práce. Při odhadu nákladů zvažuje optimalizátor následující faktory:

- systémové zdroje, které zahrnují odhadované vstupně/výstupní operace, CPU a paměť,
- odhadovaný počet vrácených řádků (kardinalita),
- velikost vstupních množin,
- přístupové struktury.

Optimalizátor porovnává hodnoty celkových nákladů různých exekučních plánů pro stejný dotaz. Po nalezení nejnižší hodnoty nákladu je vybrán exekuční plán.

Exekuční plán

Exekuční plán databázového systému Oracle zobrazuje kombinaci kroků, které slouží k vykonání celého SQL příkazu. Exekuční plán zobrazuje celkové náklady dotazu na řádku 0 viz Obrázek 4. Na dalších řádcích jsou zobrazeny samostatné operace s jejich individuální hodnotou nákladu. Hodnoty jsou viditelné pouze v exekučním plánu a nelze je měnit.

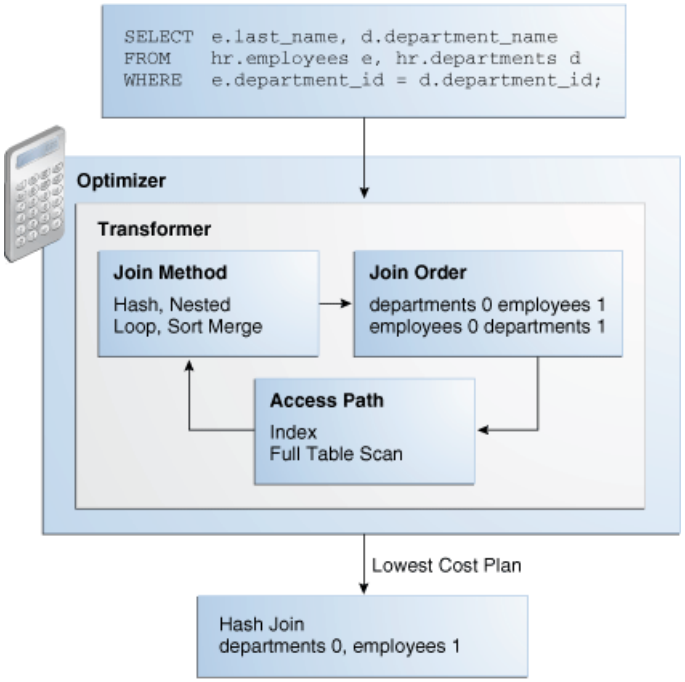
```
-------------------------------------------------------------------------------
| Id| Operation                    |Name             |Rows|Bytes|Cost %CPU|Time|
-------------------------------------------------------------------------------
| 0 | SELECT STATEMENT             |                 | 10| 250| 3 (0)| 00:00:01|
| 1 |  NESTED LOOPS                |                 |   |    |      |         |
| 2 |   NESTED LOOPS               |                 | 10| 250| 3 (0)| 00:00:01|
|*3 |    TABLE ACCESS FULL         |DEPARTMENTS      |  1|   7| 2 (0)| 00:00:01|
|*4 |    INDEX RANGE SCAN          |EMP_DEPARTMENT_IX| 10|    | 0 (0)| 00:00:01|
| 5 |   TABLE ACCESS BY INDEX ROWID|EMPLOYEES        | 10| 180| 1 (0)| 00:00:01|
-------------------------------------------------------------------------------
```

Obrázek 4 - Exekuční plán databázového systému Oracle (zdroj [19])

Generátor plánů

Generátor plánů se pokouší sestavit pro jeden dotaz různé varianty plánů, které po spuštění vracejí stejný výsledek. Varianty plánů vznikají zkoušením různých přístupových cest,

spojovacích metod a pořadí spojení. Při generování plánů je každý plán poslán odhadci, který nově navrhnutému plánu odhadne celkovou hodnotu nákladu. Po vygenerování všech možných exekučních plánů porovná optimalizátor mezi s sebou náklady a vybere plán s nejnižšími náklady.



Obrázek 5 - Generátor plánů databázového systému Oracle (zdroj [19])

Fáze generátoru zdroje řádků (SQL Row Source Generation)

V této části zpracování dotazu dojde k přeměně optimálního exekučního plánu na iterativní exekuční plán, který umí databáze zpracovat. Iterativní plán je v podstatě binární strom vracející množinu výsledků po provedení SQL příkazu. Plán je složen z kroků, kdy každý krok vrací řádky z množiny. Zdrojem řádku může být tabulka, pohled, výsledek spojení nebo operace seskupení. Ze stromu se dají vyčíst tyto informace:

- tabulky, na které se odkazuje dotaz,
- přístupová cesta pro každou použitou tabulku,
- metoda připojení tabulek ovlivněná operacemi spojení,
- datové operace jako je filtrování, řazení nebo agregace.

Fáze provedení dotazu

Během provádění dotazu provede databázový server každý krok stromu. Každý uzel stromu reprezentuje zdroj řádků. Každý uzel buď čte řádky přímo z databáze nebo je přijímá z jednoho

nebo více uzlů jako vstup. Spuštění stromu je v opačném pořadí, než uvádí exekuční plán, od dolní části směrem nahoru. K urychlení dotazu si databáze načte data z disku do paměti. [19]

3.1.2 Optimalizace SQL dotazu

Tento pododdíl vychází z dokumentace Oracle viz [19]. Jednou z možností optimalizace SQL dotazu je ovlivnění chování optimalizátoru, které lze dosáhnout pomocí inicializačních parametrů, hintů a správnou aktualizací statistik.

Změnou inicializačního parametru lze významně ovlivnit chování optimalizátoru. Hinty můžou ovlivnit: výběr přístupové cesty, pořadí spojování, metody spojování atd. Aktualizovanější statistiky poskytnou optimalizátoru lepší informace na odhad nákladů exekučního plánu.

Přístupové cesty

Full Table Scans

Optimalizátor volí Full Table Scans, když nelze najít jinou přístupovou cestu nebo jiná přístupová cesta má vyšší náklady. Mezi nejčastější důvody použití patří:

- neexistující indexy,
- dotaz je nevýběrový, bez vyhledávací podmínky,
- statistika tabulky je zastaralá,
- tabulka obsahuje málo dat,
- dotaz použil hint.

Table Access by Rowid

Zpravidla databázový systém Oracle přistupuje k tabulce přes sloupec ROWID prohledáním jednoho nebo více indexů. Pokud databáze použije přístup k tabulce přes ROWID, provede následující dva kroky:

- Prvním krokem získává ROWID vybrané řádky, buď pomocí klauzule **WHERE** nebo indexovým skenováním jednoho nebo více indexů.
- V druhém kroku vyhledá každý vybraný řádek v tabulce na základě hodnoty ROWID.

Index Unique Scans

Tento druh indexového prohledávání vrací právě jeden ROWID (záznam tabulky).

Optimalizátor použije Index Unique Scans, když je zadán predikát rovnosti. Pro použití tohoto skenování vyžaduje databázový systém Oracle následující podmínky:

- Predikát rovnosti musí odkazovat na všechny sloupce unikátního klíče pomocí operátoru rovnosti, např. **WHERE** driver_id = 5.
- Dotaz by měl obsahovat predikát rovnosti na sloupec, nad kterým je vytvořen unikátní index příkazem **CREATE UNIQUE INDEX**.

Skenování funguje na principu nalezení jedinečného zadaného klíče. Metoda skončí prohledáním indexu v momentě, kdy najde první totožný záznam. Další stejný záznam se v indexu nemůžu nacházet. Databáze obdrží ROWID z položky indexu a načte požadovaný řádek.

Index Range Scans

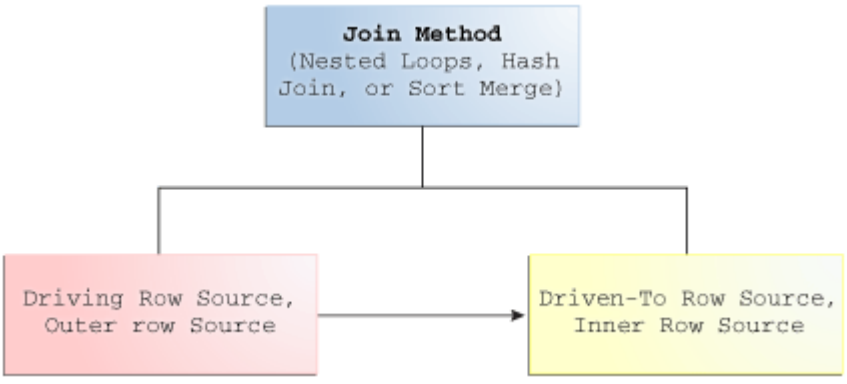
Index je obvykle čten ve vzestupném pořadí, ale může být použito i sestupné. Skenování indexu může vracet žádný, jeden, dva nebo více řádků. Optimalizátor obvykle zvažuje Index Range Scans, pokud podmínka dotazu obsahuje následující operátory =, <, >.

Index Fast Full Scans

Index Fast Full Scans čte indexové bloky v neseřazeném pořadí, jak jsou umístěny na disku. Jedná se o nahrazení Full Table Scans tím, že je prohledán celý index. Pokud dotaz přistupuje pouze ke sloupcům obsažených v indexu, tak optimalizátor zvolí toto skenování. Metoda neumožňuje vynechání operace řazení, protože indexové bloky jsou čteny v neseřazeném pořadí. K prohledání indexu je použito více blokové čtení.

Metody spojování

Optimalizátor vybírá metodu s nejnižšími odhadovanými náklady v závislosti na statistikách.



Obrázek 6 - Metody spojování v databázovém systému Oracle (zdroj [19])

Nested Loops Joins (NLJ)

Princip spočívá ve spojení vnější a vnitřní tabulky. Ke každému řádku z vnější tabulky je přiřazen odpovídající řádek z vnitřní tabulky. Databáze načte řádky z tabulek, které odpovídají predikátu spojení. K načtení řádků z vnitřní tabulky lze použít index pomocí ROWID, pokud je index k dispozici.

Optimalizátor zvažuje NLJ na základě:

- malé podmnožiny dat
- velké podmnožiny dat s nastaveným optimalizačním modem FIRST_ROWS,
- efektivního způsobu přístupu k vnitřní tabulce pomocí vhodně napsané podmínky spojení.

Optimalizátor se rozhoduje na základě počtu očekávaných řádků, nikoliv podle velikosti tabulky.

Hash Joins

Metoda je vhodná pro připojení větších tabulek. K sestavení tabulky hash použije optimalizátor menší tabulku. Vytvořený klíč pomocí funkce hash bude uložen v paměti, který určuje umístění řádku v tabulce hash. Databáze prohledává tabulku hash a větší tabulku k nalezení řádků, které odpovídají podmínce spojení. Optimalizátor volí pro metodu, když je požadováno spojit velké množství dat. Tabulka hash je umístěna v oblasti PGA, která umožňuje snížení logických vstupně/výstupních operací a tím dojde k vynechání nutnosti opakovaně blokovat a číst bloky ve vyrovnávací paměti.

Sort Merge Joins

Pokud nejsou tabulky setříděny, databáze setřídí tabulky operací SORT JOIN podle sloupce uvedeného ve spojení. Spojení tabulek provede operace MERGE JOIN. Pro každý řádek první tabulky databáze najde odpovídající řádek z druhé tabulky, podle shodné hodnoty. Databáze prohledává druhou tabulku do doby, než nalezne první neodpovídající řádek. Po nalezení neshodného řádku, databáze přejde na první tabulku a iteruje na další řádek, dokud nejsou spojeny všechny odpovídají řádky z první a druhé tabulky.

Optimalizátor volí Sort Merge Joins, když není spojení definováno přes podmínku rovnosti. Místo toho jsou použity následující operátory <, <=, >, >=. Metoda není příliš efektivní, jelikož se často musí prohledat a setřídit obě dvě vstupní množiny.

Partitioning

Oracle nabízí tři základní metody:

- range,
- hash,
- list.

Partition Pruning je varianta představující nejjednodušší a nejefektivnější prostředek ke zlepšení výkonu pomocí rozdělení. Je založena na principu rozdělení tabulek do menších částí podle definované podmínky. Díky tomu nabízí rychlejší prohledání dat v tabulce při paralelním přístupu.

Partition-Wise Joins využívá spojení tabulek k snížení množství dat, které se budou muset načítat k zpracování dotazu. Oracle rozdělí velké spojení na více malých spojení a tím se sníží celkový čas strávený nad spojením tabulek. Při zmíněné situaci bude efektivnější dotaz provádět pomocí paralelního zpracování.

3.1.3 Autentizace

Tento pododdíl vychází z dokumentace Oracle viz [20].Oracle umožňuje databázovou autentizaci pomocí těchto způsobů:

- databázová autentizace,
- autentizace prostředky operačního systému,
- síťová autentizace.

Databázová autentizace využívá tabulkový prostor **SYSTEM**, ve kterém jsou uloženy všechny potřebné informace pro autentizaci uživatele.

Autentizace prostředky operačního systému lze využít v případě, když byl uživatelský účet vytvořen s klauzulí IDENTIFIED EXTERNALLY a má nastavenou hodnotu OS_AUTHENT_PREFIX. Výchozí hodnota je OPS$.

Síťová autentizace je v podporována těmito technologiemi:

- SSL,
- Kerberos,
- PKI (Oracle Wallet Manager),
- RADIUS,
- Adresářová služba (Oracle Internet Directory).

Oracle ještě nabízí speciální postupy autentizace pro administrátory databáze, protože využívají některé speciální operace.

3.1.4 Autorizace

Tento pododdíl vychází z dokumentace Oracle viz [20]. Oracle nabízí řadu možností, jak spravovat uživatele, aby měli omezený přístup k datům a nemohli s nimi provádět veškeré operace. Pro usnadnění administrace zabezpečení dat jsou nabízeny tyto možnosti:

- uživatelské účty,
- profily,
- schémata,
- systémová oprávnění,
- objektová oprávnění,
- role.

3.2 Microsoft SQL Server

Úvodní část oddílu vychází z dokumentace Microsoft viz [21]. Microsoft SQL Server je vyvíjen společností Microsoft Corporation. Firma vyvíjí databázové systémy téměř třicet let. Z počátku společnost Microsoft vyvíjel MS SQL Server společně s firmou Sybase. První verzí, která byla vyvíjena pouze Microsoftem byla verze MS SQL Server 6.0 z roku 1995. Výhodou používaní MS SQL Serveru je úzká spolupráce s operačním systémem MS Windows. Nejnovější verze z roku 2016 a 2017 je doporučena pro řešení BI a využití pro technologii cloud computing.

Produkt je nabízen ve čtyřech edicích:

- Microsoft SQL Server 2016 **Enterprise** přináší komplexní funkce datových center pro špičkové aplikace s vysokým výkonem, neomezenou virtualizací a celosvětovou business inteligencí. Tato prémiová nabídka umožňuje vysokou úroveň služeb pro kritické pracovní úkoly a přístup koncových uživatelů k datům.
- Microsoft SQL Server 2016 **Standard** poskytuje základní správu dat a BI databázi pro oddělení a malé organizace. Je to z důvodu, aby mohli provozovat své aplikace a nabízet podporu společných vývojových nástrojů pro „*on-premise*“ a „*cloud*“. Umožní to těmto subjektům efektivní správu databáze s minimálními zdroji.
- Microsoft SQL Server 2016 **Express** je vhodná pro návrh a vývoj aplikací klasické pracovní plochy, webových a malých serverových aplikací. Edice je dostupná zdarma.

- Microsoft SQL Server 2016 **Developer** je plnohodnotná edice SQL Serveru 2016, která slouží k vývoji a testování aplikací v neprodukčním prostředí.

Microsoft SQL Server 2016 nabízí real-time provozní analýzu, bohatou vizualizaci mobilních zařízení, vestavěné rozšiřující analytické technologie (funkce), nové pokročilejší bezpečnostní technologie a nové hybridní cloud scénáře. Vývojářům je umožněno vytvářet programové rozhraní aplikací, které je možné upravovat napříč všemi edicemi.

Novinky (nové funkce):

- Columnstore Indexes,
- In-Memory OLTP,
- PolyBase,
- Stretch Database,
- Live Query Statistics,
- Query Store.

Novinkou u MS SQL Serveru je umístění cache v jádře. Je to výrazná výhoda oproti databázovému systému Oracle. Další odlišnou vlastností je, že má implementovanou vlastní nadstavbu SQL jazyka, která nese označení Transact-SQL (T-SQL).

3.2.1 Optimalizace SQL dotazu

Tento pododdíl vychází z dokumentace Microsoft viz [22].

Přístupové cesty

Table Scan načítá po jednom řádku celou tabulku.

Index Scan načítá celý index, ale nevyužívá výhody indexu jako takového.

Index Seek načítá jen ty řádky na základě vhodně napsané podmínky, která využívá výhody indexu. Tato operace je nejvhodnější pro načítání řádků.

ColumnStore Index Scan je nová metoda, která využívá výhody ColumnStore indexu.

Indexy

MS SQL Server nabízí několik typů indexů:

- Clustered Index,
- Non-Clustered Index,
- Primary XML Index,

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

Metody spojení

- Nested Loops Joins,
- Merge Joins,
- Hash Joins.

Materializované pohledy

MS SQL Server nepodporuje materializované pohledy.

Partitioning

Oficiálně MS SQL Server podporuje rozdělení tabulek metodou range partitioning.

3.2.2 Autentizace

Ověření uživatele k databázovému systému je umožněno dvěma způsoby. První způsob je ověření přes prostředky operačního systému Windows. Druhý způsob je prostřednictvím vytvořeného uživatelského účtu v MS SQL Serveru. Windows k ověřování používá protokol Kerberos poskytující vysoké zabezpečení hesla a zároveň jeho validaci. MS SQL Server lze ještě také autentizovat přes protokol LDAP. [23]

3.2.3 Autorizace

Tento pododdíl vychází z dokumentace Microsoft viz [24]. Správu uživatelů, omezený přístup k datům a administraci zabezpečení dat lze řídit následujícími možnostmi:

- login OS,
- databázový účet,
- schéma,
- role,
- oprávnění (systémová, objektová).

MS SQL Server má k dispozici tři druhy rolí:

- **Serverové role** jsou k dispozici pouze předdefinované. Nelze je měnit, přidat či smazat. Serverové role umožňují spravovat nastavení celého serveru, správu procesů, vytváření a změny databází atd.

- **Databázové role** jsou předdefinované, ale v každé databázi je lze zvlášť přidávat a mazat. Databázové role usnadňují správu uživatelů, přidání, změnu a odstraňování všech objektů v databázi a čtení, změnu a zápis dat do všech tabulek.
- **Aplikační role** vyžaduje heslo. Jsou určeny převážně pro aplikace, které svojí aplikační logikou řídí přístup k datům. Pro aplikační role platí pravidlo, že jednotlivý členové mají možnost přidělovat svoji roli ostatním uživatelům.

3.3 MySQL

Úvod oddílu vychází z dokumentace MySQL viz [25]. MySQL je celosvětově nejpopulárnější open source databáze. Prokazuje se osvědčenou výkonností, spolehlivostí a snadnou použitelností. MySQL je velmi populární z hlediska webových aplikací. Mezi webové aplikace, které používají MySQL patří Facebook, Twitter, YouTube, Google, a mnoho dalších. MySQL je vyvíjen, distribuován a podporován společností Oracle Corporation. Oracle se snaží o zlepšování MySQL přidáním nových funkcí k posílení následující generaci webových, cloudových, mobilních a vestavěných aplikací. Databázový systém je distribuován ve dvou verzích licence. První možnost je použít tento software jako Open Source pod pravidly GNU General Public License. Druhou možností je komerční placená licence od Oraclu.

Proprietární edice:

- MySQL **Enterprise** Edition
- MySQL **Standard** Edition
- MySQL **Classic** Edition
- MySQL **Cluster Carrier Grade** Edition

Svobodný software vyvíjený aktivní komunitou:

- MySQL **Community** Edition

MySQL je nákladově efektivní databáze poskytující spolehlivé a vysoce výkonné služby. Umožňuje škálovatelné e-commerce, zpracování online transakcí a vestavěné databázové aplikace. Jedná se o integrovanou, transakčně bezpečnou databázi podporující ACID. MySQL obsahuje kompletní sadu ovladačů a vizuálních nástrojů pro potřeby vývojářů k vytváření a správě aplikací.

Novinky (nové funkce):

- High Performance & Scalability,
- Self-healing Replication Clusters,

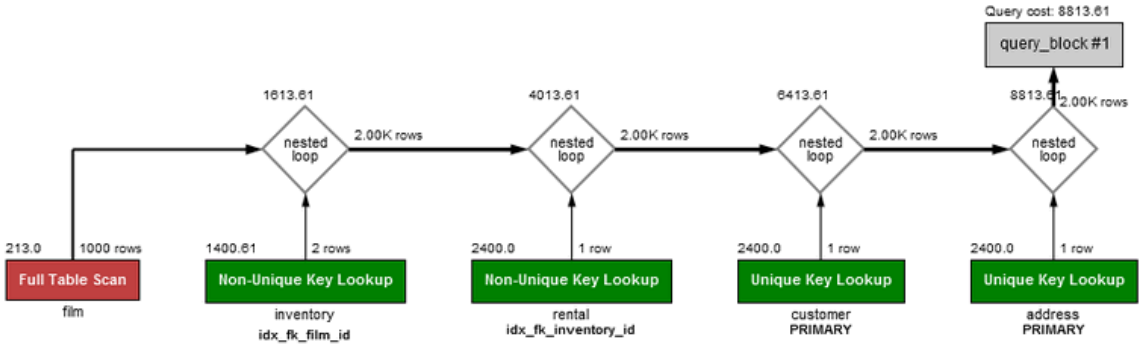
- Online Schema Changes,
- Performance Schema,
- SQL & NoSQL,
- Platform Independence,
- Big Data Interoperability.

3.3.1 Optimalizace SQL dotazu

Tento pododdíl vychází z dokumentace MySQL viz [25].

Přístupové cesty

MySQL má pro přístupové cesty dvojí označení. První označení je systémové běžné, které se používá všude v dokumentaci a druhé označení se vztahuje k vizuálnímu exekučnímu plánu.



Obrázek 7 - Vizuální exekuční plán databázového systému MySQL (zdroj [25])

EQ_REF (Unique Key Lookup) je operace s nízkou cenou nákladu. Optimalizátor je schopný najít index, který může použít k získání požadovaných záznamů. Tím může být operace rychlá, protože index vyhledá přímo na stránce, kde se nacházejí všechny řádky (data).

REF (Non-Unique Key Lookup) je operace, která může mít nízkou nebo střední cenu nákladu. Nízkou, pokud se shoduje malý počet řádků. Vysokou, když se zvyšuje počet řádků, které se shodují.

EQ_REF a REF mají zelenou barvu buňky ve vizuálním exekučním plánu.

RANGE (Index Range Scan) je operace se střední cenou nákladu. Operace prohledává část indexu. RANGE má oranžovou barvu buňky ve vizuálním exekučním plánu.

INDEX (Full Index Scan) je operace s vysokou cenou nákladu, obzvláště pro velké indexy.

ALL (Full Table Scan) je operace s velmi vysokou cenou nákladu, zejména pro velké tabulky, ale méně pro malé tabulky. Optimalizátor nenalezl použitelný index pro tabulku, proto musí prohledat každý řádek tabulky. Ve druhé variantě je rozsah vyhledávání velmi široký a použití indexu by bylo zbytečné.

INDEX a ALL mají červenou barvu buňky ve vizuálním exekučním plánu.

Metody spojování

Ke spojování tabulek je vždy použita metoda Nested-Loop Join.

Indexy

Typy indexů, které využívají strukturu B-tree k uložení dat jsou následující:

- PRIMARY KEY,
- FOREIGN KEY,
- UNIQUE,
- FULLTEXT.

Prostorové indexy (spatial indexes) využívají k uložení dat strukturu R-tree. Tuto funkcionalitu umožňuje využívat úložiště InnoDB a MyISAM.

Materializované pohledy

MySQL nepodporuje materializované pohledy.

Partitioning

V současné době MySQL podporuje partitioning na úložištích InnoDB a NDB. MySQL podporuje následující typy rozdělování tabulek:

- RANGE,
- LIST,
- COLUMNS (RANGE, LIST),
- HASH,
- KEY,
- Subpartitioning.

3.3.2 Autentizace

Tento pododdíl vychází z dokumentace MySQL viz [25]. MySQL má uložené potřebné údaje k ověřování klienta v tabulce mysql.user. Uživateli je dovolen přístup na základě identifikační

trojice, kterou tvoří uživatelský účet, heslo a host. Host představuje název počítače, IP adresa nebo localhost. MySQL nabízí dvě odlišné funkce autentizace:

- **Externí autentizace** je vhodná pro externí autentizační metody, které nevyužívají tabulku mysql.user.
- **Uživatelé proxy** vystupují pro MySQL jako jiní uživatelé. Například: uživatelský účet David je externí uživatel, který bude mít přiřazenou totožnost od Jakuba. Jakub je proxy uživatel s uživatelským účtem. David vystupuje pod oprávněnými, která má přiřazená od Jakubova uživatelského účtu.

Autentizaci je možné provádět několika metodami:

- **Nativní autentizace** využívající hašování hesla.
- **Šifrování hesla** pomocí **SHA-256**.
- **Autentizace na straně klienta** bez úpravy hesla.
- **Externí autentizace** pomocí PAM, LDAP, OS Windows.
- **Peer autentizace** prostřednictvím socketů.

3.3.3 Autorizace

Oprávnění nabízená v MySQL jsou odlišná podle úrovní operací:

- **Administrativní oprávnění** poskytuje uživatelům řídit provoz serveru MySQL, zpravidla se jedná o globální oprávnění.
- **Databázová oprávnění** jsou vztažena na vybranou databázi a na její obsah, tj. všechny objekty.
- **Oprávnění pro databázové objekty** jako jsou tabulky, pohledy atd. Možnost přidělit všem objektům daného typu oprávnění.

MySQL server kontroluje dostatek oprávnění pomocí uložených grantových tabulek v paměti. Grantové tabulky se nacházejí v databázi mysql. Oprávnění tedy pocházejí z tabulek USER, DB, TABLES_PRIV, COLUMNS_PRIV a PROCS_PRIV. MySQL nemá implementované objekty typu role, profil, skupinu uživatelů.[25]

3.4 PostgreSQL

PostgreSQL je objektově-relační databázový systém spadající do skupiny softwaru Open Source. PostgreSQL byl původně projekt Kalifornské Univerzity v Berkeley. Tento databázový systém nabízí velkou podporu nejrůznějšími operačními systémy. Liberální licence umožňuje komukoliv kód PostgreSQL použít, změnit nebo distribuovat bezplatně jakkoliv ať už pro

soukromé obchodní nebo akademické účely. Výhodou tohoto databázového systému je jeho komunita, která se ho snaží vylepšit a díky tomu může nabídnout mnohem lepší informace než technické dokumentace proprietárních databázových systémů. [26]

Poslední vydaná verze má označení:

- **PostgreSQL 10.3.**

Novinky (nové funkce) přinášejí verze 9.6 a 10 dle [27]:

- declarative table partitioning (10),
- frozen page map (9.6),
- parallel bitmap heap scans (10),
- parallel B-tree index scans (10),
- parallel JOIN, aggregate (9.6),
- parallel merge joins (10),
- parallel query (9.6),
- parallel seqscan (9.6).

3.4.1 Optimalizace SQL dotazu

Tento pododdíl vychází z dokumentace PostgreSQL viz [28].

Přístupové cesty

Seq Scan načítá po jednom řádku celou tabulku.

Index Scan prohledává index střídavě, skáče mezi indexem a tabulkou. Je to z důvodu vyhledání řádku v indexu a načtení řádku z tabulky. Nevýhodou střídavého načítání je velký počet náhodných vstupně/výstupních operací.

Index Only Scan načítá data pouze z vybraného indexu, nikoliv z tabulky.

Bitmap Index Scan a Bitmap Heap Scan jsou dvě spolupracující operace. Bitmap Index Scan použije indexové skenování na nalezení požadovaných řádků, které mají být použity. Tyto řádky skutečně načte Bitmap Heap Scan, který obdržel informace, o jaký index se jedná.

Parallel Seq Scan přistupuje k tabulce sekvenčně, ale jednotlivé tabulkové bloky jsou rozděleny mezi spolupracující procesy.

Parallel Bitmap Heap Scan vybere jeden z procesů jako hlavní proces, který provede prohledání jednoho nebo více indexů. Hlavní proces vytvoří bitovou mapu, která označuje

tabulkové bloky, které je potřeba navštívit. Označené bloky jsou rozděleny mezi spolupracující procesy stejně jako v Parallel Seq Scan. To znamená, že Heap Scan je prováděn paralelně, ale základní skenování indexu nikoliv.

Parallel Index Scan a Parallel Index-Only Scan funguje na principu, kdy spolupracující procesy přebírají data z indexu. V současné době PostgreSQL podporuje Parallel Index Scans pouze pro B-tree indexy. Každému procesoru bude přiřazen jeden indexový blok, který bude prohlížet. Proces vrátí všechny n-tice, na které odkazuje daný blok. Jiný proces může současně vracet n-tice z jiného indexového bloku. Výsledky paralelního skenování jsou vráceny v seřazeném pořadí v rámci každého pracovního procesu.

Indexy

Ve výchozím nastavení PostgreSQL ukládá hodnoty indexu do stromové struktury B-tree, která patří mezi nejčastěji používanými. Databázový systém nabízí několik druhů indexů:

- B-tree
- Hash,
- GiST,
- SP-GiST,
- GIN,
- BRIN.

Metody spojování

Nested Loop Join provede spojení tím, že pro pravou tabulku je naskenován každý řádek nalezený v levé tabulce. Tato strategie je jednoduchá, ale je velmi časově náročná. Pokud lze pravou tabulku prohledat pomocí indexového skenování, může to být výhodná strategie.

Merge Join před spojením provede setřídění tabulek podle spojovaného sloupce. Potom jsou setříděné tabulky naskenovány paralelně. Výsledné řádky spojení jsou vytvořeny podle definované podmínky. Tento typ spojení je efektivnější, jelikož je skenování tabulek provedeno pouze jednou.

Hash Join nejprve naskenuje pravou tabulku, kterou načte do hash tabulky. Definovaný sloupec v podmínce spojení je použit jako klíč hash. Dalším krokem je skenování levé tabulky. Příslušné hodnoty každého nalezeného řádku levé tabulky jsou použity jako hash klíče k vyhledání odpovídajících řádků v hash tabulce.

Materializované pohledy

PostgreSQL podporuje materializované pohledy od verze 9.3.

Partitioning

PostgreSQL nabízí dvě vestavěné možnosti rozdělení tabulek:

- range,
- list.

Tyto dvě možnosti lze implementovat pomocí deklarativního rozdělení, které mají několik výkonnostních výhod. Oproti tomu rozdělení tabulky pomocí dědičnosti, pohledů atd. nemá tak velký vliv na výkon databázového systému PostgreSQL.

3.4.2 Autentizace

Tento pododdíl vychází z dokumentace PostgreSQL viz [28]. PostgreSQL poskytuje širokou řadu odlišných způsobů autentizací klienta. Ověření klienta je řízeno konfiguračním souborem pg_hba.conf, který je uložen v adresáři databáze. V konfiguračním souboru lze nastavit různé parametry jako jsou autentizační metody:

- **Autentizace důvěryhodnosti** je vhodná pouze v případě, že existuje odpovídající ochrana na úrovni operačního systému.
- **Autentizace heslem** je ověřování založené na funkci MD5.
- **GSSAPI** je založena na podpoře protokolu GSSAPI s ověřováním podle protokolu KERBEROS.
- **SSPI** je technologie od MS Windows pro bezpečnou autentizaci s jediným přihlášením, využívající protokol KERBEROS.
- **Autentizace adres** je podporována pouze u připojení TCP/IP.
- **Peer autentizace** získává uživatelské jméno z jádra operačního systému klienta. Metoda je vhodná pouze pro místní připojení.
- **LDAP, KERBEROS.**
- **Autentizace prostředky certifikátu** je založena na připojení pomocí SSL.
- **Autentizace PAM** je metoda, v které vystupuje PAM jako autentizační mechanismus.
- **Autentizace BSD** je podporována pouze pro operační systém OpenBSD.

3.4.3 Autorizace

PostgreSQL pomocí konceptu rolí řídí oprávnění přístupu k databázi. Role představují uživatelské účty databáze nebo skupinu rolí v závislosti, jak je role nastavena. Role mají

možnost vlastnit databázové objekty a přiřazovat oprávnění k těmto objektům pomocí jiných rolí. Všechny informace o řízení přístupu jsou uchovány v datových slovníkách:

- pg_roles,
- information_schema,
- role_table_grants,
- pg_auth_members.

PostgreSQL poskytuje řízení přístupu na základě schématu a oproti Oraclu nepodporuje profily.[28]

3.5 Srovnání

Společnost Gartner patří mezi uznávané společnosti, která publikuje nejrůznější články v oblasti informačních technologií. Články společnosti vznikají během dlouhodobého výzkumu. Společnost Gartner publikovala v roce 2017 článek, ve kterém porovnává trh systémů pro správu databází. Gartner popisuje trh zahrnující relační a nerelační produkty SŘBD vhodné pro širokou škálu transakčních aplikací na podnikové úrovni. Dále trh obsahuje produkty pro ERP, CRM, IoT atd. Gartner popisuje definici SŘBD jako plnohodnotný softwarový systém využívaný k definování, vytváření, správě a aktualizaci databáze. Společnost Gartner uvádí předpoklad, že relační technologie budou do roku 2020 nadále využívány pro nejméně 70 % nových aplikací a projektů. V článku jsou detailněji rozebrány produkty společností Microsoft a Oracle. Datatabázové systémy MySQL a PostgreSQL nejsou obsahem článku, jelikož nesplňují kritéria viz [29].

3.5.1 Microsoft

Silné stránky

V roce 2016 zaznamenal Gartner u společnosti Microsoft největší tržby ze všech dodavatelů na trhu. Microsoft nabízí šikovně produkt SQL Serveru bezplatně ve vývojářské edici. Společnost patří mezi největší průkopníky nabízející víceúčelové SQL pro IMDBMS. Zákazníci společnosti vyjadřují spokojenost z hlediska celkové zkušenosti, požadavků na uspokojování potřeb, poměru mezi kvalitou a cenou. Mají dobré zkušenosti s vyjednáváním, integrací a nasazením, službami a podporou.

Slabé stránky

Společnost Microsoft nezískala vysoké referenční skóre za cenové nabídky z hlediska cloudu jako společnosti Google a AWS. Stále více zákazníků dává přednost alternativám před

Microsoftem v oblasti cloudu. Partnerům společnosti Microsoft se nevedlo tak dobře s nasazením cloudu jako partnerům od AWS. Některé nástroje zaostávají za jinými dodavateli, především vývojové nástroje. Referenční zákazníci nebyli spokojeni s cenovými metodami společnosti Microsoft. Jednalo se o případy, kdy nebyl dobře vysvětlen účel produktu.

Pro zobrazení dodavatelů jsou v magickém kvadrantu vyhodnoceny všechny jeho produkty jako jedna entita [29].

Obrázek 8 - Magický kvadrant dodavatelů SŘBD (zdroj [29])

Kvadrant Vedoucí (LEADERS) představuje lídry společností SŘBD viz

Obrázek 8. Vedoucí společnosti se prokazují podporou široké škály nejrůznějších aplikací. Aplikace jsou založené na podpoře mnoha různých datových typů a mají víceúčelové použití. Společnosti se můžou těšit z trvalé zákaznické spokojenosti a silné zákaznické podpory. Mnoho z nich si vybudovalo za několik desetiletí svůj vlastní partnerský ekosystém. Z toho lze usoudit, že tyto společnosti představují pro své zákazníky nejnižší riziko v oblasti výkonu, škálovatelnosti, spolehlivosti a podpory. S měnícím se trhem mohou vedoucí společnosti prosazovat silné vize, které aktuálně podporují trh, ale i vznikající nové trendy. [29]

3.5.2 Oracle

Silné stránky

Oracle pracuje neustále na rozšíření svého veřejného cloudového produktu, který zahrnuje Iaas, Pass a Saas. Společnost nabízí nejnovější programové rozhraní API a podporu odlišným datovým typům. Tři čtvrtiny referenčních zákazníků společnosti Oracle využívá databázi Oracle více než 10 let. V ostatních průzkumech a v průzkumech společnosti Gartner jsou nejčastěji zmíněnými přednostmi Oraclu: výkon, funkce a spolehlivost produktů. Referenční zákazníci hodnotili společnost Oracle jako nadprůměrnou pro úplnou spokojenost s produkty, kvalitní školení pro koncového uživatele a výbornou dokumentaci. Společnost získala dobré hodnocení za bezpečnostní prvky ve svých řešeních.

Slabé stránky

Společnost Oracle s cílem navýšení a zlepšení výkonu zvýšila počet licencí per procesor kvůli stávajícím a budoucím zákazníkům cloudu, což zdvojnásobilo náklady na provozování Oracle. Více referenčních zákazníků uvedlo komplikované licencování a složité jednání, které stále pokračuje. Někteří referenční zákazníci popsali velké a složité úsilí se získáním odpovídající podpory ze strany společnosti Oracle. Zákazníci si stěžují na počet oprav a s nimi související zdlouhavé procesy nasazení. Společnost zareagovala zavedením čtvrtletních aktualizací s cílem zjednodušit opravy.

4 PRAKTICKÁ ČÁST

4.1 Testovací aplikace DbFit

DbFit je open source testovací databázový nástroj, jenž slouží k testování SQL příkazů. Podporuje širokou škálu databázových serverů, mezi které patří Oracle, MS SQL Server, MySQL, PostgreSQL atd. Výhodou této aplikace je psaní, řízení a spouštění testů z webového prohlížeče. Aplikace byla zvolena pro intuitivní ovládání a jednoduché nastavení z hlediska připojení na jednotlivé databázové servery.

4.2 Hardwarové prostředky

Praktická část byla prováděna na laptopu Acer Aspire F5-573G s následujícími parametry:

Tabulka 2 – Specifikace hardwaru (zdroj vlastní)

Parametr	Acer Aspire F5-573G
Procesor	Intel® Core™ i5-7200U CPU @ 2.50GHz
Paměť	8 GB DDR4
Pevný disk 1	LITEON CV3-8D128 SATA 128GB M.2 SSD
Pevný disk 2	TOSHIBA MQ01ABD100 1TB 2,5 HDD
Grafická karta	NVIDIA® GeForce® GTX 950M
Operační systém	Microsoft Windows 10 Home 64bitový

Na SSD disku Liteon je nainstalován OS, testované databázové servery společně s programy DbFit, System explorer.

4.3 Testované databázové systémy

- Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production
- MySQL Community Server (GPL) 5.7.21-log, innodb version 5.7.21, x86 64, Win64
- Microsoft SQL Server 2016 (SP1) (KB3182545) - 13.0.4001.0 (X64), Copyright (c) Microsoft Corporation Enterprise Edition (64-bit)
- PostgreSQL 10.1, compiled by Visual C++ build 1800, 64-bit

4.4 Schéma testovaného databázového modelu

Databázový model obsahuje tabulky, které představují důležité objekty ze světa Formule 1. V databázovém modelu se nacházejí objekty:

- jezdci,
- týmy,

- závodní okruhy,
- závody,
- adresy,
- země,
- národnosti,
- výsledky závodů,
- kvalifikace,
- nejrychlejší kola,
- tréninky.

Databázový model byl vytvořen ve vývojovém nástroji Oracle SQL Developer Data Modeler, který vyvinula společnost Oracle Corporation. Nástroj nabízí využití pro více databázových systémů. Prvotní databázový model byl vytvořen pro databázový systém Oracle, jehož schéma je znázorněno v Příloze A. Přetransformování modelu do ostatních databázových systémů, které jsou použity v této práci, bylo dosaženo pomocí programu SQLines Studio. SQLines Studio patří mezi OSS. Nástroj je dostupný jako online aplikace na webové stránce [30] nebo v desktopové verzi aplikace, kterou lze zadarmo stáhnout. Změna databázových modelů byla provedena využitím jazyka SQL, který databázové modely vytváří. Tento druh SQL příkazů se označuje DDL.

Tabulky

Databázový model je reprezentován několika tabulkami, které uchovávají různé informace. Z tabulek lze zjistit jména závodníka, počet týmů za které jezdil, jakých závodů se účastnil, jaké měl výsledky v trénincích, kvalifikaci, závodu a zda zajel nejrychlejší kola závodu.

Datové typy

V následující tabulce jsou uvedeny změny datových typů pro jednotlivé databázové systémy oproti databázovému modelu Oracle:

Tabulka 3 – Změna datových typů (zdroj vlastní)

Oracle	NUMBER(5)	NUMBER(10)	NUMBER(2)	NUMBER(6,3)
…	↓	↓	↓	↓
MS SQL Server	INT	BIGINT	SMALLINT	DECIMAL(6,3)
MySQL	INT(5)	BIGINT(10)	TINYINT(2)	DECIMAL(6,3)
PostgreSQL	INT	BIGINT	SMALLINT	DECIMAL(6,3)

Indexy

Při sestavení celého databázového modelu, tj. všech tabulek jsou vytvořeny indexy u každé tabulky. Indexy si každý testovaný databázový systém vytvořil implicitně. Každá tabulka má vytvořený index nad sloupci primárního klíče. U databázového systému MySQL má každá tabulka vytvořený index i nad sloupci cizích klíčů.

4.5 Testovaná data

Testovaná data obsahují informace z oblasti Formule 1. Některá použitá data jsou skutečná.

V tabulce JEZDCI jsou uložena skutečná celá jména jezdců a jejich státní příslušnost, která byla čerpána viz [31]. Tabulka VYSLEDKY, která v sobě ukrývá kalendáře závodů F1, obsahuje skutečné termíny závodů pouze pro ročník 2017 viz [32]. Tabulka TYMY má uložena skutečná celá data ze stránek [32] a [33]. Tabulky VYSLEDKY_ZAVODU, KVALIFIKACE, NEJRYCHLEJSI_KOLA a TRENINKY mají zahrnuta téměř všechna data z oficiální stránek Formule 1 viz [33]. Tabulky OKRUHY, ZAVODY a ADRESY obsahují všechna skutečná data z oficiálních stránek Formule 1 viz [33]. V tabulce ADRESY jsou uloženy skutečná data pro okruhy a týmy za rok 2017.

Ostatní tabulky mají vygenerována fiktivní data pomocí napsané aplikace v programovacím jazyku JAVA. Některá fiktivní data pocházejí ze skutečných dat, která byla duplikována za pomocí přidaného jedinečného identifikátoru. Důležitá vygenerovaná data jsou v tabulkách VYSLEDKY_ZAVODU, KVALIFIKACE, NEJRYCHLEJSI_KOLA, TRENINKY, která se převážně využívá v SQL dotazech. Prvních 400 řádků vycházelo ze skutečných dat, která byla následně rozšířena v každé tabulce do počtu 960 000 řádků. Data byla navýšena pro potřebu vysoce zatížit testované databázové systémy. Pokud se podaří zatížit databázový systém, je tu předpoklad zatížení hardwaru na kterém je nainstalován databázový systém.

4.6 Způsob testování

Testování probíhá spuštěním SQL dotazů, které lze spustit na vybraných databázových systémech. Prvním sledovaným parametrem je doba provedení dotazu pomocí nástroje DbFit. Dalšími sledovanými parametry je průměrné a maximální využití CPU. Průměrné využití CPU je bráno za celý čas spuštěného dotazu. Maximální využití CPU znázorňuje maximální vytížení CPU za dobu spuštěného dotazu. Tyto sledované parametry jsou zaznamenávány z programu System Explorer.

DbFit

Každý test SQL dotazu představuje webovou stránku, kde je uložen SQL dotaz a jeho výsledné řádky. Je velmi důležité, aby uložené výsledné řádky byly seřazeny přesně ve stejném pořadí, ve kterém je vrátí úspěšně spuštěný SQL dotaz. Z tohoto důvodu nebude ovlivněna doba provedení dotazu. To znamená, že DbFit nejprve provede dotaz a poté zkontroluje výsledné řádky dotazu s očekávanými uloženými výsledky na webové stránce.

Zda dotaz proběhl v pořádku, poznáme podle zeleného pozadí, které má řádek informující o správném výsledku dotazu. Řádek informuje o správném provedení dotazu, době provedení dotazu a počtu hodnot, které byly správně zkontrolovány. Počet zkontrolovaných hodnot je vypočítán vynásobením počtu řádků a sloupců. (To znamená, že pokud výsledek dotazu má mít deset řádků a patnáct sloupců výsledný počet hodnot musí být celkově sto padesát.) Zelené pozadí mají ještě uložené všechny řádky, které představují výsledné řádky SQL dotazu.

System Explorer

System Explorer zobrazuje využití hardwarových prostředků. Využití hardwarových prostředků je sledováno přes procesy jednotlivých programů. System Explorer je velice podobný programu Správci úloh z OS Windows.

4.7 Podmínky testování

Při testování dotazů, které byly spuštěny v databázové aplikaci DbFit, byl dále spuštěn pouze program System Explorer. Program byl použit ke sledování procesů jednotlivých databázových serverů s ohledem na využití procesoru (CPU). K dosažení výsledků z programu System Explorer byl použit program dropbox, který umožňuje stisknutím klávesy PrtSc uložit výsledný snímek obrazovky na paměťové médium.

4.8 Univerzální SQL dotazy

Univerzální SQL dotazy jsou dotazy, které lze spustit na každém databázovém serveru bez jakýchkoliv úprav. Využívá se standard jazyka SQL. Každý databázový server podporuje odlišnou verzi SQL standardu, ale nemusí podporovat všechny prvky daného SQL standardu.

Pro všechny univerzální dotazy je použita klauzule INNER JOIN, která slouží k vnitřnímu spojení tabulek. Klauzule INNER JOIN je podporována každým databázovým serverem, který je použit v bakalářské práci.

V každém univerzálním dotazu je každá tabulka přejmenována. Přejmenování je provedeno zkrácením názvu celé tabulky pomocí dvou písmen. Zkratka tabulky představuje odlišnou instanci dané tabulky:

Tabulka 4 - Seznam zkratek k jednotlivým tabulkám (zdroj vlastní)

Název tabulky	Zkrácený název tabulky
ADRESY	AA
JEZDCI	JJ
KVALIFIKACE	KK
NARODNOSTI	NN
NEJRYCHEJSI_KOLA	NK
PORADI	PP
PSC	Tabulka není v žádném dotazu použita.
STARTOVNI_LISTINY	SL
TRENINKY	Tabulka není v žádném dotazu použita.
TYMY	TT
VYSLEDKY	VV
VYSLEDKY_ZAVODU	VZ
ZAVODNI_OKRUHY	ZO
ZAVODY	ZZ
ZEME	ZE

Pokud je v dotazu použit vnořený dotaz, musíme odlišit názvy pomocí odlišných zkratek. Odlišení zkrácených názvů tabulek je provedeno pomocí číslic například: SL1, SL2, SL3 atd.

Popis každého univerzálního dotazu je rozdělen na dvě části. V první části je popsán univerzální dotaz. Ve druhé části je popsán výsledek. To znamená, co dotaz zobrazuje.

V průběhu psaní univerzálních SQL dotazů bylo zjištěno, že databázový systém MS SQL Server 2016 zpracovává vytvořené dotazy několikanásobně rychleji než ostatní databázové systémy. Z tohoto důvodu byly univerzální SQL dotazy přepsány, aby alespoň doba provedení dotazů přesáhla na MS SQL Serveru 10 sekund. Jedná se o dotazy č. 1, 2, 7 a 8. Tato podmínka se negativně projevila na ostatních databázových systémech s ohledem na dobu provedení dotazu.

4.8.1 Univerzální SQL dotaz č. 1

Dotaz č. 1 je složen z několika vnořených dotazů viz Příloha B. Základním zdrojem veškerých sloupců pro provedení dotazu je vnořený dotaz (VD1), který představuje pojmenovanou tabulku VYPOCET_VYHRANEHO_ZAVODU. VD1 vyfiltruje výsledné řádky podle sloupce CAS_ODSTUP. Podmínkou projdou jen ty řádky, které jsou vítězi závodu. Po ukončení VD1 se dotaz dostane k dalšímu vnořenému (VD2), který představuje pojmenovanou tabulku VYPOCET_VYHRANE_KVALIFIKACE. VD2 filtruje výsledné řádky z VD1 podle sloupce Q3. Podmínkou projdou pouze řádky, které mají zahrnuty nejrychlejší čas kvalifikace. Po ukončení VD2 se dotaz posune k provedení posledního vnořeného dotazu (VD3), který představuje pojmenovanou tabulku VYPOCET_NEJRYCHLEJSI_KOLA. VD3 filtruje výsledné řádky z VD2 podle sloupce CAS. Podmínkou projdou pouze řádky, které obsahují nejrychlejší čas závodu. Po ukončení VD3 se provede nadřazený dotaz (ND), který je finálním dotazem. ND vybere sloupce ROK, JEZDEC, TYM, MOTOR, SASI, ZAVOD, OKRUH z tabulky VYPOCET_NEJRYCHLEJSI_KOLA. Dotaz nebude snižovat výsledný počet řádků, jelikož restrikci provedly vnořené dotazy. Na závěr v ND je použita klauzule ORDER BY, která seřadí všechny výsledné řádky vzestupně podle sloupce ROK.

Výsledek dotazu zobrazí jezdce, kteří během závodního víkendu splnili tři důležité podmínky. Vyhráli závod, kvalifikaci a dosáhli v závodě nejrychlejšího kola. Na každém řádku je zobrazený rok závodu, celé jméno jezdce, název týmu, motoru a šasi. Poslední dva sloupce zobrazují název závodu a okruhu. Jezdec se může vyskytnout v seznamu vícekrát, protože mohl vyhrát více závodů za těchto podmínek.

První verze dotazu obsahovala jeden vnořený dotaz a druhý nadřazený dotaz. Ve vnořeném dotazu byly umístěny tři korelované dotazy v klauzuli WHERE. Dotaz musel být upraven do nynější podoby, jelikož původní dotaz by trval minimálně 100 hodin na databázových systémech MySQL a PostgreSQL.

4.8.2 Univerzální SQL dotaz č. 2

Dotaz obsahuje vnořený dotaz (VD) a nadřazený dotaz (ND) viz Příloha C. Ve VD je dvacet vnořených téměř identických korelovaných dotazů. Výsledné hodnoty každého korelovaného dotazu jsou zjištěny na základě vstupních sloupců. V tomto případě máme dva druhy vstupních sloupců. První druh sloupců je zadán ručně, kde se vyhodnocuje, zdali jezdec daný závod vyhrál a jaký závod jel. Druhý druh sloupců pochází z nadřazeného dotazu v tomto případě VD a tyto sloupce rozhodují pro jakého závodníka požadujeme výslednou hodnotu. Výsledné řádky VD

jsou použity k zobrazení v ND. ND zobrazuje výsledné řádky ve vzestupném pořadí podle sloupce JEZDEC.

Dotaz zobrazí u každého jednotlivého jezdce Formule 1 celé jméno a počet vítězství podle jednotlivých závodů Formule 1.

4.8.3 Univerzální SQL dotaz č. 3

Dotaz je složen z nadřazeného dotazu (ND) a několika vnořených dotazů, viz Příloha D. Tabulku HLAVNI_VYPOCET vzniká provedením vnořeného dotazu (VD). VD prohledá všechny výsledky jezdců a na základě těchto informací zjistí jejich statistiky. Hodnoty sloupců VYHRANA_KVALIFIKACE a NEJRYCHLEJSI_KOLO jsou zjištěny na základě spuštění korelovaných dotazů, které se provádějí pro každého jezdce podle vstupních sloupců z VD dotazu. Ostatní sloupce jsou zjištěny za použití agregačních funkcí s klauzulí CASE WHEN. Po provedení celého VD, kdy dostáváme všechny řádky pro tabulku HLAVNI_VYPOCET se spustí ND. ND zobrazí specifické sloupce. Na závěr jsou všechny řádky seřazeny vzestupně podle sloupce JEZDEC.

Dotaz zobrazí u každého jednotlivého jezdce Formule 1 celé jméno, národnost a různé statistiky. Mezi statistiky patří počet vyhraných kvalifikací, počet nejrychlejších kol, počet umístění na první, druhé a třetí pozici. Dále celkový počet pódiových umístění, všechny bodované, dokončené a nedokončené závody. Skutečný počet absolvovaných kol, závodů a sezón. Celkový počet získaných bodů.

U původního dotazu nelze zobrazit statistiky, které jsou v procentech. Hlavním důvodem byl MS SQL Server, který nedokázal zobrazit po dělení sloupců výsledky. Důvodem je, že MS SQL Server nedokáže zobrazit výsledek při operaci dělení dvou celočíselných míst. Řešení lze dosáhnout pomocí explicitního přetypování jednoho celočíselného datového typu na číslo s desetinou čárkou. Tento fakt neumožňuje vytvořit univerzální dotaz.

4.8.4 Univerzální SQL dotaz č. 4

Dotaz je složen z vnořeného dotazu (VD) a nadřazeného dotazu (ND), viz Příloha E. VD prohledá všechny výsledky jezdců ze třetí části kvalifikace. Korelovaný dotaz (KD) umístěný vně VD spočítá každému jezdci jednotlivé umístění kvalifikace, které uloží do sloupce STARTOVNI_POZICE. ND zpracuje všechny výsledné řádky z VD, protože tabulka VYPOCET_PORADI vznikne provedením VD. ND spočítá počet umístění v kvalifikaci od prvního do desátého místa na základě sloupce STARTOVNI_POZICE. Výpočet probíhá

pomocí agregační funkce COUNT(). Výsledek je seřazen podle všech sloupců, které obsahují počet umístění v kvalifikaci v sestupném pořadí.

Dotaz zobrazí u všech jezdců, počet umístění v kvalifikaci od prvního do desátého místa za celou svoji kariéru.

Dotaz byl vytvořen za účelem použití analytické funkce při následovné optimalizaci. Nicméně tento univerzální dotaz nedokáže správně ohodnotit pořadí jezdců, kteří mají naprosto stejný čas ve třetí části kvalifikace. To znamená, že při následovné optimalizaci za použití analytické funkce DENSE_RANK() by univerzální a optimalizovaný dotaz zobrazil jiné výsledky. Z důvodu této skutečnosti jsou data upravena.

4.8.5 Univerzální SQL dotaz č. 5

Dotaz se skládá z vnořeného dotazu (VD) a nadřazeného dotazu (ND), viz Příloha F. VD a ND obsahují v sobě další menší vnořené dotazy. Výsledek VD je pojmenovaná tabulka HLAVNI_VYPOCET. Ve VD jsou zajímavé sloupce VYHRANA_KVALIFKACE a NEJRYCHLEJSI_KOLO_ZAVODU. Hodnoty těchto dvou sloupců jsou zjištěny po vykonání vnořených korelovaných dotazů. Hodnoty výsledných korelovaných dotazů jsou zjištěny na základě vstupních sloupců z nadřazeného dotazu, který je v tomto případě VD. Po dokončení VD máme zjištěny různé statistiky pro jednotlivé jezdce. K tomuto výsledku se připojí řádky z tabulky VYPOCET_NARODNOST. Tabulka VYPOCET_NARODNOST je další vnořený dotaz, který zjišťuje různé statistiky pro jednotlivé národnosti. Tabulku VYPOCET_NARODNOST připojíme k tabulce HLAVNI_VYPOCET za pomocí sloupců, které v sobě obsahují název národnosti. Nyní je na řadě ND, který má k dispozici řádky z tabulek HLAVNI_VYPOCET a VYPOCET_NARODNOST. ND zobrazí jednotlivé sloupce a k tomu si za pomoci dalších vnořených dotazů zjistí sloupce POCET_JEZDCU a POCET_SEZON. Všechny řádky ND jsou seskupeny klauzulí GROUP BY sloupcem NARODNOST. Výsledné řádky jsou seřazeny sestupně podle sloupce POCET_JEZDCU.

Část dotazu pochází z dotazu č. 3. Dotaz zobrazuje statistiky pro jednotlivé národnosti na rozdíl od dotazu č. 3, který zobrazuje statistiky jezdců. Původní verze dotazu byla více podobná dotazu č. 3. Nicméně po neúspěšném provedení u databázového systému MySQL, byl tento dotaz upraven na nynější podobu.

Dotaz zobrazí jednotlivé statistiky podle národností jezdců Formule 1. Statistiky zobrazují celkový počet jezdců za jednotlivou národnost. Počet vyhraných kvalifikací, nejrychlejších kol,

prvních, druhých, třetích míst a celkový počet pódiových umístění. Počet dokončených, nedokončených závodů a počet závodů na bodech. Celkový počet bodů a odjetých kol.

4.8.6 Univerzální SQL dotaz č. 6

Dotaz č. 6 je složen z několika vnořených dotazů viz Příloha G. Nejvnitřnější vnořený dotaz (pojmenovaná tabulka VYPOCET_MAXIMUM_BODU) je základ celého dotazu. Dotaz vypočítá pro každý ročník Formule 1 mistra světa v poháru jezdců. Princip celého dotazu spočívá v určování nejlepšího jezdce podle stanovených atributů. Prvním atributem je maximální bodový zisk, dále počet prvních, druhých, třetích, čtvrtých a pátých míst.

Dotaz zobrazí pro každý rok celé jméno jezdce, startovní číslo, název týmu, motoru a šasi. Dále je zobrazen počet získaných bodů a kolikrát se jezdec umístil od prvního do pátého místa v závodě za celý ročník. Výsledné řádky jsou seřazeny vzestupně podle sloupce ROK.

4.8.7 Univerzální SQL dotaz č. 7

Dotaz obsahuje vnořený dotaz (VD) a nadřazený dotaz (ND), viz Příloha H. Ve VD je deset vnořených téměř identických korelovaných dotazů. Výsledné hodnoty každého korelovaného dotazu jsou zjištěny na základě vstupních sloupců. V tomto případě máme dva druhy vstupních sloupců. První druh sloupců je zadán ručně, kde se vyhodnocuje, zdali jezdec daný závod vyhrál a za jaký tým jel. Druhý druh sloupců pochází z nadřazeného dotazu v tomto případě VD a tyto sloupce rozhodují pro jakého závodníka požadujeme výslednou hodnotu. Výsledné řádky VD jsou použity k zobrazení v ND. ND zobrazuje výsledné řádky ve vzestupném pořadí podle sloupce JEZDEC.

Dotaz zobrazí všechny jezdce Formule 1 a ke každému z nich zobrazí počet vítězství, které získal za jednotlivé týmy Formule 1.

4.8.8 Univerzální SQL dotaz č. 8

Dotaz se skládá z vnořeného dotazu (VD) a nadřazeného dotazu (ND), viz Příloha CH. VD vybere všechny sloupce z tabulek, které jsou součástí dotazu. Důležitá část dotazu spočívá v klauzuli WHERE. Klauzule WHERE obsahuje tři korelované dotazy, které na základě vstupních sloupců rozhodují, zdali dotazovaný řádek splňuje všechny podmínky. Po ukončení VD jsou výsledné řádky v tabulce VYPOCET_HLAVNI. ND zobrazí všechny řádky s vybranými sloupci z pojmenované tabulky VYPOCET_HLAVNI. Na závěr jsou všechny řádky seřazeny ve vzestupném pořadí podle sloupce ROK.

Dotaz zobrazí jednotlivé závody Formule 1 za následujících třech podmínek. Závod Formule 1 vyhraje závodník, který pochází ze země, která pořádá závod a tým je ze stejné země jako závodník.

4.8.9 Univerzální SQL dotaz č. 9

Dotaz se skládá z vnořeného dotazu (VD) a nadřazeného dotazu (ND) viz Příloha I. VD a ND obsahují v sobě další menší vnořené dotazy. Výsledek VD představuje pojmenovanou tabulku HLAVNI_VYPOCET. Ve VD jsou zajímavé sloupce VYHRANA_KVALIFKACE a NEJRYCHLEJSI_KOLO_ZAVODU, kterých je dosaženo za použití vnořených korelovaných dotazů. Hodnoty výsledných korelovaných dotazů jsou zjištěny na základě vstupních sloupců z nadřazeného dotazu, který je v tomto případě VD. Po dokončení VD jsou zjištěny různé statistiky pro jednotlivé týmy. K tomuto výsledku se připojí řádky z tabulky VYPOCET_TYMY. Tabulka VYPOCET_TYMY je další vnořený dotaz, který zjišťuje různé statistiky pro jednotlivé týmy. Tabulku VYPOCET_TYMY připojíme k tabulce HLAVNI_VYPOCET za pomocí sloupců, které v sobě obsahují identifikátor týmu (sloupec ID_TYMU). Nyní je na řadě ND, který má k dispozici řádky z tabulek HLAVNI_VYPOCET a VYPOCET_TYMY. ND zobrazí všechny řádky s vybranými sloupci. Výsledná hodnota sloupce POCET_SEZON je vybrána vnořeným korelovaným dotazem na základě vstupních sloupců z ND. Všechny řádky ND jsou seskupeny klauzulí GROUP BY sloupcem TYM. Výsledné řádky jsou seřazeny vzestupně podle sloupce TYM.

Dotaz č. 9 vznikl modifikací dotazu č. 5. Dotaz zobrazuje statistiky pro jednotlivé týmy na rozdíl od dotazu č. 5, který zobrazuje statistiky národností.

Dotaz zobrazí jednotlivé statistiky podle týmů Formule 1. U daného týmu je zobrazena i informace, z jaké země pochází. Statistiky zobrazují počet vyhraných kvalifikací, nejrychlejších kol, prvních, druhých a třetích míst. Počet dokončených, nedokončených, pódiových a bodovaných závodů. Počet nedokončených závodů je určen za podmínky, kdy dva jezdci ze stejného týmu odjeli méně než 75 % kol závodu. Celkový počet bodů a odjetých kol jsou statistiky za oba jezdce z daného týmu. Poslední dva sloupce informují o počtu závodů a počtu sezón, které tým absolvoval.

4.8.10 Univerzální SQL dotaz č. 10

Dotaz č. 10 je složen z několika vnořených dotazů viz Příloha J. Nejvnitřnější vnořený dotaz (pojmenovaná tabulka VYPOCET_MAXIMUM_BODU) je základ celého dotazu. Dotaz zjistí pro každý ročník Formule1 mistra světa podle poháru konstruktérů. Princip celého dotazu

spočívá v určování nejlepšího týmu podle různých atributů. Prvním atributem je maximální bodový zisk, dále počet prvních, druhých, třetích, čtvrtých a pátých míst.

Dotaz zobrazí pro každý rok celý název týmu, motoru a šasi. Na řádku jsou uvedeny i oba jezdci. Dále jsou zobrazeny statistiky, které dosáhl tým za daný ročník. Počet získaných bodů a kolikrát se umístil od prvního do pátého místa v závodě za celý ročník. Výsledné řádky jsou seřazeny vzestupně podle sloupce ROK.

Dotaz funguje na stejném principu jako dotaz č. 6, který místo týmu určuje nejlepšího jezdce za jeden ročník Formule 1.

5 VÝSLEDKY UNIVERZÁLNÍCH SQL DOTAZŮ

Pro vyhodnocení testovaných univerzálních a optimalizovaných SQL dotazů byla zvolena metoda tzv. „*Vícekriteriálních hodnotících variant*“. Metoda spočívá v určování kritérií a hodnocení variant, více o metodě uvádějí dokumenty [34] a [35].

Po diskuzi s vedoucí bakalářské práce byly stanoveny tři vybraná kritéria:

- doba provedení dotazu,
- průměrné využití procesoru (CPU AVG),
- maximální využití procesoru (CPU MAX).

Zvolené váhy pro jednotlivá kritéria byly navrhnuty vedoucí bakalářské práce:

- doba provedení dotazu (0,7),
- CPU AVG (0,2),
- CPU MAX (0,1).

Největší váhu 0,7 má doba provedení dotazu. Důvodem je obrovský důraz na databázové systémy z hlediska odezvy. Další dvě varianty mají váhy 0,2 a 0,1, které se týkají využití CPU. Využití CPU není důležité z pohledu zákazníka, který se nezajímá o využití hardwarových prostředků.

Klasifikace pro jednotlivá kritéria je zvolena od jedničky do desítky. Rozsah hodnot pro hodnocení je určen podle počtu variant. Varianta v tomto případě představuje počet testovaných SQL dotazů. Hodnota jedna představuje nejhorší výsledek naopak hodnota deset je nejlepší výsledek. Všechna tři vybraná kritéria jsou minimalizační.

Výpočet bodů pro každý jednotlivý dotaz databázového systému představuje hodnotu váženého součtu pro každou variantu. Hodnota váženého součtu je vypočtena podle metody váženého součtu [35].

Ukázka výpočtu bodů pro univerzální SQL dotaz č. 1 u databázového systému Oracle:

Obecný vzorec:

(hodnota hodnotící škály (doba provedení dotazu) * váha (doba provedení dotazu)) + (hodnota hodnotící škály (CPU AVG) * váha (CPU AVG)) + (hodnota hodnotící škály (CPU MAX) * váha (CPU MAX)) = výsledek

(8 * 0,7) + (4 * 0,2) + (7 * 0,1) = 5,6 + 0,8 + 0,7 = 7,1

5.1 Oracle

Tabulka 5 – Výsledky univerzálních SQL dotazů (Oracle) (zdroj vlastní)

Dotaz	Doba provedení dotazu		Využití CPU (%)				Body	Pořadí
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)		
č. 1	83,819	8	24,99	4	27,00	7	7,1	**3**
č. 2	14809,445	1	24,66	8	55,00	2	2,5	**10**
č. 3	531,331	5	24,99	4	28,00	6	4,9	**5**
č. 4	715,209	4	25,02	1	29,00	4	3,4	**7**
č. 5	520,125	6	25,02	1	40,00	3	4,7	**6**
č. 6	10,344	10	15,00	10	26,00	9	9,9	**1**
č. 7	359,560	7	24,99	4	27,00	7	6,4	**4**
č. 8	1619,815	3	24,99	4	29,00	4	3,3	**9**
č. 9	5201,547	2	24,64	9	58,00	1	3,3	**8**
č. 10	35,205	9	25,00	3	26,00	9	7,8	**2**

Výsledky univerzálních SQL dotazů (Oracle) jsou znázorněny v tabulce 5.

Nejlépe hodnoceným univerzálním dotazem u databázového systému Oracle byl dotaz č. 6.

Z výsledků je zřejmé, že dotaz č. 6 dosahoval nejlepších výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. To je s největší pravděpodobností způsobené tím, že dotaz vyhovuje Oraclu z hlediska principu načítání určitého množství dat, ke kterému vždy postupně v daném dotazu vypočítá jeden sloupec skrze korelovaný dotaz. Dotaz postupně zpracovává čím dál tím méně řádků, protože pomocí vypočítaných sloupců skrze korelované dotazy tento počet redukuje. Je velikým překvapením, že právě tento dotaz je nejrychlejší i když je velice složitě napsaný.

Na druhé straně nejhorším hodnoceným dotazem byl dotaz č. 2. Značný vliv na tento výsledek měla doba provedení dotazu. Velmi dlouhá doba provedení dotazu je zapříčiněna zpracováním dvaceti korelovaných dotazů. Korelované dotazy musí databázový systém Oracle zpracovat najednou, jelikož zjišťují hodnoty všech dvaceti sloupců pro jeden výsledný řádek.

Nejkratší doby provedení dotazu bylo dosaženo u dotazu č. 6. Druhé nejnižší hodnoty bylo docíleno u dotazu č. 10. U univerzálního dotazu č. 10 byla doba provedení dotazu více než třikrát větší oproti dotazu č. 6.

Nejnižšího průměrného využití procesoru bylo dosaženo u dotazu č. 6. Průměrné využití procesoru bylo nižší přibližně o deset procentních bodů od ostatních dotazů. Jedná se o velmi výrazný rozdíl.

Nejhorších výsledků z hlediska průměrného využití procesoru dosáhly dotazy č. 4 a 5.

Nejhorších výsledků z hlediska maximálního využití procesoru dosáhly dotazy č. 2 a 9. Maximální využití procesoru v určitý okamžik přesáhlo 55 %, což je dvojnásobek oproti ostatním dotazům.

5.2 MS SQL Server

Tabulka 6 - Výsledky univerzálních SQL dotazů (MS SQL Server) (zdroj vlastní)

Dotaz	Doba provedení dotazu		Využití CPU (%)				Body	Pořadí
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)		
č. 1	226,360	1	25,02	5	80,00	3	2	**10**
č. 2	16,459	4	22,00	9	26,00	9	5,5	**7**
č. 3	11,788	8	40,00	3	75,00	4	6,6	**3**
č. 4	16,257	5	23,00	8	27,00	7	5,8	**5**
č. 5	7,060	10	70,00	2	99,00	1	7,5	**2**
č. 6	36,109	3	24,00	7	26,50	8	4,3	**8**
č. 7	13,554	6	26,00	4	27,25	6	5,6	**6**
č. 8	12,930	7	22,00	9	25,50	10	7,7	**1**
č. 9	9,584	9	75,00	1	99,00	1	6,6	**3**
č. 10	75,008	2	25,02	5	28,00	5	2,9	**9**

Výsledky univerzálních SQL dotazů (MS SQL Server) jsou znázorněny v tabulce 6.

Nejlépe hodnoceným univerzálním dotazem u databázového systému MS SQL Server byl dotaz č. 8.

Z výsledků vyplývá, že dotaz č. 8 neproběhl nejrychleji, ale nejmenší využití procesoru z hlediska průměru a maxima rozhodlo o nejvyšším zisku bodů. To je s největší pravděpodobností způsobené tím, že dotaz vyžaduje načtení velkého množství dat z důvodů velkého počtu tabulek. Dotaz musí pro každý řádek v klauzuli WHERE vyhodnotit tři podmínky. Každá podmínka se vyhodnocuje po provedení vnořeného korelovaného dotazu. MS

SQL server zvládne rychle načíst velké množství dat, ale dokáže i za krátkou dobu zpracovat tři korelované dotazy umístěné v klauzuli WHERE.

Dotaz č. 1 podle hodnocení dopadl nejhůře i když je velice podobný dotazu č. 8. Rozdíl spočívá, v tom že tři podmínky jsou rozdělené do tří vnořených dotazů, které postupně filtrují výsledné řádky. Značný vliv na tento výsledek měla nejdelší doba provedení dotazu.

Dotaz č. 4 podle hodnocení skončil na pátém místě. Obsahuje pouze jeden korelovaný dotaz, takže dotaz lze zařadit mezi méně složité. Nicméně korelovaný dotaz pracuje s velkým objemem dat, takže doba zpracování jednoho výsledného řádku je stejná, jako u dotazů, které mají více korelovaných dotazů. Příkladem můžou být dotazy č. 2 a 7, které mají téměř stejné hodnocení.

Nejkratší doby provedení dotazu bylo dosaženo u dotazu č. 5.

Nejhorších výsledků z hlediska průměrného a maximální využití procesoru dosáhly dotazy č. 5 a 9. Hodnota těchto výsledků byla trojnásobně vyšší než u většiny dotazů v obou kritérií. Na druhé straně dotazy měly nejrychlejší dobu provedení. Dotazy jsou téměř identické a umístily se na druhém a třetím místě v celkovém hodnocení. Dotazy jsou nádhernou ukázkou extrémních výsledků podle hodnocených kritérií.

5.3 MySQL

Tabulka 7 - Výsledky univerzálních SQL dotazů (MySQL) (zdroj vlastní)

Dotaz	Doba provedení dotazu		Využití CPU (%)				Body	Pořadí
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)		
č. 1	11658,468	2	22,85	6	25,00	9	3,5	**7**
č. 2	1926,400	5	24,99	3	26,00	6	4,7	**6**
č. 3	16,767	10	20,00	8	24,00	10	9,6	**1**
č. 4	5262,095	3	24,64	4	27,50	3	3,2	**8**
č. 5	62,671	9	12,85	9	25,50	7	8,8	**2**
č. 6	1006,989	6	25,02	2	26,50	5	5,1	**5**
č. 7	21384,945	1	23,92	5	28,00	2	1,9	**10**
č. 8	2850,253	4	27,14	1	31,00	1	3,1	**9**
č. 9	178,751	8	11,07	10	25,25	8	8,4	**3**
č. 10	458,593	7	21,00	7	27,00	4	6,7	**4**

Výsledky univerzálních SQL dotazů (MySQL) jsou znázorněny v tabulce 7.

Nejlépe hodnoceným univerzálním dotazem u databázového systému MySQL byl dotaz č. 3.

Na druhém a třetím místě skončily dotazy č. 5 a 9, které jsou podobné dotazu č. 3. Hlavním důvodem, proč tyto dotazy obsadili přední příčky hodnocení je, že obsahují pouze dva až tři korelované dotazy, které databázový systém MySQL zvládne rychle zpracovat. Oproti tomu ostatní univerzální dotazy MySQL zpracovává déle, protože obsahují více než tři korelované dotazy. Dotaz č. 3 dosáhl ve dvou hodnotících kritérií na plný počet bodů, kdy doba provedení dotazu má velký koeficient, který zaručuje mnoho bodů. Skvělý výsledek má určitě na svědomí implicitně vytvořené indexy nad cizími klíči, které dotazy využívají. MySQL tento dotaz zvládl vykonat za téměř stejný časový úsek jako databázový systém MS SQL Server. Na druhé straně je využití procesoru menší z hlediska průměru i maxima.

Nejhorším hodnoceným dotazem byl dotaz č. 7. Značný vliv na tento výsledek měla doba provedení dotazu. Velmi dlouhá doba provedení dotazu je zapříčiněna načítáním velkého objemu dat a zpracováním deseti korelovaných dotazů.

Na pátém místě v hodnocení se umístil dotaz č. 6. Dotaz je na první pohled velice složitý a dlouhý. Nicméně databázový systém MySQL dokáže dotaz zpracovat v porovnání s nejhoršími dotazy za adekvátní čas. Dotaz provede pouze jednou načtení velkého množství řádků, které postupně filtruje přes jednotlivé vnořené dotazy. U každého vnořeného dotazu se provádí vždy jeden korelovaný dotaz.

Nejkratší doby provedení dotazu bylo dosaženo u dotazu č. 3. Druhé nejnižší hodnoty bylo docíleno u dotazu č. 5. U univerzálního dotazu č. 5 byla doba provedení dotazu více než třikrát větší oproti dotazu č. 3.

Nejnižšího průměrného využití procesoru bylo dosaženo u dotazů č. 5 a 9. Průměrné využití procesoru bylo nižší přibližně o deset procentních bodů od ostatních dotazů. Jedná se o velmi výrazný rozdíl.

Nejhoršího výsledku z hlediska průměrného a maximálního využití procesoru dosáhl dotaz č. 8.

5.4 PostgreSQL

Tabulka 8 - Výsledky univerzálních SQL dotazů (PostgreSQL) (zdroj vlastní)

Dotaz	Doba provedení dotazu		Využití CPU (%)				Body	Pořadí
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)		
č. 1	4398,927	6	25,02	1	27,75	3	4,7	**6**
č. 2	5761,466	4	24,99	3	27,50	4	3,8	**7**
č. 3	520,160	10	22,49	10	27,25	7	9,7	**1**
č. 4	20174,919	3	24,99	3	27,50	4	3,1	**8**
č. 5	522,582	9	23,89	9	26,75	10	9,1	**2**
č. 6	689,828	8	24,64	7	27,25	7	7,7	**3**
č. 7	57747,584	2	25,02	1	27,50	4	2	**9**
č. 8	82033,705	1	24,99	3	30,00	2	1,5	**10**
č. 9	5061,803	5	23,92	8	33,00	1	5,2	**5**
č. 10	1500,729	7	24,66	6	27,00	9	7	**4**

Výsledky univerzálních SQL dotazů (PostgreSQL) jsou znázorněny v tabulce 8.

Nejlépe hodnoceným univerzálním dotazem u databázového systému PostgreSQL byl dotaz č. 3.

Databázové systémy MySQL a PostgreSQL jsou si velice podobné, takže platí stejný komentář, který je uveden u MySQL. Jediný rozdíl je, že dotaz č. 9 není na třetím místě, ale na pátém.

Na druhé straně nejhorším hodnoceným dotazem byl dotaz č. 8. Značný vliv na tento výsledek měla doba provedení dotazu. Velmi dlouhá doba provedení dotazu je zapříčiněna třemi podmínkami v klauzuli WHERE, které jsou vyhodnocovány na základě vnořených korelovaných dotazů. Dále dotaz načítá velké množství tabulek, které musí mezi sebou spojit. Na druhé straně databázový systém MS SQL Server zpracoval dotaz mnohonásobně rychleji a podle hodnocení dopadl nejlépe. To znamená, že PostgreSQL nedokáže efektivně načíst mnoho tabulek a vyhodnotit vnořené korelované dotazy v klauzuli WHERE jako MS SQL Server.

Z výsledků lze vyčíst hodnocení dotazu č. 9, který se ukazuje jako vyvážená varianta univerzálního dotazu pro databázový systém PostgreSQL. Tento dotaz obsahuje více než pět vnořených dotazů a pouhé tři korelované dotazy. Dotaz je ukázkou dvou extrémů, protože jako

jediný přesáhne hranici třicet procent z hlediska maximálního využití CPU. Na druhé straně průměrné využití CPU je mezi třemi nejlepšími. Doba provedení dotazu odpovídá pátému místu, ale dotaz se provede rychleji oproti nejpomalejšímu dotazu.

Nejdelší doby provedení dotazu bylo dosaženo u dotazu č. 8. Druhé nejdelší doby bylo docíleno u dotazu č. 7. U univerzálního dotazu č. 8 byla doba provedení dotazu o více než 25 000 s delší oproti dotazu č. 7. Dotaz č. 8 byl nejpomalejší ze všech univerzálních dotazů napříč všemi databázovými systémy.

6 VÝSLEDKY OPTIMALIZOVANÝCH SQL DOTAZŮ

Uvedené časy u pokusů můžou být rychlejší nebo pomalejší oproti výsledkům optimalizovaných dotazů, jelikož uvedené časy pocházejí z IDE nástrojů pro jednotlivé databázové systémy. Hodnota nákladu každého databázového systému je bezrozměrné číslo určující jednotku práce. Optimalizované dotazy jsou součástí příloh na přiloženém DVD.

6.1 Oracle

Tabulka 9 - Výsledky optimalizovaných SQL dotazů (Oracle) (zdroj vlastní)

Dotaz	Doba p. d.		Využití CPU (%)				Body	Pořadí	Pořadí UNI	Rozdíl
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)				
č. 1	4,138	5	14,00	3	25,00	4	4,5	**6**	3	▼ 3
č. 2	2,810	8	3,00	8	5,50	8	8,0	**3**	10	▲ 7
č. 3	2,971	7	1,00	9	2,20	9	7,6	**4**	5	▲ 1
č. 4	3,510	6	10,00	6	18,00	6	6,0	**5**	7	▲ 2
č. 5	19,734	1	21,00	1	25,50	3	1,2	**10**	6	▼ 4
č. 6	7,733	4	14,00	3	25,00	4	3,8	**7**	1	▼ 6
č. 7	2,151	10	1,00	9	2,00	10	9,8	**1**	4	▲ 3
č. 8	2,287	9	5,00	7	8,50	7	8,4	**2**	9	▲ 7
č. 9	17,198	2	21,00	1	27,00	1	1,7	**9**	8	▼ 1
č. 10	9,469	3	14,00	3	26,00	2	2,9	**8**	2	▼ 6

Výsledky optimalizovaných SQL dotazů (Oracle) jsou znázorněny v tabulce 9.

Nejlépe hodnoceným dotazem u databázového systému Oracle byl po optimalizaci dotaz č. 7.

Dotaz č. 7 byl postupně optimalizován třemi pokusy. První pokus optimalizace spočíval ve velké úpravě dotazu, kde byly odstraněny všechny korelované dotazy a nepotřebné tabulky. Zmenšil se počet sloupců agregujících v klauzuli GROUP BY, byla přidána podmínka, která vybere nejmenší počet potřebných řádků k provedení dotazu. Došlo k zkrácení doby provedení dotazu z 359,560 s na 0,1 s. Náklady viditelné v exekučních plánech klesly z 2951011 na 2309. Druhým pokusem byl odebrán nadřazený dotaz, který zbytečně vypisoval informace z vnořeného dotazu. Po této úpravě dotaz již neobsahoval žádný vnořený dotaz. Výsledkem je stejná doba provedení, ale náklady znovu klesly na 1217. Třetím pokusem byl znovu více upraven dotaz, aby používal vestavěnou funkci PIVOT(). Změnou došlo ke zvýšení nákladů na

1761, ale doba provedení zůstala stejná. V této optimalizaci nedocházelo ke změnám přístupových cest. Finální doba provedení dotazu přes program DbFit je 2,151 s.

Z tabulky je zřejmé, že dotaz se posunul podle hodnocení ze čtvrtého místa na první. Z výsledků je zřejmé, že dotaz č. 7 dosahoval nejlepších výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. To je s největší pravděpodobností způsobené použitím vestavěné funkce PIVOT(), která je dobře zoptimalizována databázovým systémem Oracle.

U dotazu č. 8 byla provedena optimalizace tím, že došlo k velkým změnám u dotazu. Z vnořeného dotazu se stal hlavní dotaz. V klauzuli WHERE byly nahrazeny tři vnořené korelované dotazy zadanými hodnotami. Po optimalizaci neobsahuje dotaz žádné vnořené nebo korelované dotazy, a je velmi jednoduchý. Náklady z původních 4254965 klesly na 859 a doba provedení dotazu z 1619,815 s na 2 s. Finální doba provedení dotazu přes program DbFit je 2,287 s. Po optimalizaci přistupuje databázový systém Oracle k menším tabulkám pomocí sekvenčního skenování (TABLE ACCESS FULL). K větším tabulkám přistupuje pomocí indexů (INDEX RANGE SCAN a INDEX UNIQUE SCAN). Dotaz podle hodnocení zaznamenal velký posun pořadím vpřed, jelikož se posunul z devátého místa na druhé.

Dotaz č. 6 byl z optimalizován třemi pokusy. V prvním pokusu byla provedena změna dotazu. V celém dotazu byla odstraněna tabulka VYSLEDKY, která nebyla potřebná. Připojení tabulek JEZDCI a TYMY bylo přesunuto z nejvnitřnějšího vnořeného dotazu na úplný závěr dotazu. Tabulky byly přesunuty kvůli zmenšení velikosti dat, která byla vybrána na samotném začátku a postupně filtrována v průběhu celého dotazu. Na úplném konci dotazu připojujeme k výslednému počtu řádků vypisované sloupce z tabulek JEZDCI a TYMY. Dotaz byl proveden z původních 10,344 s na 2,9 s. Náklady klesly z 26328 na 25892. Druhý pokus optimalizace byl proveden pomocí partitioningu, kdy byla rozdělena tabulka VYSLEDKY_ZAVODU na dva oddíly podle sloupce PORADI. Doba provedení dotazu se prodloužila z 2,9 s na 3,5 s a náklady vzrostly na 72091. U třetího pokusu byla přidána do dotazu podmínka, která zaručí přístup pouze do jednoho oddílu tabulky VYSLEDKY_ZAVODU. Tato podmínka je v celém dotazu přidána celkově sedmkrát, protože dotaz si načítá data celkem sedmkrát z tabulky VYSLEDKY_ZAVODU. Náklady se snížily na 40654 a dotaz proběhl za 2,5 s. Finální doba provedení dotazu přes program DbFit činí 7,733 s. I když byl dotaz úspěšně zoptimalizován, oproti hodnocení univerzálních dotazů si pohoršil.

Dotaz v pořadí klesl z první pozice na sedmou. Důvodem je, že dotaz je velice komplikovaný a nenabízí takové možnosti k optimalizaci jako ostatní dotazy.

Dotaz č. 5 byl postupně optimalizován pomocí pěti pokusů. Před optimalizací byla změněna struktura dvou tabulek KVALIFIKACE a VYSLEDKY_ZAVODU. Změna struktury spočívá v použití partitioningu neboli rozdělení tabulky na více oddílů. Tabulka KVALIFIKACE je rozdělena na dva oddíly, kdy v prvním oddílu partition_kk_1 jsou uloženy všechny hodnoty časů ‚0:00.000'. Druhý oddíl partition_kk_2 obsahuje ostatní hodnoty, které nevyhovují prvnímu oddílu. Tabulka VYSLEDKY_ZAVODU je rozdělena na 4 oddíly následovně:

- první oddíl obsahuje hodnoty pořadí 1.–5.,
- druhý oddíl obsahuje hodnoty pořadí 6.–10.,
- třetí oddíl obsahuje hodnoty pořadí 11.–15.,
- čtvrtý oddíl obsahuje hodnoty 16.–20. a hodnotu NC. Hodnota NC znamená Not Classified.

Spuštění univerzálního dotazu po těchto změnách způsobilo navýšení doby provedení dotazu z 520,125 s na 642 s. Náklady vzrostly z 1901786 na 3662019. V prvním pokusu byly přidány čtyři indexy na zrychlení dotazu. Po přidání indexů K_INDEX_DATUM_Q3, NK_INDEX_DATUM_CAS, VZ_INDEX se změnili přístupové cesty z TABLE ACCESS FULL na INDEX FAST FULL SCAN. Operace INDEX FAST FULL SCAN má ve všech třech případech menší náklady než TABLE ACCESS FULL. Přidání indexu KKKK_INDEX_DATUM_Q3 nahradilo dvě operace (INDEX RANGE SCAN a TABLE ACCESS BY GLOBAL INDEX ROWID) za jednu INDEX RANGE SCAN na tabulce KVALIFIKACE. Po přidání indexů byl dotaz znovu spuštěn. Doba provedení dotazu se snížila z 642 s na 463 s. Náklady klesly na 2866764. Ve druhém pokusu byla provedena změna SQL dotazu. Změna spočívala v připojení tabulky NARODNOSTI. Připojení tabulky bylo přesunuto z nejvnitřnějšího vnořeného dotazu do finálního nadřazeného dotazu. Po této úpravě došlo ke zkrácení doby provedení dotazu z 463 s na 413 s. Náklady klesly 2866764 z na 2782896. Ve třetím pokusu byla znovu provedena změna SQL dotazu, ale z původního univerzálního dotazu. Změna byla provedena v připojení tabulky NARODNOSTI a JEZDCI. Připojení tabulek bylo přesunuto z nejvnitřnějšího vnořeného dotazu do finálního nadřazeného dotazu. Po této úpravě došlo ke zkrácení doby provedení dotazu z 463 s na 383 s. Náklady vzrostly z 2866764 na 5536465. Nyní jsou k dispozici dvě verze dotazů, které jsou optimalizovány. V další fázi optimalizace byly přidány dva indexy SL_INDEX_1 a INDEX2. Zbylé dva pokusy optimalizace byly provedeny tak, že se vzaly přepsané dotazy z pokusu dva a tři. Tyto dotazy

se spustili znovu. Rozdíl spočíval v tom, že v databázi byly vytvořeny zmíněné dva indexy. Čtvrtý pokus optimalizace spočíval ve spuštění dotazu z pokusu dva. Došlo ke zkrácení doby provedení dotazu z 413 s na 394 s. Náklady klesly z 2782896 na 2735112. Pátý pokus optimalizace spočíval ve spuštění dotazu z pokusu tři. Došlo k výraznému zkrácení doby provedení dotazu z 383 s na 19 s. Náklady klesly z 5536465 na 2154702. K finální optimalizaci byl zvolen pokus tři respektivě pokus pět. Finální doba provedení dotazu přes program DbFit činí 19,734 s. I když optimalizace dotazu byla úspěšná, tak přesto se dotaz propadl z šestého místa na poslední desáté. Důvodem je velká složitost dotazu, která už neumožňuje dotaz více zrychlit z hlediska doby provedení dotazu.

Nejnižšího průměrného využití procesoru bylo dosaženo u dotazu č. 7. Průměrné využití procesoru bylo nižší přibližně o deset procentních bodů od ostatních dotazů. Jedná se o velmi výrazný rozdíl.

Nejhorších výsledků z hlediska průměrného využití procesoru dosáhly dotazy č. 5 a 9.

Nejnižšího maximálního využití procesoru bylo dosaženo u dotazu č. 7. Maximální využití procesoru bylo nižší přibližně o dvacet procentních bodů od ostatních dotazů. Jedná se o velmi výrazný rozdíl.

Z výsledků vyplývá, že doba provedení všech dotazů nepřesáhla hranici 20 s.

6.2 MS SQL Server

Tabulka 10 - Výsledky optimalizovaných SQL dotazů (MS SQL Server) (zdroj vlastní)

Dotaz	Doba p. d.		Využití CPU (%)				Body	Pořadí	Pořadí UNI	Rozdíl
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)				
č. 1	2,722	8	28,00	2	52,00	2	6,2	**4**	10	▲ 6
č. 2	2,730	7	2,00	9	3,20	9	7,6	**3**	7	▲ 4
č. 3	3,612	3	10,00	7	18,00	7	4,2	**8**	3	▼ 5
č. 4	2,986	6	18,00	4	34,00	3	5,3	**6**	5	▼ 1
č. 5	3,593	4	14,00	5	25,00	5	4,3	**7**	2	▼ 5
č. 6	14,203	2	24,28	3	26,50	4	2,4	**9**	8	▼ 1
č. 7	2,411	9	1,00	10	2,10	10	9,3	**2**	6	▲ 4
č. 8	2,191	10	5,00	8	9,00	8	9,4	**1**	1	**0**
č. 9	3,458	5	13,00	6	24,00	6	5,3	**5**	3	▼ 2
č. 10	33,554	1	75,00	1	100,00	1	1,0	**10**	9	▼ 1

Výsledky optimalizovaných SQL dotazů (MS SQL Server) jsou znázorněny v tabulce 10.

Nejlépe hodnoceným dotazem u databázového systému MS SQL Server byl po optimalizaci dotaz č. 8.

U dotazu č. 8 byla provedena optimalizace dvěma pokusy. První pokus optimalizace byl stejný jako u databázového systému Oracle. Podle exekučního plánu náklady (Estimated Subtree Cost) klesly z původních cca 902,721 na 18,2472, přidělená paměť (Memory Grant) z 557960 na 29504. Doba provedení dotazu klesla z 12,930 s na 1 s. Druhý pokus spočíval v úpravě dotazu, která měla změnit načítání tabulek ZEME A TYMY za použití klauzule WITH. Optimalizátor databázového systému MS SQL Server po úpravě dotazu vygeneroval stejný exekuční plán, takže úprava dotazu neměla žádný vliv. Finální doba provedení dotazu přes program DbFit je 2,191 s. Z výsledků je zřejmé, že dotaz č. 8 dosahoval nejlepších výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. Dotaz podle hodnocení skončil na prvním místě ze všech optimalizovaných dotazů stejně jako u univerzálních dotazů.

U dotazu č. 1 byla provedena optimalizace celkem čtyřmi pokusy. Prvním pokusem byla provedena kompletní změna dotazu. Z dotazu byly odstraněny všechny korelované dotazy umístěné v klauzuli WHERE, které byly přepsány do vnořených dotazů umístěných v klauzuli INNER JOIN. Tyto vnořené dotazy byly umístěny do vnořeného dotazu, který představuje zdrojovou tabulku (VYPOCET_HLAVNI) pro nadřazený dotaz. To znamená, že dotaz se skládá z nadřazeného a vnořeného dotazu. Tabulka VYPOCET_HLAVNI vyfiltruje všechny nepotřebné řádky na základě tří podmínek. Dvě podmínky ze tří jsou sestaveny na základě zjištěných hodnot z nově napsaných vnořených dotazů. Tyto vnořené dotazy zjišťují pro každý řádek hodnotu nejrychlejšího času kvalifikace a nejrychlejšího kola závodu pro jednotlivé závody. V nadřazeném dotazu se připojí zbylé tabulky, které se použijí na vypsání vybraných sloupců společně se sloupci, které byly vybrány v tabulce VYPOCET_HLAVNI. Podle exekučního plánu náklady klesly z původních 909,683 na 60,5481, přidělená paměť z 160024 na 92352. Došlo ke zkrácení doby provedení dotazu z 226,360 s na 1,5 s. Druhý pokus optimalizace spočíval ve změně SQL dotazu. V dotazu byl celý vnořený dotaz (VYPOCET_HLAVNI) přesunut do nadřazeného dotazu, takže dotaz se skládá z hlavního dotazu a ze dvou vnořených dotazů, které byly umístěny v klauzuli INNER JOIN. U vnořených dotazů byly změněny podmínky v klauzuli WHERE. Dále byly přidány dvě podmínky ke každé spojovací podmínce. Jedna podmínka filtrovala řádky ze zdrojových tabulek a druhá podmínka

se přesunula z klauzule WHERE, která filtrovala výsledné řádky. Podle exekučního plánu náklady klesly z původních 60,5481 na 59,3091, přidělená paměť z 92352 na 15960. Doba provedení dotazu se nezměnila. Třetí pokus optimalizace proběhl podle exekučního plánu z druhého pokusu. Exekuční plán doporučoval vytvořit index pro tabulku KVALIFIKACE. Po vytvoření indexu, došlo ke zkrácení doby provedení dotazu z 1,5 s na 0,9 s. Náklady klesly z 59,3091 na 44,0215, přidělená paměť se navýšila z 15960 na 95872. Čtvrtý pokus optimalizace proběhl podle exekučního plánu z třetího pokusu, který doporučoval vytvořit index pro tabulku NEJRYCHLEJSI_KOLA. Po vytvoření indexu, došlo ke zkrácení doby provedení dotazu z 0,9 s na 0,8 s. Finální doba provedení dotazu přes program DbFit činí 2,722 s. Náklady klesly z původních cca 44,0215 na 38,7835, přidělená paměť se nezměnila. Z výsledků lze vyčíst velké zlepšení dotazu. Dotaz se posunul v pořadí vpřed, z desátého místa na čtvrté. Dotaz měl jednu z nejrychlejších dob provedení ze všech optimalizovaných dotazů. Na druhou stranu měl druhý nejhorší výsledek využití procesoru z hlediska průměru a maxima. Tento fakt ovlivnil konečné pořadí dotazu č. 1.

U dotazu č. 10 byla provedena optimalizace pomocí tří pokusů. První pokus optimalizace proběhl úpravou SQL dotazu. Z celého dotazu bylo odstraněno používání tabulky VYSLEDKY. Podle exekučního plánu náklady klesly z 144,724 na 71,6807, přidělená paměť se zvýšila z 50384 na 99840. Došlo ke zkrácení doby provedení dotazu z původních 75,008 s na 25,4 s. Druhý pokus optimalizace spočíval v úpravě SQL dotazu. Dotaz obsahuje několik vnořených dotazů, které přistupují k tabulce VYSLEDKY_ZAVODU. K těmto dotazům je přidána klauzule WHERE, do které je vložena podmínka, která zajistí načítání řádků, které dotaz pouze potřebuje. Podmínka vybere všechny řádky, které mají ve sloupci PORADI hodnoty od 1. do 10. místa. Náklady klesly z 71,6807 na 61,0745, přidělená paměť klesla z 99840 na 64896. Došlo ke zkrácení doby provedení dotazu z 25,4 s na 24,2 s. Třetí pokus optimalizace byl proveden vytvořením indexu, který doporučil exekuční plán z druhého pokusu. Tato optimalizace byla neúspěšná z hlediska doby provedení dotazu. U dotazu došlo ke prodloužení doby provedení dotazu z 24,2 s na 54,9 s. Náklady klesly z 61,0745 na 39,8979, přidělená paměť byla snížena z 64896 na 29264. Třetí pokus optimalizace upozorňuje na skutečnost, že ne vždy je dobré se řídit radami exekučního plánu, proto třetí pokus nebyl zahrnut do optimalizace tohoto dotazu, takže testovaný dotaz proběhl pouze s úpravami SQL dotazu po druhém pokusu. Finální doba provedení dotazu v programu DbFit je 33,554 s. Z výsledků lze vyčíst, že dotaz si v pořadí pohoršil, klesl z devátého místa na desáté. Z výsledků vyplývá, že dotaz č. 10 dosahoval nejhorších výsledků u všech třech hodnotících kritérií. Ať už se jednalo

o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. Dotaz dosáhl nejhoršího možného bodového hodnocení, co lze získat, tj. ze všech třech hodnotících kritérií získal jeden bod. Dotaz je velmi podobný dotazu č. 6, který skončil v hodnocení na devátém místě. Optimalizace u těchto dvou dotazů byla úspěšná, nicméně složitost dotazu byla tak vysoká, že neumožnila takové zlepšení jako u ostatních dotazů, které nabízely větší možnosti optimalizace.

U dotazu č. 3 byla provedena optimalizace pomocí čtyř pokusů. První pokus byl proveden úpravou SQL dotazu. Z dotazu byly odebrány tabulky KVALIFIKACE A VYSLEDKY, které nejsou potřebné. Tabulky JEZDCI A NARODNOSTI byly přemístěny z vnořeného dotazu (HLAVNI_VYPOCET) do nadřazeného dotazu, aby ve vnořeném dotazu bylo co nejméně vypisovaných sloupců. Tyto odebrané sloupce byly přidány do nadřazeného dotazu z důvodu připojených tabulek JEZDCI A NARODNOSTI. Poslední úprava, byla změna podmínek u korelovaných dotazů, které zjišťují u každého jezdce počet vyhraných kvalifikací a počet nejrychlejších kol v závodě. Podle exekučního plánu náklady klesly z 162,109 na 106,667, přidělená paměť z 186656 na 183240. Došlo ke zkrácení doby provedení dotazu z 11,788 s na 5,4 s. Druhý pokus optimalizace spočíval ve vytvoření indexu typu COLUMNSTORE pro tabulku NEJRYCHLEJSI_KOLA. Náklady klesly z 106,667 na 27,7061, přidělená paměť vzrostla z 183240 na 660912. Došlo ke zkrácení doby provedení dotazu z 5,4 s na 1,8 s. Třetí pokus spočíval ve vytvoření indexu typu COLUMNSTORE pro tabulku KVALIFIKACE. Podle exekučního plánu náklady klesly z 27,7061 na 13,5436, přidělená paměť klesla z 660912 na 658488. Došlo ke zkrácení doby provedení dotazu z 1,8 s na 1,33 s. Čtvrtý pokus spočíval ve vytvoření indexu typu COLUMNSTORE pro tabulku VYSLEDKY_ZAVODU. Podle exekučního plánu náklady klesly z 13,5436 na 11,1596, přidělená paměť klesla z 658488 na 660912. Došlo ke zkrácení doby provedení dotazu z 1,33 s na 1,27 s. Finální doba provedení dotazu přes program DbFit činí 3,612 s. Provedená optimalizace pomocí indexů COLUMNSTORE velmi snížila celkové náklady a dobu provedení dotazu, ale oproti tomu zvýšila přidělenou paměť. Dotaz i přes výrazné zlepšení všech hodnocených kritérií se propadl z třetího místa na osmé místo. Důvodem je složitější konstrukce dotazu.

Dotazy č. 3, 5 a 9 jsou si velmi podobné a u všech třech byla provedená téměř stejná optimalizace, přesto si všechny dotazy pohoršili podle hodnocení. Důvodem je složitější konstrukce dotazu, která nenabízí takové zlepšení dotazu oproti jednodušším dotazům.

U dotazů č. 2 a 7 byly provedeny téměř stejné optimalizační pokusy jako tomu bylo u databázového systému Oracle. To znamená, optimalizace změnou SQL, změna SQL za využití funkce PIVOT() a nakonec vytvoření indexu. U dotazu č. 7 byla použita pro výslednou optimalizaci funkce PIVOT(). Z výsledků je zřejmé, že dotaz č. 7 má jen o 0,1 horší bodový součet než dotaz č. 8. Dotaz měl druhý nejrychleji provedený dotaz a využití procesoru z obou hledisek nejmenší, to znamená, že stejně jako u databázového systému Oracle funkce PIVOT() minimálně zatěžuje databázový server z hlediska využití procesoru. Pro výslednou optimalizaci u dotazu č. 2 byla použita verze dotazu bez funkce PIVOT(). Z výsledků vyplývá, že dotaz skončil podle hodnocení na třetím místě. Stejně jako dotaz č. 7 minimálně vytížil procesor a doba provedení dotazu patřila k nejrychlejším.

Nejdelší doby provedení dotazu bylo dosaženo u dotazu č. 10. Třetí nejvyšší hodnotu bylo docíleno u dotazu č. 3. U dotazu č. 10 byla doba provedení dotazu více než desetkrát delší oproti dotazu č. 3.

Největší nároky na využití procesoru z hlediska průměru i maxima se vyskytly u dotazu č. 10. U dotazu č. 10 bylo využití procesoru z hlediska průměru téměř třikrát větší a z hlediska maxima téměř dvakrát větší oproti dotazu č. 1.

6.3 MySQL

Tabulka 11 - Výsledky optimalizovaných SQL dotazů (MySQL) (zdroj vlastní)

Dotaz	Doba p. d.		Využití CPU (%)				Body	Pořadí	Pořadí UNI	Rozdíl
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)				
č. 1	4,337	8	9,00	7	18,00	7	7,7	**3**	7	▲ 4
č. 2	4,526	7	5,00	8	10,00	8	7,3	**4**	6	▲ 2
č. 3	8,983	6	22,00	4	28,00	1	5,1	**5**	1	▼ 4
č. 4	28,837	3	24,30	3	26,00	6	3,3	**8**	8	**0**
č. 5	18,957	5	18,00	5	27,50	3	4,8	**6**	2	▼ 4
č. 6	749,146	1	24,66	1	27,75	2	1,1	**10**	5	▼ 5
č. 7	2,611	10	4,00	10	7,00	10	10	**1**	10	▲ 9
č. 8	3,283	9	5,00	8	9,00	9	8,8	**2**	9	▲ 7
č. 9	21,587	4	17,00	6	26,50	5	4,5	**7**	3	▼ 4
č. 10	225,073	2	24,64	2	27,00	4	2,2	**9**	4	▼ 5

Výsledky optimalizovaných SQL dotazů (MySQL) jsou znázorněny v tabulce 11.

Nejlépe hodnoceným dotazem u databázového systému MySQL byl po optimalizaci dotaz č. 7.

Dotaz č. 7 byl postupně optimalizován dvěma pokusy. První pokus optimalizace spočíval ve velké úpravě dotazu, kde byly odstraněny všechny korelované a vnořené dotazy. Zmenšil se počet sloupců agregujících v klauzuli GROUP BY, byla přidána podmínka, která vybere nejmenší počet potřebných řádků k provedení dotazu. Dotaz byl proveden z původních 21384,945 s na 1,6 s. Podle porovnaných exekučních plánů náklady klesly z 1140278,75 na 275429,36. Ve druhém pokusu byly odebrány nepotřebné tabulky a ve funkci COUNT() byla nahrazena klauzule CASE WHEN za IF. Po těchto úpravách došlo ke zkrácení doby provedení dotazu z 1,6 s na 0,4 s. Finální doba provedení dotazu přes program DbFit činí 2,611 s. Náklady klesly z 275429,36 na 122357,11. Změna klauzule nepřinesla žádné zrychlení dotazu. Na druhé straně odebrané tabulky měly velký vliv na snížení nákladů a zrychlení doby provedení dotazu. Z exekučního plánu je patrné, že databázový systém MySQL využil k načítání tabulek dva indexy vytvořené nad cizími klíči a jedno sekvenční skenování (FULL TABLE SCAN) malé tabulky. Z výsledků je zřejmé, že dotaz č. 7 dosahoval nejlepších výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. Dotaz dosáhl nejlepšího možného bodového součtu, co lze získat, tj. ze všech třech hodnotících kritérií získal maximální počet bodů. Z výsledků vyplývá, že dotaz č. 7 zaznamenal největší posun v pořadí vpřed ze všech optimalizovaných dotazů. Dotaz se posunul z desátého místa na první. Dotaz po optimalizaci je velice triviální, který používá malý počet tabulek. Je ze všech optimalizovaných dotazů nejrychlejší a má minimální využití CPU.

Dotaz č. 6 byl optimalizován třemi pokusy. V prvním pokusu byla provedena změna dotazu. V celém dotazu byla odstraněna tabulka VYSLEDKY, která nebyla potřebná. Připojení tabulek JEZDCI a TYMY bylo přesunuto z nejvnitřnějšího vnořeného dotazu na úplný závěr dotazu. Tabulky byly přesunuty kvůli zmenšení velikosti dat, která byla vybrána na samotném začátku a postupně filtrována v průběhu celého dotazu. Na úplném konci dotazu připojujeme k výslednému počtu řádků vypisované sloupce z tabulek JEZDCI a TYMY. Dotaz byl proveden z původních 1006,989 na 771 s. Náklady vzrostly z 1073171,25 na 3439832,05. Druhý pokus optimalizace byl proveden za využití partitioningu, kdy byla rozdělena tabulka VYSLEDKY_ZAVODU na dva oddíly podle sloupce PORADI. Došlo k navýšení doby provedení dotazu z 771 s na 779 s. Náklady klesly z 3439832,05 na 1716944,45. Třetím pokusem je přidána do dotazu podmínka, která zaručí přístup pouze do jednoho oddílu tabulky VYSLEDKY_ZAVODU. Tato podmínka je v celém dotazu přidána celkově sedmkrát, protože

dotaz si načítá data celkem sedmkrát z tabulky VYSLEDKY_ZAVODU. Náklady klesly na 858475,40 a došlo ke zkrácení doby provedení dotazu z 779 s na 749,146 s. Finální doba provedení dotazu v programu DbFit je 749,146 s. V uloženém exekučním plánu (HTML) z třetího pokusu, lze vidět přístup pouze do jednoho oddílu tabulky VYSLEDKY_ZAVODU oproti druhému pokusu. Z výsledků vyplývá, že dotaz klesl z pátého místa na poslední desáté místo. Z výsledků je zřejmé, že dotaz získal v celkovém součtu 1,1 bodů a přiblížil se k nejmenšímu možnému zisku, který je 1,0. Všechny výsledné hodnoty všech hodnocených kritérií měly značný vliv na tento výsledek. Dotaz č. 6 je velice komplikovaný a nenabízí takové možnosti optimalizace jako ostatní dotazy.

U dotazu č. 8 byla provedena optimalizace čtyřmi pokusy. První pokus spočívá v úpravě celého SQL dotazu. Z vnořeného dotazu se stal hlavní dotaz. V klauzuli WHERE byly nahrazeny tři vnořené korelované dotazy zadanými hodnotami. Po optimalizaci neobsahuje dotaz žádné vnořené nebo korelované dotazy a je velmi jednoduchý. Náklady klesly z původních 4197457,75 na 65629,29. Dotaz byl proveden z původních 2850,253 s na 3,5 s. Podle exekučního plánu je přístup k tabulkám zprostředkován přes primární a cizí klíče, které mají vytvořené indexy. Druhou optimalizací je využití výhod partitioningu, kdy je rozdělena tabulka VYSLEDKY_ZAVODU na dva oddíly podle sloupce CAS_ODSTUP. Oddíl p1 obsahuje zhruba 70 % řádků tabulky. Oddíl p2 obsahuje zbylých 30 % řádků, v kterých jsou obsaženy řádky využívané v dotazu. Podle vizuálního exekučního plánu náklady vzrostly z 65629,29 na 69557,74. Došlo ke prodloužení doby provedení dotazu z 3,5 s na 3,7 s. Podle tabulkového exekučního plánu lze vyčíst, že u tabulky VYSLEDKY_ZAVODU (VZ) je přístup přes dva oddíly p1 a p2. Třetí pokus optimalizace spočívá v úpravě SQL dotazu. U dotazu byla přidána podmínka, která zajistí přístup pouze k oddílu p2. Dotaz nebude muset procházet celé dva oddíly tabulky VZ. Podle vizuálního exekučního plánu náklady klesly z 69557,74 na 69309,38. Došlo ke zkrácení doby provedení dotazu z 3,7 s na 2,4 s. V tabulkovém exekučním plánu lze vidět úspěšnou změnu přístupu k oddílům. Čtvrtý pokus optimalizace spočíval v úpravě dotazu. Změna dotazu se zakládala na přidání dvou podmínek do spojovací podmínky pro tabulky VYSLEDKY a VYSLEDKY_ZAVODU. Přidané dvě podmínky kopírovaly dvě poslední podmínky z klauzule WHERE. Podle vizuálního exekučního plánu náklady klesly z 69309,38 na 59165,68. Došlo ke zkrácení doby provedení dotazu z 2,4 s na 1,4 s. Finální doba provedení dotazu v programu DbFit je 3,283 s. Z obou poslední vytvořených exekučních plánů lze odvodit, že optimalizátor změnil kompletně celý exekuční plán. Změna spočívá v načítání tabulek, které optimalizátor načítá pomocí primárních klíčů (unikátní indexy). Výjimkou je

tabulka VYSLEDKY_ZAVODU, u které optimalizátor prohledá celý oddíl. Dotaz č. 8 se posunul v pořadí vpřed, z devátého místa na druhé. Z výsledků lze vyčíst, že dotaz dosahoval druhých nejlepších výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru.

U dotazu č. 4 probíhala optimalizace ve dvou fázích. První fáze optimalizace spočívala v testování na malé množině dat z důvodu zkrácení doby čekání. Při této fázi optimalizace bylo zjištěno, že úprava SQL dotazu nepřinese téměř žádný účinek. Oproti tomu vytvoření indexu v tabulce STARTOVNI_LISTINY na sloupci ROK přinese extrémní zkrácení doby provedení dotazu. Po této zjištěné informaci byla databáze vrácena do původního stavu. Druhá fáze optimalizace probíhala na základě zjištěných informací z první fáze. Druhá fáze optimalizace spočívala ve třech pokusech. V prvním pokusu byl vytvořen zmíněný index, který zajistil zkrácení doby provedení dotazu. Díky tomu mohla optimalizace probíhat mnohem rychleji. To znamená, že první pokus byl proveden pomocí univerzálního dotazu, kdy v databázi byl již vytvořen index pro tabulku STARTOVNI_LISTINY. Při porovnání vizuálních exekučních plánů univerzálního dotazu a prvního pokusu optimalizace jsou náklady stejné tj. 25146,50. Náklady jsou stejné, protože se nezměnil počet připojených tabulek v dotazu. Vytvořený index použije vnořený korelovaný dotaz, který neovlivní celkové náklady. Dotaz byl proveden z původních 5262,095 s na 36 s. Druhý pokus optimalizace byl proveden několika změnami v SQL dotazu. Byly odebrány tabulky VYSLEDKY, ZAVODY, ZAVODNI_OKRUHY. Z vnořeného korelovaného dotazu byla odebrána třetí podmínka v klauzuli WHERE. Z vnořeného dotazu, který představuje zdrojovou tabulku (VYPOCET_PORADI) byly odebrány sloupce, které nejsou potřebné v nadřazeném dotazu. U vnořeného dotazu ještě byla změněna podmínka, kdy hodnota na pravé straně se změnila z %1:% na 1:%. Dále u funkce COUNT() byla nahrazena klauzule CASE WHEN za klauzuli IF. Po těchto úpravách v dotazu došlo ke zkrácení doby provedení dotazu z 36 s na 23,1 s. Podle vizuálního exekučního plánu jsou pryč odebrané tabulky, ale cena nákladů se nezměnila. Třetí pokus optimalizace spočíval v přidání jedné podmínky ke spojovací podmínce tabulek KVALIFIKACE A STARTOVNI_LISTINY. Přidaná podmínka je stejná jako podmínka hlavního dotazu v klauzuli WHERE. Podle vizuálního exekučního plánu náklady klesly z 25146,50 na 2961,50 z důvodu změny metod přístupu k tabulkám. Optimalizátor načte tabulku KVALIFIKACE sekvenčním skenováním (TABLE FULL SCAN). Tabulky STARTOVNI_LISTINY a JEZDCI načte pomocí indexů, které jsou vytvořeny nad sloupci primárního klíče zmíněných tabulek. Došlo ke zkrácení doby provedení dotazu z 23,1 s na 21,9 s. Finální doba provedení dotazu

v programu DbFit činí 28,837 s. Dotaz podle hodnocení si udržel po optimalizaci stejnou pozici, jelikož ostatní dotazy měly většinou lepší výsledky v oblasti doby provedení dotazu. Optimalizace dotazu č. 4 je zmíněna z důvodu, že databázový systém MySQL nepodporuje analytické funkce. K optimalizaci nemohla být použita funkce ROW_NUMBER() jako u ostatních databázových systémů. Tato optimalizace znázorňuje nádherný příklad, správně zvoleného indexu, který může extrémně zrychlit dotaz. Z výsledků je zřejmé, že dotaz po velice úspěšné optimalizaci dosáhl stejného pořadí jako u univerzálních dotazů. Z toho můžeme usoudit, že ostatní dotazy byly optimalizovány v podobném měřítku jako dotaz č. 4.

Nejnižšího maximálního využití procesoru bylo dosaženo u dotazu č. 7. Maximální využití procesoru bylo nižší přibližně o dvacet procentních bodů od ostatních dotazů. Jedná se o velmi výrazný rozdíl.

Nejdelší doby provedení dotazu bylo dosaženo u dotazu č. 6. Druhé nejdelší doby bylo docíleno u dotazu č. 10. U optimalizovaného dotazu č. 6 byla doba provedení dotazu více než třikrát delší oproti dotazu č. 10. Dotaz č. 6 byl u databázového systému MySQL nejpomalejší ze všech optimalizovaných dotazů napříč všemi databázovými systémy.

6.4 PostgreSQL

Tabulka 12 - Výsledky optimalizovaných SQL dotazů (PostgreSQL) (zdroj vlastní)

Dotaz	Doba p. d.		Využití CPU (%)				Body	Pořadí	Pořadí UNI	Rozdíl
	(s)	(0,7)	AVG	(0,2)	MAX	(0,1)				
č. 1	23,959	5	24,99	2	27,75	2	4,1	**6**	6	**0**
č. 2	3,646	7	13,00	7	24,50	7	7	**4**	7	▲ 3
č. 3	296,228	1	24,99	2	30,00	1	1,2	**10**	1	▼ 9
č. 4	3,971	6	14,00	6	25,00	6	6	**5**	8	▲ 3
č. 5	294,665	2	25,02	1	27,00	5	2,1	**9**	2	▼ 7
č. 6	81,741	3	24,99	2	27,50	3	2,8	**8**	3	▼ 5
č. 7	1,946	8	0,20	8	0,40	8	8	**3**	9	▲ 6
č. 8	1,750	9	0,15	9	0,30	9	9	**2**	10	▲ 8
č. 9	1,366	10	0,15	9	0,30	9	9,7	**1**	5	▲ 4
č. 10	81,404	4	24,99	2	27,25	4	3,6	**7**	4	▼ 3

Výsledky optimalizovaných SQL dotazů (PostgreSQL) jsou znázorněny v tabulce 12.

Nejlépe hodnoceným dotazem u databázového systému PostgreSQL byl po optimalizaci dotaz č. 9.

U dotazu č. 9 je provedena optimalizace dvěma pokusy. První pokus optimalizace spočíval v rozložení celého dotazu na pět částí, kdy jednotlivé části byly do databáze uloženy jako materializované pohledy. Po vytvoření pohledů byl kompletně napsán celý nový dotaz, který použil všech pět vytvořených materializovaných pohledů jako připojené tabulky. Pohledy k dotazu byly připojeny na základě sloupce TYM, který představoval sloupec ID_TYMU. Vytvořený dotaz je triviální, protože vybere pouze potřebné sloupce ze všech připojených pohledů, které na konci seřadí podle sloupce TYM. Dotaz byl proveden z původních 1500,729 s na 0,3 s. Podle exekučního plánu náklady klesly z 183146412955,23 na 2003,72. U nového exekučního plánu můžeme vidět, že všechny tabulky (materializované pohledy) jsou načteny sekvenčním skenování (SEQ SCAN), které je levné a rychlé. Druhý pokus optimalizace spočíval v provedení příkazu, který obsahoval slovo ANALYZE. ANALYZE v celé databázi provede sběr statistik k jednotlivým tabulkám, které uloží do systémového katalogu pg_statistic. Tyto shromážděné statistiky plánovač dotazů (Query Planner) použije ke sestavení nejúčinnějšího exekuční plánu. Po aktualizaci statistik došlo ke zkrácení doby provedení dotazu z 0,3 s na 0,15 s. Náklady klesly z 2003,72 na 6,74. Finální doba provedení dotazu přes program DbFit činí 1,366 s. Z exekučních plánů lze rozpoznat, že plánovač dotazu změnil metody spojování tabulek z Merge Join na Hash Join, které jsou méně nákladné. Z výsledků je zřejmé, že dotaz č. 9 dosahoval nejlepších výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. To je s největší pravděpodobností způsobené tím, že dotaz využívá materializované pohledy. Dotaz č. 9 se posunul v pořadí vpřed, z pátého místa na první.

U dotazu č. 8 je provedena optimalizace dvěma pokusy. První pokus optimalizace spočívá v modifikaci SQL dotazu, která je provedena úplně stejně jako u databázového systému Oracle. Po této změně byl dotaz proveden z původních 82033,705 s na 0,8 s. Náklady klesly z 77303,97 na 6919,92. Druhý pokus optimalizace byl proveden vytvořením indexu v databázi. Vytvořený index VZ_INDEX_CAS_ODSTUP zahrnoval sloupce ID, DATUM, CAS_ODSTUP. Index byl vytvořen k tabulce VYSLEDKY_ZAVODU. Dále u indexu byla ještě nadefinována podmínka pro sloupec CAS_ODSTUP. Tato podmínka byla úplně stejná jako u modifikovaného dotazu. Po spuštění dotazu došlo ke zkrácení doby provedení dotazu z 0,8 s na 0,3 s. Náklady klesly z 6919,92 na 5986,61. Finální doba provedení dotazu přes program DbFit činí 1,750 s. Exekuční plány prvního a druhého pokusu byly rozdílné jen v jedné operaci.

Operace použila pro prohledání tabulky VYSLEDKY_ZAVODU vytvořený index místo primárního klíče. Z výsledků je zřejmé, že dotaz č. 8 dosahoval nejlepších druhých výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. To je s největší pravděpodobností způsobené tím, že složitost dotazu je jednoduchá a nevyžaduje pro databázový systém příliš mnoho práce. Dotaz č. 8 se posunul v pořadí vpřed, z posledního desátého místa na druhé.

Dotaz č. 2 byl postupně optimalizován dvěma pokusy. První pokus optimalizace spočíval ve velké modifikaci dotazu, kde byly odstraněny všechny korelované dotazy a nepotřebné tabulky. Zmenšil se počet sloupců agregujících v klauzuli GROUP BY, byla přidána podmínka, která vybere nejmenší počet potřebných řádků k provedení dotazu. Po těchto úpravách se dotaz skládá pouze z nadřazeného a vnořeného dotazu. Dotaz byl proveden z původních 5761,466 s na 0,8 s. Podle exekučního plánu náklady klesly z 7375792031,55 na 39989,49. Poslední druhý pokus optimalizace byl proveden pomocí podobné funkce PIVOT() jako u databázových systémů Oracle a MS SQL Serveru. Funkce u databázové systému PostgreSQL je označena pod názvem CROSSTAB(). Jedinou podmínkou k použití této funkce je provedení následujícího příkazu: CREATE EXTENSION tablefunc;. Druhý pokus optimalizace spočíval v přepsání celého dotazu s využitím funkce CROSSTAB(). Po provedení příkazu a úpravě dotazu došlo k prodloužení doby provedení dotazu z 0,8 s na 2 s. Náklady klesly z 39989,49 na 10,00. K optimalizaci dotazu je zvolen druhý pokus, který ukáže, jak náročná je na využití procesoru funkce CROSSTAB(). Finální doba provedení dotazu přes program DbFit činí 3,646 s. Z výsledků je zřejmé, že dotaz č. 2 dosahoval nejlepších čtvrtých výsledků u všech hodnocených kritérií. Ať už se jednalo o dobu provedení dotazu, dále pak průměrné nebo maximální využití procesoru. Dotaz č. 2 se posunul v pořadí vpřed, ze sedmého místa na čtvrté. Z výsledku dotazu můžeme usoudit, že použitá funkce není tak dobře zoptimalizována jako u databázových systémů Oracle a MS SQL Serveru.

Dotaz č. 3 byl postupně optimalizován čtyřmi pokusy. Před optimalizací byla změněna struktura tabulky VYSLEDKY_ZAVODU. Tabulka VYSLEDKY_ZAVODU je rozdělena na 2 oddíly následovně:

- první oddíl obsahuje hodnoty pořadí 1.–10.,
- druhý oddíl obsahuje hodnoty pořadí 10.–20. a hodnotu NC. Hodnota NC znamená Not Classified.

Spuštění univerzálního dotazu po těchto změnách způsobilo navýšení doby provedení dotazu z 520,160 s na 600 s. Náklady klesly z 123634027605,06 na 15363407,60. První pokus byl proveden úpravou SQL dotazu. Z dotazu byly odebrány tabulky KVALIFIKACE A VYSLEDKY. Tabulky JEZDCI A NARODNOSTI byly přemístěny z vnořeného dotazu (HLAVNI_VYPOCET) do nadřazeného dotazu, aby ve vnořeném dotazu bylo co nejméně vypisovaných sloupců. Tyto odebrané sloupce byly přidány do nadřazeného dotazu z důvodu připojených tabulek JEZDCI A NARODNOSTI. Po modifikaci a spuštění dotazu došlo k zkrácení doby provedení dotazu z 600 s na 540 s. Náklady vzrostly 15363407,60 na 102985657,42. Druhý pokus optimalizace byl proveden vytvořením indexu v databázi. Vytvořený index KK_INDEX_Q3_3 zahrnoval sloupce DATUM a Q3. Index byl vytvořen k tabulce KVALIFIKACE. Dále byla u indexu ještě nadefinována podmínka pro sloupec Q3. Tato podmínka byla úplně stejná jako u modifikovaného dotazu. Náklady klesly z 102985657,42 na 91739242,02. Došlo ke zkrácení doby provedení dotazu z 540 s na 420 s. Třetí pokus optimalizace byl proveden vytvořením indexu v databázi. Vytvořený index NK_INDEX_CAS_3 zahrnoval sloupce DATUM a CAS. Index byl vytvořen k tabulce NEJRYCHLEJSI_KOLA. Dále u indexu byla ještě nadefinována podmínka pro sloupec CAS. Tato podmínka byla úplně stejná jako u modifikovaného dotazu. Náklady klesly z 91739242,02 na 75682781,13. Došlo ke zkrácení doby provedení dotazu z 420 s na 360 s. Čtvrtý pokus optimalizace byl proveden vytvořením indexu v databázi. Vytvořený index SL_INDEX_ID_JEZDCE zahrnoval sloupec ID_JEZDCE. Index byl vytvořen k tabulce STARTOVNI_LISTINY. Náklady klesly z 75682781,13na 74414352,28. Došlo k zkrácení doby provedení dotazu z 360 s na 300 s. Finální doba provedení dotazu přes program DbFit činí 296,228 s. Z výsledků vyplývá, že dotaz klesl z prvního místa na poslední desáté místo. Z výsledků je zřejmé, že dotaz získal v celkovém součtu 1,2 bodu a přiblížil se k nejmenšímu možnému zisku, který je 1,0. Značný vliv na tento výsledek měly všechny výsledné hodnoty hodnocených kritérií. Dotaz č. 3 je velice komplikovaný a nenabízí takové možnosti optimalizace jako ostatní dotazy.

Nejdelší doby provedení dotazu bylo dosaženo u dotazu č. 3. Třetí nejdelší doby bylo docíleno u dotazu č. 6. U dotazu č. 3 byla doba provedení dotazu více než třikrát delší oproti dotazu č. 6.

Nejkratší doby provedení dotazu bylo dosaženo u dotazu č. 9.

ZÁVĚR

První část práce je zaměřena na teoretická východiska spojená s aplikací do praktické části. V teoretické části byly popsány základní pojmy a jednotlivé databázové systémy, jednalo se o Oracle Database, MS SQL Server, MySQL a PostgreSQL. Nemalá část práce v teoretických východiscích se věnovala bezpečnosti databázových systémů.

V praktické části práce byly testovány a komparovány možnosti využití univerzálních SQL dotazů a optimalizovaných SQL dotazů u následujících databázových systémů Oracle, MS SQL Server, MySQL, PostgreSQL včetně jejich vlivu na dobu provedení dotazu a využití CPU. Pro testování SQL dotazů byl použit program DbFit. Univerzální a optimalizované SQL dotazy byly vícekriteriálně posouzeny u všech 4 vybraných databázových systémů, jak pro dobu provedení dotazu, tak i využití CPU.

Před testováním SQL dotazů byl vytvořen jeden základní databázový model. Tento model sloužil jako základ pro následně vygenerované skripty jazyka SQL, které jsou použitelné pro vytvoření databázového modelu na jednotlivé databázové systémy. Každý databázový server podporuje odlišné datové typy, které zabraňují vytvořit jeden univerzální SQL skript na vytvoření databázového modelu ve všech databázových systémech zároveň. Základ databázového modelu tvoří databáze obsahující data ze závodů Formule 1.

Pro testování databázových serverů byly vytvořeny univerzální SQL dotazy. Univerzální SQL dotazy lze spustit na všech databázových serverech, které byly vybrány pro tuto práci. Vytvořit dotazy bylo časově náročné a z hlediska podpory standardu jazyka SQL i velmi složité. Vytvoření přenositelných SQL dotazů mezi jednotlivými databázovými systémy naznačuje unikátnost jazyka SQL, i když přináší některá omezení z důvodu podpory standardu jazyka SQL.

Podrobné vícekriteriální hodnocení univerzálních dotazů je uvedeno v tabulkách č. 5, 6, 7 a 8. Z výsledků univerzálních SQL dotazů je zřejmé, že dotazy nejsou efektivní. Důvodem je použití vnořených a korelovaných dotazů v univerzálních SQL dotazech. Databázové systémy zpracují vnořené a korelované dotazy za vysoké náklady. Výjimkou je MS SQL Server, který má nejdelší dobu provedení okolo 230 s. Ostatní tři databázové systémy měly nejdelší dobu provedení univerzálního dotazu přes 10 000 s. Z toho lze usoudit, že vytvořené univerzální SQL dotazy nejsou vhodné pro použití v praxi.

Po optimalizování všech dotazů je zřejmé, že byly maximálně použity poznatky z teorie. U každého dotazu na jednotlivém databázovém systému byla zkrácena doba provedení dotazu. Podle výsledných exekučních plánů bylo zřejmé, že když se v exekučním plánu snížily náklady, tak se zpravidla zkrátila i doba provedení dotazu. Zmíněný trend platil u každého databázového systému. Bylo zjištěno, že u každého databázového systému došlo ke zlepšení výkonu dotazu po vytvoření vhodně zvoleného indexu. U MySQL jsou z historických důvodu vývoje tohoto systému implicitně vytvořeny indexy i nad sloupci cizích klíčů. Tato vlastnost souvisí s používáním úložiště InnoDB. PostgreSQL umožňuje vytvořit index s nadefinovanou podmínkou. Z toho lze usoudit, že vytvořený index určitě není unikátní a neobsahuje veškeré záznamy. Pokud index neobsahuje veškeré záznamy nezabírá příliš velký objem místa na disku a lze ho velice rychle prohledat. Optimalizace SQL dotazu za pomoci materializovaných pohledů je velice efektivní. Materializované pohledy umožňují vykonat dotazy okamžitě. Materializované pohledy podporuje Oracle a PostgreSQL. U MS SQL Serveru byly vytvořeny COLUMNSTORE indexy. Tento druh indexů způsobil extrémní zkrácení doby provedení dotazu. Na druhou stranu vzrostlo prudce využití paměti podle exekučních plánů. Oracle, MS SQL Server a PostgreSQL podporují analytické funkce. Analytické funkce zkrátí dobu provedení dotazu a využití CPU databázového systému bude minimální. Tyto funkce mají databázové systémy velice dobře zoptimalizovány. Další dobrou vlastností analytické funkce je zpřehlednění dotazu SQL. Pro zjištění větších dopadů na výkon pomocí partitioningu by bylo lepší mít k dispozici více dat.

Podrobné vícekriteriální hodnocení optimalizovaných dotazů je uvedeno v tabulkách č. 9, 10, 11 a 12. Databázové systémy MS SQL Server a Oracle jsou z pohledu optimalizace na stejné úrovni. V úvahu není brán dotaz č. 10, který u MS SQL Server měl dobu provedení dotazu přes 33 s. Dále následuje databázový systém PostgreSQL, který všechny dotazy zvládl vykonat do 300 s. Na poslední místě skončil MySQL, u kterého nejpomalejší dotaz proběhl za 750 s. PostgreSQL zpracoval tři dotazy pod 2 s jako jediný ze všech databázových systémů. Zjištění bylo překvapující, protože PostgreSQL je k dispozici jako open source.

DbFit nezobrazuje korektní čas za který databázový systém zpracuje dotaz, protože musí provést kontrolu výsledků dotazu s uloženými výsledky z programu DbFit. DbFit využívá JDBC pro komunikaci s každým databázovým systémem.

Výsledky bakalářské práce korespondují s výsledky výzkumu společnosti Gartner z hlediska doby provedení dotazu. Z výsledků hodnocení optimalizovaných SQL dotazů dopadly

databázové systémy Oracle a MS SQL Server nejlépe podle kritéria doby provedení dotazu. Společnost Gartner vyhodnotila podle definovaných kritérií jako nejlepší databázové systémy Oracle a MS SQL Server.

Práce by se mohla dále rozvíjet v oblasti optimalizace výkonu databázových systému z hlediska nastavení samotného serveru a databázového systému a dále v oblasti využití statistických a analytických funkcí.

POUŽITÁ LITERATURA

[1] HERNANDEZ, Michael J. *Návrh databází*. Praha: Grada, 2006. Profesionál. ISBN 80- 247-0900-7.

[2] BUCHALCEVOVÁ, Alena, Iva STANOVSKÁ a Milan ŠIMŮNEK. *Základy softwarového inženýrství - základní témata*. Praha: Oeconomica, 2002. ISBN 80-245-0346-8.

[3] FARANA, Radim. Sylaby, elektronické učebnice. *Databázové systémy* [online]. Ostrava: Fakulta strojní - VŠB-TU Ostrava, 1995 [cit. 2018-05-06]. Dostupné z: http://books.fs.vsb.cz/dbacc20/Welcome.htm

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

[5] OPPEL, Andrew J. *Databáze bez předchozích znalostí: [průvodce pro samouky]*. Brno: Computer Press, 2006. ISBN 80-251-1199-7.

[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] SHELDON, Robert. *SQL: začínáme programovat*. Praha: Grada, 2005. Průvodce (Grada). ISBN 80-247-0999-6.

[8] DOSEDĚL, Tomáš. *Počítačová bezpečnost a ochrana dat*. Brno: Computer Press, 2004. ISBN 80-251-0106-1.

[9] ZDVOŘÁČEK, Bc. Jan. *Public Key Infrastructure (PKI) - návrh metrik pro výběr [online]*. Praha, 2013 [cit. 2017-07-28], 102 s. Dostupné také z: https://vskp.vse.cz/eid/55498. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce RNDr. Igor Čermák, CSc.

[10] FreeRADIUS Technical Guide. *Network RADIUS SARL* [online]. 2014 [cit. 2017-07-28]. Dostupné z: http://networkradius.com/doc/FreeRADIUS%20Technical%20Guide.pdf

[11] THERIAULT, Marlene a Aaron NEWMAN. *Bezpečnost v Oracle: metody, nástroje a řešení problémů : [platné pro Oracle 9i, 8i, 8 a 7.x]*. Brno: Computer Press, 2004. Security (CP Books). ISBN 80-722-6979-8.

[12] BRYLA, Bob a Kevin LONEY. *Mistrovství v Oracle Database 11g*. Brno: Computer Press, 2009. ISBN 978-80-251-2189-4.

[13] GROFF, James R. a Paul N. WEINBERG. *SQL: kompletní průvodce*. Brno: CP Books, 2005. Programování (CP Books). ISBN 80-251-0369-2.

[14] SPĚVÁKOVÁ, Barbora. *Optimalizace SQL dotazů v Oracle*. Hradec Králové, 2016 [cit. 2017-07-28], s. 67. Dostupné také z: http://theses.cz/id/usnhch. Bakalářská práce. Univerzita Hradec Králové, Fakulta informatiky a managementu. Vedoucí práce Ing. Barbora Tesařová, Ph.D.

[15] Tag Cloud of all Terms on Techopedia.com. *Techopedia - Where Information Technology and Business Meet* [online]. Techopedia, c2017 [cit. 2017-07-29]. Dostupné z: https://www.techopedia.com/dictionary/tags

[16] Online Learning. *What is Access Path* [online]. Online Learning by GeekInterview.com, 2008 [cit. 2017-07-29]. Dostupné z: http://www.learn.geekinterview.com/data-warehouse/dw-basics/what-is-access-path.html

[17] KRCH, David. Partitioning v souvislostech. *Databázový svět – informační portál ze světa databázových technologií* [online]. Brno: AVRE Publishing, 13. 04. 2004 [cit. 2017-07-30]. Dostupné z: http://www.dbsvet.cz/rservice.php?akce=tisk&cisloclanku=2004041302¨

[18] Oracle database 12c: Product Family. *Oracle* [online]. Redwood City: Oracle Corporation, March 2017 [cit. 2017-07-30]. Dostupné z: http://www.oracle.com/technetwork/database/oracle-database-editions-wp-12c-1896124.pdf

[19] ASHDOWN, Lance. Oracle Database SQL Tuning Guide, 12c Release 2 (12.2). *Oracle Help Center* [online]. Redwood City: Oracle Corporation, c2018, March 2018 [cit. 2018-04-28]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tgsql/index.html

[20] HUEY, Patricia. Oracle Database Database Security Guide, 12c Release 2 (12.2). *Oracle Help Center* [online]. Redwood City: Oracle Corporation, c2017, December 2017 [cit. 2018-04-28]. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/dbseg/index.html

[21] Editions and supported features of SQL Server 2016. *Technical documentation, API, and code examples* [online]. Redmond: Microsoft Corporation, 2017, 05/24/2017 [cit. 2018-04-28]. Dostupné z: https://docs.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-2016?view=sql-server-2016

[22] Performance Center for SQL Server Database Engine and Azure SQL Database. *Technical documentation, API, and code examples* [online]. Redmond: Microsoft Corporation, 2016, 04/08/2016 [cit. 2018-04-28]. Dostupné z: https://docs.microsoft.com/en-us/sql/relational-databases/performance/performance-center-for-sql-server-database-engine-and-azure-sql-database?view=sql-server-2016

[23] Technical documentation, API, and code examples. *Choose an Authentication Mode* [online]. Redmond: Microsoft Corporation, 2017, 14. 03. 2017 [cit. 2018-04-28]. Dostupné z: https://docs.microsoft.com/cs-cz/sql/relational-databases/security/choose-an-authentication-mode?view=sql-server-2016

[24] Managing Logins, Users, and Schemas How-to Topics. *Technical documentation, API, and code examples* [online]. Redmond: Microsoft Corporation, 2017, 14. 03. 2017 [cit. 2018-04-28]. Dostupné z: https://docs.microsoft.com/cs-cz/sql/relational-databases/security/authentication-access/managing-logins-users-and-schemas-how-to-topics?view=sql-server-2016

[25] MySQL 5.7 Reference Manual. *MySQL Documentation* [online]. Redwood City: Oracle Corporation, c2018 [cit. 2018-04-28]. Dostupné z: https://dev.mysql.com/doc/refman/5.7/en/

[26] PostgreSQL: About. *PostgreSQL: The world's most advanced open source database* [online]. The PostgreSQL Global Development Group, c2018 [cit. 2018-04-28]. Dostupné z: https://www.postgresql.org/about/

[27] PostgreSQL: Feature Matrix. *PostgreSQL: The world's most advanced open source database* [online]. The PostgreSQL Global Development Group, c2018 [cit. 2018-04-28]. Dostupné z: https://www.postgresql.org/about/featurematrix/

[28] PostgreSQL 10.3 Documentation. *PostgreSQL: The world's most advanced open source database* [online]. The PostgreSQL Global Development Group, c2018 [cit. 2018-04-28]. Dostupné z: https://www.postgresql.org/docs/10/static/index.html

[29] HEUDECKER, Nick, Donald FEINBERG a Merv ADRIAN. Magic Quadrant for Operational Database Management Systems. *Gartner* [online]. Stamford: Gartner, c2017, 02 November 2017 [cit. 2018-04-28]. Dostupné z: https://www.gartner.com/doc/reprints?id=1-4JVFEAM&ct=171103&st=sb&type=PD&publisher=DCK&utm_source=&utm_medium=contentsyn&utm_content=gartner-odbms&utm_campaign=gartner-link

[30] *Data and Analytics Platform Migration - SQLines Open Source Tools* [online]. c2010-2017 [cit. 2018-02-19]. Dostupné z: http://www.sqlines.com/home

[31] GRAND PRIX ENCYCLOPEDIA - DRIVERS. *Latest Formula 1 Breaking News - Grandprix.com* [online]. Palisades: Inside F1, c1988-2018 [cit. 2018-02-18]. Dostupné z: http://www.grandprix.com/gpe/driversa.html

[32] *F1NEWS.CZ - Svět Formule 1* [online]. Praha 7 - Holešovice: F1NEWS.cz, 2018 [cit. 2018-02-18]. Dostupné z: http://f1news.autoroad.cz/

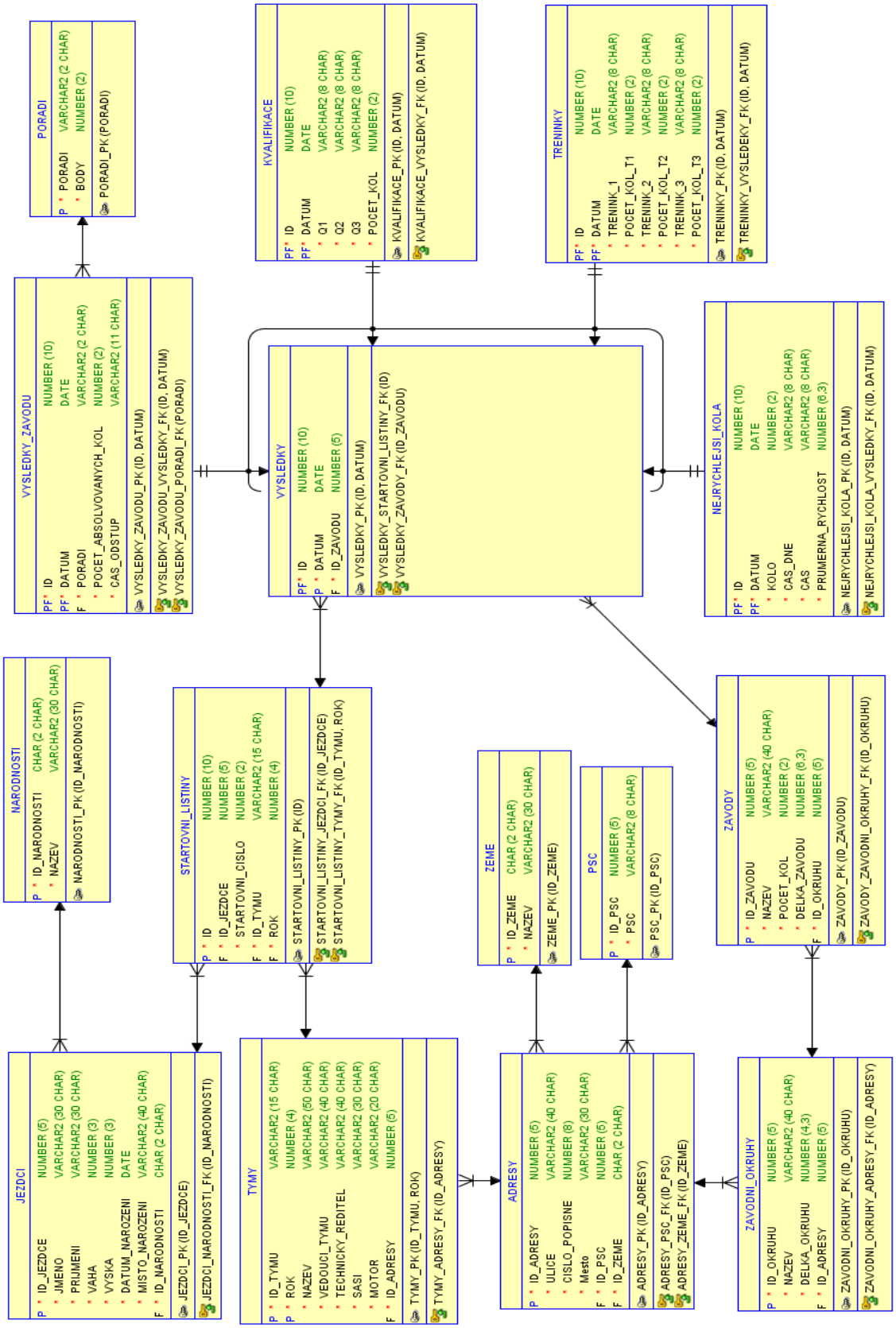
[33] *Formula 1*® [online]. Formula One World Championship Limited, c2003-2018 [cit. 2018-02-18]. Dostupné z: http://www.formula1.com/

[34] KLICNAROVÁ, Jana. Vícekriteriální hodnocení variant – úvod: Ekonomická fakulta JU. *Ekonomická fakulta JU* [online]. České Budějovice: Jihočeská univerzita v Českých Budějovicích, 2010 [cit. 2018-05-07]. Dostupné z: http://home.ef.jcu.cz/~janaklic/oa_zsf/VHV_I.pdf

[35] KALČEVOVÁ, Jana. Kriteriální matice a hodnocení variant. *Výuka* [online]. Praha: Vysoká škola ekonomická v Praze [cit. 2018-05-07]. Dostupné z: http://jana.kalcev.cz/vyuka/kestazeni/EKO422-KriterialniMatice.pdf

SEZNAM PŘÍLOH

Příloha A – *Schéma testovaného databázového modelu*



Příloha B – *Univerzální SQL dotaz č. 1*

```
SELECT ROK, JEZDEC, TYM, MOTOR, SASI, ZAVOD, OKRUH FROM


(SELECT * FROM

(SELECT * FROM

(SELECT CONCAT(JMENO, CONCAT(' ', PRIJMENI)) AS JEZDEC,
TT.NAZEV AS TYM, TT.MOTOR AS MOTOR, TT.SASI AS SASI, SL.ROK,
ZZ.NAZEV AS ZAVOD, ZO.NAZEV AS OKRUH, NK.CAS AS CAS_NEJRYCHLEJSI_KOLO,
KK.Q3 AS CAS_Q3, VV.DATUM AS DATUM_ZAVODU
FROM VYSLEDKY VV
INNER JOIN VYSLEDKY_ZAVODU VZ ON(VV.ID = VZ.ID AND VV.DATUM = VZ.DATUM)
INNER JOIN NEJRYCHLEJSI_KOLA NK ON (VV.ID = NK.ID AND VV.DATUM = NK.DATUM)
INNER JOIN KVALIFIKACE KK ON (VV.ID = KK.ID AND VV.DATUM = KK.DATUM)
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN ZAVODY ZZ ON (VV.ID_ZAVODU = ZZ.ID_ZAVODU)
INNER JOIN ZAVODNI_OKRUHY ZO ON (ZZ.ID_OKRUHU = ZO.ID_OKRUHU)
INNER JOIN TYMY TT ON (SL.ID_TYMU = TT.ID_TYMU AND SL.ROK = TT.ROK)
WHERE VZ.CAS_ODSTUP =
(SELECT VZ2.CAS_ODSTUP
FROM VYSLEDKY_ZAVODU VZ2
INNER JOIN VYSLEDKY VV2 ON (VZ2.ID = VV2.ID AND VZ2.DATUM = VV2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ROK = SL.ROK AND VV2.DATUM = VV.DATUM
AND VZ2.CAS_ODSTUP LIKE '%:%')) VYPOCET_VYHRANEHO_ZAVODU

WHERE VYPOCET_VYHRANEHO_ZAVODU.CAS_Q3 =
(SELECT MIN(KK2.Q3)
FROM KVALIFIKACE KK2
INNER JOIN VYSLEDKY VV2 ON (KK2.ID = VV2.ID AND KK2.DATUM = VV2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ROK = VYPOCET_VYHRANEHO_ZAVODU.ROK
AND VV2.DATUM = VYPOCET_VYHRANEHO_ZAVODU.DATUM_ZAVODU
AND KK2.Q3 NOT LIKE '%0:%')) VYPOCET_VYHRANE_KVALIFIKACE

WHERE VYPOCET_VYHRANE_KVALIFIKACE.CAS_NEJRYCHLEJSI_KOLO =
(SELECT MIN(NK2.CAS)
FROM NEJRYCHLEJSI_KOLA NK2
INNER JOIN VYSLEDKY VV2 ON (NK2.ID = VV2.ID AND NK2.DATUM = VV2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ROK = VYPOCET_VYHRANE_KVALIFIKACE.ROK
AND VV2.DATUM = VYPOCET_VYHRANE_KVALIFIKACE.DATUM_ZAVODU
AND NK2.CAS NOT LIKE '%0:%')) VYPOCET_NEJRYCHLEJSIHO_KOLA


ORDER BY ROK;
```

Příloha C – *Univerzální SQL dotaz č. 2*

```
SELECT JEZDEC, AUSTRALIE, CINA, BAHRAJN, RUSKO, SPANELSKO, MONAKO, KANADA,
AZERBAJDZAN, RAKOUSKO, VELKA_BRITANIE, MADARSKO, BELGIE, ITALIE, SINGAPUR,
MALAJSIE, JAPONSKO, USA, MEXIKO, BRAZILIE, ABU_DHABI FROM


(SELECT CONCAT(JMENO, CONCAT(' ', PRIJMENI)) AS JEZDEC,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '1' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS AUSTRALIE,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '2' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2
ON (VV2.ID = SL2.ID) WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS CINA,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '3' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS BAHRAJN,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '4' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS RUSKO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '5' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS SPANELSKO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '6' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS MONAKO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '7' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
```

```
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS KANADA,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '8' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS AZERBAJDZAN,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '9' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS RAKOUSKO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '10' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS VELKA_BRITANIE,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '11' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS MADARSKO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '12' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS BELGIE,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '13' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS ITALIE,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '14' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS SINGAPUR,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
```

```
AND VV2.ID_ZAVODU = '15' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS MALAJSIE,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '16' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS JAPONSKO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '17' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS USA,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '18' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS MEXIKO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '19' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS BRAZILIE,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND VV2.ID_ZAVODU = '20' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE) AS ABU_DHABI

FROM JEZDCI JJ
INNER JOIN STARTOVNI_LISTINY SL ON (JJ.ID_JEZDCE = SL.ID_JEZDCE)
INNER JOIN TYMY TT
ON (TT.ID_TYMU = SL.ID_TYMU AND TT.ROK = SL.ROK)) VYPOCET


GROUP BY JEZDEC, AUSTRALIE, CINA, BAHRAJN, RUSKO, SPANELSKO, MONAKO,
KANADA, AZERBAJDZAN, RAKOUSKO, VELKA_BRITANIE, MADARSKO, BELGIE, ITALIE,
SINGAPUR, MALAJSIE, JAPONSKO, USA, MEXIKO, BRAZILIE, ABU_DHABI
ORDER BY JEZDEC;
```

Příloha D – *Univerzální SQL dotaz č. 3*

```
SELECT JEZDEC, NARODNOST, VYHRANA_KVALIFIKACE, NEJRYCHLEJSI_KOLO,
PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO, PODIOVE_UMISTENI, BODOVANE_ZAVODY,
DOKONCENE_ZAVODY, NEDOKONCENE_ZAVODY, POCET_KOL, POCET_BODU, POCET_ZAVODU,
POCET_SEZON FROM


(SELECT SL.ID_JEZDCE,
CONCAT(JJ.JMENO,CONCAT(' ',JJ.PRIJMENI)) AS JEZDEC, NN.NAZEV AS NARODNOST,

(SELECT COUNT(KK3.Q3) FROM KVALIFIKACE KK3
INNER JOIN STARTOVNI_LISTINY SL3 ON (KK3.ID = SL3.ID)
INNER JOIN (SELECT KK2.DATUM, MIN(KK2.Q3) AS NEJRYCHLEJSI_KOLO_KVALIFIKACE
FROM KVALIFIKACE KK2
WHERE KK2.Q3 NOT LIKE '%0:%'
GROUP BY KK2.DATUM) VYPOCET_KVALIFIKACE
ON (VYPOCET_KVALIFIKACE.DATUM = KK3.DATUM)
WHERE SL3.ID_JEZDCE = SL.ID_JEZDCE
AND KK3.Q3 = VYPOCET_KVALIFIKACE.NEJRYCHLEJSI_KOLO_KVALIFIKACE
GROUP BY SL3.ID_JEZDCE) AS VYHRANA_KVALIFIKACE,

(SELECT COUNT(NK3.CAS) FROM NEJRYCHLEJSI_KOLA NK3
INNER JOIN STARTOVNI_LISTINY SL3 ON (NK3.ID = SL3.ID)
INNER JOIN (SELECT KK2.DATUM, MIN(KK2.CAS) AS NEJRYCHLEJSI_KOLO_ZAVODU
FROM NEJRYCHLEJSI_KOLA KK2
WHERE KK2.CAS NOT LIKE '%0:%'
GROUP BY KK2.DATUM) VYPOCET_NEJRYCHLEJSIHO_KOLA
ON (VYPOCET_NEJRYCHLEJSIHO_KOLA.DATUM = NK3.DATUM)
WHERE SL3.ID_JEZDCE = SL.ID_JEZDCE
AND NK3.CAS = VYPOCET_NEJRYCHLEJSIHO_KOLA.NEJRYCHLEJSI_KOLO_ZAVODU
GROUP BY SL3.ID_JEZDCE) AS NEJRYCHLEJSI_KOLO,

COUNT(CASE WHEN VZ.PORADI = '1' THEN VZ.PORADI END) AS PRVNI_MISTO,
COUNT(CASE WHEN VZ.PORADI = '2' THEN VZ.PORADI END) AS DRUHE_MISTO,
COUNT(CASE WHEN VZ.PORADI = '3' THEN VZ.PORADI END) AS TRETI_MISTO,
COUNT(CASE VZ.PORADI WHEN '1' THEN VZ.PORADI WHEN '2' THEN VZ.PORADI WHEN
'3' THEN VZ.PORADI END) AS PODIOVE_UMISTENI,
COUNT(CASE WHEN VZ.PORADI IN ('1','2','3','4','5','6','7','8','9','10')
THEN VZ.PORADI END) AS BODOVANE_ZAVODY,
COUNT(CASE WHEN VZ.PORADI <> 'NC' THEN VZ.PORADI END) AS DOKONCENE_ZAVODY,
COUNT(CASE WHEN VZ.PORADI = 'NC' THEN VZ.PORADI END) AS NEDOKONCENE_ZAVODY,
SUM(VZ.POCET_ABSOLVOVANYCH_KOL) AS POCET_KOL,
SUM(PP.BODY) AS POCET_BODU,
COUNT(DISTINCT VV.DATUM) AS POCET_ZAVODU,
COUNT(DISTINCT SL.ROK) AS POCET_SEZON
FROM VYSLEDKY VV
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN VYSLEDKY_ZAVODU VZ ON(VV.ID = VZ.ID AND VV.DATUM = VZ.DATUM)
INNER JOIN KVALIFIKACE KK ON(VV.ID = KK.ID AND VV.DATUM = KK.DATUM)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN PORADI PP ON (VZ.PORADI = PP.PORADI)
INNER JOIN NARODNOSTI NN ON (JJ.ID_NARODNOSTI = NN.ID_NARODNOSTI)
GROUP BY SL.ID_JEZDCE, JJ.JMENO, JJ.PRIJMENI, NN.NAZEV) HLAVNI_VYPOCET


ORDER BY JEZDEC;
```

Příloha E – *Univerzální SQL dotaz č. 4*

```
SELECT CONCAT(JMENO, CONCAT(' ', PRIJMENI)) AS JEZDEC,
COUNT(CASE WHEN STARTOVNI_POZICE = '1' THEN Q3 END) AS STARTOVNI_POZICE_1,
COUNT(CASE WHEN STARTOVNI_POZICE = '2' THEN Q3 END) AS STARTOVNI_POZICE_2,
COUNT(CASE WHEN STARTOVNI_POZICE = '3' THEN Q3 END) AS STARTOVNI_POZICE_3,
COUNT(CASE WHEN STARTOVNI_POZICE = '4' THEN Q3 END) AS STARTOVNI_POZICE_4,
COUNT(CASE WHEN STARTOVNI_POZICE = '5' THEN Q3 END) AS STARTOVNI_POZICE_5,
COUNT(CASE WHEN STARTOVNI_POZICE = '6' THEN Q3 END) AS STARTOVNI_POZICE_6,
COUNT(CASE WHEN STARTOVNI_POZICE = '7' THEN Q3 END) AS STARTOVNI_POZICE_7,
COUNT(CASE WHEN STARTOVNI_POZICE = '8' THEN Q3 END) AS STARTOVNI_POZICE_8,
COUNT(CASE WHEN STARTOVNI_POZICE = '9' THEN Q3 END) AS STARTOVNI_POZICE_9,
COUNT(CASE WHEN STARTOVNI_POZICE = '10' THEN Q3 END) AS STARTOVNI_POZICE_10
FROM


(SELECT JJ.ID_JEZDCE, JJ.JMENO, JJ.PRIJMENI, ZZ.NAZEV, KK.Q3,

(SELECT COUNT(*)-9 FROM KVALIFIKACE KK2
INNER JOIN STARTOVNI_LISTINY SL2 ON (KK2.ID = SL2.ID)
WHERE SL2.ROK = SL.ROK AND KK2.DATUM = KK.DATUM
AND KK.Q3 LIKE '%1:%' AND KK.Q3 > KK2.Q3) AS STARTOVNI_POZICE

FROM VYSLEDKY VV
INNER JOIN KVALIFIKACE KK ON (VV.ID = KK.ID AND VV.DATUM = KK.DATUM)
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN ZAVODY ZZ ON (VV.ID_ZAVODU = ZZ.ID_ZAVODU)
INNER JOIN ZAVODNI_OKRUHY ZO ON (ZZ.ID_OKRUHU = ZO.ID_OKRUHU)
WHERE KK.Q3 LIKE '%1:%') VYPOCET_PORADI


GROUP BY JMENO, PRIJMENI
ORDER BY 2 DESC, 3 DESC, 4 DESC, 5 DESC, 6 DESC,
7 DESC, 8 DESC, 9 DESC, 10 DESC, 11 DESC;
```

Příloha F – *Univerzální SQL dotaz č. 5*

```
SELECT NARODNOST,

(SELECT COUNT(*) FROM JEZDCI JJ2
INNER JOIN NARODNOSTI NN2 ON (JJ2.ID_NARODNOSTI = NN2.ID_NARODNOSTI)
WHERE NN2.NAZEV = NARODNOST) AS POCET_JEZDCU,

SUM(VYHRANA_KVALIFIKACE) AS VYHRANA_KVALIFIKACE,
SUM(NEJRYCHLEJSI_KOLO) AS NEJRYCHLEJSI_KOLO,
SUM(PRVNI_MISTO) AS PRVNI_MISTO,
SUM(DRUHE_MISTO) AS DRUHE_MISTO,
SUM(TRETI_MISTO) AS TRETI_MISTO,
VYPOCET_NARODNOST.PODIOVE_ZAVODY,
VYPOCET_NARODNOST.BODOVANE_ZAVODY,
VYPOCET_NARODNOST.DOKONCENE_ZAVODY,
VYPOCET_NARODNOST.NEDOKONCENE_ZAVODY,
SUM(HLAVNI_VYPOCET.POCET_KOL) AS POCET_KOL,
SUM(HLAVNI_VYPOCET.POCET_BODU) AS POCET_BODU,
VYPOCET_NARODNOST.POCET_ZAVODU,

(SELECT SUM(CASE WHEN POCET_SEZON > 0 THEN 1 END)
FROM
(SELECT NN2.NAZEV, COUNT(DISTINCT SL2.ROK) AS POCET_SEZON, SL2.ROK
FROM VYSLEDKY_ZAVODU VZ2
INNER JOIN STARTOVNI_LISTINY SL2 ON (VZ2.ID = SL2.ID)
INNER JOIN JEZDCI JJ2 ON (SL2.ID_JEZDCE = JJ2.ID_JEZDCE)
INNER JOIN NARODNOSTI NN2 ON (JJ2.ID_NARODNOSTI = NN2.ID_NARODNOSTI)
GROUP BY NN2.NAZEV, SL2.ROK) VYPOCET_POCTU_SEZON
WHERE VYPOCET_POCTU_SEZON.NAZEV = HLAVNI_VYPOCET.NARODNOST) AS POCET_SEZON
FROM


(SELECT SL.ID_JEZDCE,
CONCAT(JJ.JMENO, CONCAT(' ', JJ.PRIJMENI)) AS JEZDEC,
NN.NAZEV AS NARODNOST,

(SELECT COUNT(KK3.Q3)
FROM KVALIFIKACE KK3
INNER JOIN STARTOVNI_LISTINY SL3 ON (KK3.ID = SL3.ID)
INNER JOIN (SELECT KK2.DATUM, MIN(KK2.Q3) AS NEJRYCHLEJSI_KOLO_KVALIFIKACE
FROM KVALIFIKACE KK2
WHERE KK2.Q3 NOT LIKE '%0:%'
GROUP BY KK2.DATUM) VYPOCET_KVALIFIKACE
ON (VYPOCET_KVALIFIKACE.DATUM = KK3.DATUM)
WHERE SL3.ID_JEZDCE = SL.ID_JEZDCE
AND KK3.Q3 = VYPOCET_KVALIFIKACE.NEJRYCHLEJSI_KOLO_KVALIFIKACE
GROUP BY SL3.ID_JEZDCE) AS VYHRANA_KVALIFIKACE,

(SELECT COUNT(NK3.CAS) FROM NEJRYCHLEJSI_KOLA NK3
INNER JOIN STARTOVNI_LISTINY SL3 ON (NK3.ID = SL3.ID)
INNER JOIN (SELECT KK2.DATUM, MIN(KK2.CAS) AS NEJRYCHLEJSI_KOLO_ZAVODU
FROM NEJRYCHLEJSI_KOLA KK2
WHERE KK2.CAS NOT LIKE '%0:%'
GROUP BY KK2.DATUM) VYPOCET_NEJRYCHLEJSIHO_KOLA
ON (VYPOCET_NEJRYCHLEJSIHO_KOLA.DATUM = NK3.DATUM)
WHERE SL3.ID_JEZDCE = SL.ID_JEZDCE
AND NK3.CAS = VYPOCET_NEJRYCHLEJSIHO_KOLA.NEJRYCHLEJSI_KOLO_ZAVODU
GROUP BY SL3.ID_JEZDCE) AS NEJRYCHLEJSI_KOLO,
```

```
COUNT(CASE WHEN VZ.PORADI = '1' THEN VZ.PORADI END) AS PRVNI_MISTO,
COUNT(CASE WHEN VZ.PORADI = '2' THEN VZ.PORADI END) AS DRUHE_MISTO,
COUNT(CASE WHEN VZ.PORADI = '3' THEN VZ.PORADI END) AS TRETI_MISTO,
SUM(VZ.POCET_ABSOLVOVANYCH_KOL) AS POCET_KOL,
SUM(PP.BODY) AS POCET_BODU
FROM VYSLEDKY VV
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN VYSLEDKY_ZAVODU VZ ON(VV.ID = VZ.ID AND VV.DATUM = VZ.DATUM)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN PORADI PP ON (VZ.PORADI = PP.PORADI)
INNER JOIN NARODNOSTI NN ON (JJ.ID_NARODNOSTI = NN.ID_NARODNOSTI)
GROUP BY SL.ID_JEZDCE, JJ.JMENO, JJ.PRIJMENI, NN.NAZEV) HLAVNI_VYPOCET


INNER JOIN (SELECT NAZEV,
SUM(CASE WHEN PODIOVE_UMISTENI > 0 THEN 1 END) AS PODIOVE_ZAVODY,
SUM(CASE WHEN BODOVANE_ZAVODY > 0 THEN 1 END) AS BODOVANE_ZAVODY,
SUM(CASE WHEN DOKONCENE_ZAVODY > 0 THEN 1 END) AS DOKONCENE_ZAVODY,
SUM(CASE WHEN NEDOKONCENE_ZAVODY = POCET_ZAVODNIKU_V_ZAVODE THEN 1 END) AS
NEDOKONCENE_ZAVODY,
SUM(CASE WHEN POCET_ZAVODU > 0 THEN 1 END) AS POCET_ZAVODU
FROM
(SELECT NN2.NAZEV, COUNT(CASE VZ2.PORADI WHEN '1' THEN VZ2.PORADI WHEN '2'
THEN VZ2.PORADI WHEN '3' THEN VZ2.PORADI END) AS PODIOVE_UMISTENI,
COUNT(CASE WHEN VZ2.PORADI IN ('1', '2', '3', '4', '5', '6', '7', '8', '9',
'10') THEN VZ2.PORADI END) AS BODOVANE_ZAVODY,
COUNT(CASE WHEN VZ2.PORADI <> 'NC' THEN VZ2.PORADI END) AS
DOKONCENE_ZAVODY,
COUNT(CASE WHEN VZ2.PORADI = 'NC' THEN VZ2.PORADI END) AS
NEDOKONCENE_ZAVODY,
COUNT(DISTINCT VV2.DATUM) AS POCET_ZAVODU,
COUNT(VZ2.PORADI) AS POCET_ZAVODNIKU_V_ZAVODE, VZ2.DATUM
FROM VYSLEDKY VV2
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON(VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN JEZDCI JJ2 ON (SL2.ID_JEZDCE = JJ2.ID_JEZDCE)
INNER JOIN NARODNOSTI NN2 ON (JJ2.ID_NARODNOSTI = NN2.ID_NARODNOSTI)
GROUP BY NN2.NAZEV, VZ2.DATUM)
VYPOCET_ZAVODY GROUP BY NAZEV)
VYPOCET_NARODNOST ON (HLAVNI_VYPOCET.NARODNOST = VYPOCET_NARODNOST.NAZEV)

GROUP BY NARODNOST, VYPOCET_NARODNOST.PODIOVE_ZAVODY,
VYPOCET_NARODNOST.BODOVANE_ZAVODY,
VYPOCET_NARODNOST.DOKONCENE_ZAVODY,
VYPOCET_NARODNOST.NEDOKONCENE_ZAVODY,
VYPOCET_NARODNOST.POCET_ZAVODU
ORDER BY POCET_JEZDCU DESC;
```

Příloha G – *Univerzální SQL dotaz č. 6*

```
SELECT ROK, STARTOVNI_CISLO, JEZDEC, TYM, MOTOR, SASI, BODY, PRVNI_MISTO,
DRUHE_MISTO, TRETI_MISTO, CTVRTE_MISTO, PATE_MISTO FROM


(SELECT ID_JEZDCE, JEZDEC, ROK, STARTOVNI_CISLO, TYM, MOTOR, SASI, BODY,
PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO, CTVRTE_MISTO, PATE_MISTO,
MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST, MAXIMUM_DRUHYCH_MIST,
MAXIMUM_TRETICH_MIST, MAXIMUM_CTVRTYCH_MIST,
(SELECT MAX(POCET_PATYCH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS
POCET_DRUHYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '3' THEN PP3.PORADI END) AS
POCET_TRETICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '4' THEN PP3.PORADI END) AS
POCET_CTVRTYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '5' THEN PP3.PORADI END) AS POCET_PATYCH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID, SL3.ROK) VYPOCET
ON (VYPOCET.ID = SL2.ID AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_CTVRTYCH_MIST.ROK
AND VYPOCET.BODY = VYPOCET_CTVRTYCH_MIST.MAXIMUM_BODU
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_CTVRTYCH_MIST.MAXIMUM_PRVNICH_MIST
AND VYPOCET.POCET_DRUHYCH_MIST = VYPOCET_CTVRTYCH_MIST.MAXIMUM_DRUHYCH_MIST
AND VYPOCET.POCET_TRETICH_MIST = VYPOCET_CTVRTYCH_MIST.MAXIMUM_TRETICH_MIST
AND VYPOCET.POCET_CTVRTYCH_MIST =
VYPOCET_CTVRTYCH_MIST.MAXIMUM_CTVRTYCH_MIST) AS MAXIMUM_PATYCH_MIST
FROM

(SELECT ID_JEZDCE, JEZDEC, ROK, STARTOVNI_CISLO, TYM, MOTOR, SASI, BODY,
PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO, CTVRTE_MISTO, PATE_MISTO,
MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST, MAXIMUM_DRUHYCH_MIST,
MAXIMUM_TRETICH_MIST,
(SELECT MAX(POCET_CTVRTYCH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS
POCET_DRUHYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '3' THEN PP3.PORADI END) AS
POCET_TRETICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '4' THEN PP3.PORADI END) AS
POCET_CTVRTYCH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID, SL3.ROK) VYPOCET
ON (VYPOCET.ID = SL2.ID AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_TRETICH_MIST.ROK
AND VYPOCET.BODY = VYPOCET_TRETICH_MIST.MAXIMUM_BODU
```

```
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_TRETICH_MIST.MAXIMUM_PRVNICH_MIST
AND VYPOCET.POCET_DRUHYCH_MIST = VYPOCET_TRETICH_MIST.MAXIMUM_DRUHYCH_MIST
AND VYPOCET.POCET_TRETICH_MIST = VYPOCET_TRETICH_MIST.MAXIMUM_TRETICH_MIST)
AS MAXIMUM_CTVRTYCH_MIST FROM

(SELECT ID_JEZDCE, JEZDEC, ROK, STARTOVNI_CISLO, TYM, MOTOR, SASI, BODY,
PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO, CTVRTE_MISTO, PATE_MISTO,
MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST, MAXIMUM_DRUHYCH_MIST,
(SELECT MAX(POCET_TRETICH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS
POCET_DRUHYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '3' THEN PP3.PORADI END) AS POCET_TRETICH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID, SL3.ROK) VYPOCET
ON (VYPOCET.ID = SL2.ID AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_DRUHYCH_MIST.ROK
AND VYPOCET.BODY = VYPOCET_DRUHYCH_MIST.MAXIMUM_BODU
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_DRUHYCH_MIST.MAXIMUM_PRVNICH_MIST
AND VYPOCET.POCET_DRUHYCH_MIST = VYPOCET_DRUHYCH_MIST.MAXIMUM_DRUHYCH_MIST)
AS MAXIMUM_TRETICH_MIST FROM

(SELECT ID_JEZDCE, JEZDEC, ROK, STARTOVNI_CISLO, TYM, MOTOR, SASI, BODY,
PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO, CTVRTE_MISTO, PATE_MISTO,
MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST,
(SELECT MAX(POCET_DRUHYCH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS POCET_DRUHYCH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID, SL3.ROK) VYPOCET
ON (VYPOCET.ID = SL2.ID AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_PRVNICH_MIST.ROK
AND VYPOCET.BODY = VYPOCET_PRVNICH_MIST.MAXIMUM_BODU
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_PRVNICH_MIST.MAXIMUM_PRVNICH_MIST)
AS MAXIMUM_DRUHYCH_MIST FROM

(SELECT ID_JEZDCE, JEZDEC, ROK, STARTOVNI_CISLO, TYM, MOTOR, SASI, BODY,
PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO, CTVRTE_MISTO, PATE_MISTO,
MAXIMUM_BODU,
(SELECT MAX(POCET_PRVNICH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS POCET_PRVNICH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID, SL3.ROK) VYPOCET
ON (VYPOCET.ID = SL2.ID AND VYPOCET.ROK = SL2.ROK)
```

```
WHERE SL2.ROK = VYPOCET_MAXIMUM_BODU.ROK
AND VYPOCET.BODY = VYPOCET_MAXIMUM_BODU.MAXIMUM_BODU) AS
MAXIMUM_PRVNICH_MIST FROM

(SELECT SL.ID_JEZDCE AS ID_JEZDCE,
CONCAT(JJ.JMENO, CONCAT(' ', JJ.PRIJMENI)) AS JEZDEC,
SL.ROK, SL.STARTOVNI_CISLO, TT.NAZEV AS TYM, TT.MOTOR, TT.SASI,
SUM(PP.BODY) AS BODY,
COUNT(CASE VZ.PORADI WHEN '1' THEN VZ.PORADI END) AS PRVNI_MISTO,
COUNT(CASE VZ.PORADI WHEN '2' THEN VZ.PORADI END) AS DRUHE_MISTO,
COUNT(CASE VZ.PORADI WHEN '3' THEN VZ.PORADI END) AS TRETI_MISTO,
COUNT(CASE VZ.PORADI WHEN '4' THEN VZ.PORADI END) AS CTVRTE_MISTO,
COUNT(CASE VZ.PORADI WHEN '5' THEN VZ.PORADI END) AS PATE_MISTO,
(SELECT MAX(VYPOCET.BODY) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID, SL3.ROK, SUM(PP3.BODY) AS BODY
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID, SL3.ROK) VYPOCET
ON (VYPOCET.ID = SL2.ID AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = SL.ROK) AS MAXIMUM_BODU
FROM VYSLEDKY VV
INNER JOIN VYSLEDKY_ZAVODU VZ ON (VV.ID = VZ.ID AND VV.DATUM = VZ.DATUM)
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN TYMY TT ON (SL.ID_TYMU = TT.ID_TYMU AND SL.ROK = TT.ROK)
INNER JOIN PORADI PP ON (VZ.PORADI = PP.PORADI)
GROUP BY SL.ID_JEZDCE, JEZDEC, SL.ROK, SL.STARTOVNI_CISLO, TT.NAZEV,
TT.MOTOR, TT.SASI) VYPOCET_MAXIMUM_BODU)

VYPOCET_PRVNICH_MIST)

VYPOCET_DRUHYCH_MIST)

VYPOCET_TRETICH_MIST)

VYPOCET_CTVRTYCH_MIST)

VYPOCET_HLAVNI


WHERE VYPOCET_HLAVNI.BODY = VYPOCET_HLAVNI.MAXIMUM_BODU
AND VYPOCET_HLAVNI.PRVNI_MISTO = VYPOCET_HLAVNI.MAXIMUM_PRVNICH_MIST
AND VYPOCET_HLAVNI.DRUHE_MISTO = VYPOCET_HLAVNI.MAXIMUM_DRUHYCH_MIST
AND VYPOCET_HLAVNI.TRETI_MISTO = VYPOCET_HLAVNI.MAXIMUM_TRETICH_MIST
AND VYPOCET_HLAVNI.CTVRTE_MISTO = VYPOCET_HLAVNI.MAXIMUM_CTVRTYCH_MIST
AND VYPOCET_HLAVNI.PATE_MISTO = VYPOCET_HLAVNI.MAXIMUM_PATYCH_MIST
ORDER BY ROK;
```

Příloha H – *Univerzální SQL dotaz č. 7*

```
SELECT JEZDEC, FERRARI, FORCE_INDIA, HAAS, MCLAREN, MERCEDES, RED_BULL,
RENAULT, SAUBER, TORO_ROSSO, WILLIAMS FROM


(SELECT JJ.ID_JEZDCE,
CONCAT(JJ.JMENO, CONCAT(' ', JJ.PRIJMENI)) as JEZDEC,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Mclaren' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS MCLAREN,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Ferrari' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2
ON (VV2.ID = SL2.ID) WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS FERRARI,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Mercedes' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS MERCEDES,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Red Bull' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS RED_BULL,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Haas' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS HAAS ,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Force India' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
```

```
GROUP BY SL2.ID_JEZDCE) AS FORCE_INDIA,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Sauber' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS SAUBER,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Toro Rosso' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS TORO_ROSSO,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Williams' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS WILLIAMS,

(SELECT COUNT(CASE WHEN VZ2.PORADI = '1'
AND SL2.ID_TYMU = 'Renault' THEN VZ2.PORADI END)
FROM VYSLEDKY VV2
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON (VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
WHERE SL2.ID_JEZDCE = SL.ID_JEZDCE
GROUP BY SL2.ID_JEZDCE) AS RENAULT

FROM JEZDCI JJ
INNER JOIN STARTOVNI_LISTINY SL ON (JJ.ID_JEZDCE = SL.ID_JEZDCE)
INNER JOIN TYMY TT ON (TT.ID_TYMU = SL.ID_TYMU AND TT.ROK = SL.ROK)
INNER JOIN VYSLEDKY VV ON (VV.ID = SL.ID)
INNER JOIN VYSLEDKY_ZAVODU VZ
ON (VZ.ID = VV.ID AND VZ.DATUM = VV.DATUM)) VYPOCET


GROUP BY JEZDEC, FERRARI, FORCE_INDIA, HAAS, MCLAREN, MERCEDES, RED_BULL,
RENAULT, SAUBER, TORO_ROSSO, WILLIAMS
ORDER BY JEZDEC;
```

Příloha CH – *Univerzální SQL dotaz č. 8*

```
SELECT ROK, NAZEV_ZAVODU, NAZEV_OKRUHU, ZEME_OKRUHU, NAZEV_TYMU, ZEME_TYMU,
JEZDEC, NARODNOST_PILOTA, PORADI_V_ZAVODE FROM


(SELECT SL.ROK, ZZ.NAZEV AS NAZEV_ZAVODU, ZO.NAZEV AS NAZEV_OKRUHU,
ZETT.NAZEV AS ZEME_OKRUHU, TT.NAZEV AS NAZEV_TYMU, ZETT.NAZEV AS ZEME_TYMU,
CONCAT(JJ.JMENO, CONCAT(' ', JJ.PRIJMENI)) AS JEZDEC,
NN.NAZEV AS NARODNOST_PILOTA, VZ.PORADI AS PORADI_V_ZAVODE
FROM VYSLEDKY VV
INNER JOIN VYSLEDKY_ZAVODU VZ ON(VV.ID = VZ.ID AND VV.DATUM = VZ.DATUM)
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN NARODNOSTI NN ON (JJ.ID_NARODNOSTI = NN.ID_NARODNOSTI)
INNER JOIN TYMY TT ON (SL.ID_TYMU = TT.ID_TYMU AND SL.ROK = TT.ROK)
INNER JOIN ADRESY AATT ON (TT.ID_ADRESY = AATT.ID_ADRESY)
INNER JOIN ZEME ZETT ON (AATT.ID_ZEME = ZETT.ID_ZEME)
INNER JOIN ZAVODY ZZ ON (VV.ID_ZAVODU = ZZ.ID_ZAVODU)
INNER JOIN ZAVODNI_OKRUHY ZO ON (ZZ.ID_OKRUHU = ZO.ID_OKRUHU)
INNER JOIN ADRESY AA ON (ZO.ID_ADRESY = AA.ID_ADRESY)
INNER JOIN ZEME ZE ON (AA.ID_ZEME = ZE.ID_ZEME)
WHERE ZETT.NAZEV =
(SELECT DISTINCT ZT.NAZEV
FROM TYMY TT2
INNER JOIN ADRESY TA ON (TT2.ID_ADRESY = TA.ID_ADRESY)
INNER JOIN ZEME ZT ON (TA.ID_ZEME = ZT.ID_ZEME)
WHERE TT2.ID_TYMU = TT.ID_TYMU AND TT2.ROK = TT.ROK)

AND ZETT.NAZEV =
(SELECT ZZTT2.NAZEV FROM JEZDCI JJ2
INNER JOIN NARODNOSTI NN2 ON (JJ2.ID_NARODNOSTI = NN2.ID_NARODNOSTI)
INNER JOIN ZEME ZZTT2 ON (JJ2.ID_NARODNOSTI = ZZTT2.ID_ZEME)
WHERE JJ2.JMENO = JJ.JMENO AND JJ2.PRIJMENI = JJ.PRIJMENI)

AND ZETT.NAZEV =
(SELECT DISTINCT ZT.NAZEV
FROM ZAVODNI_OKRUHY ZO2
INNER JOIN ADRESY TA ON (ZO2.ID_ADRESY = TA.ID_ADRESY)
INNER JOIN ZEME ZT ON (TA.ID_ZEME = ZT.ID_ZEME)
WHERE ZO2.NAZEV = ZO.NAZEV)

AND VZ.CAS_ODSTUP =
(SELECT VZ2.CAS_ODSTUP FROM VYSLEDKY_ZAVODU VZ2 INNER JOIN VYSLEDKY VV2 ON
(VZ2.ID = VV2.ID AND VZ2.DATUM = VV2.DATUM) WHERE VV2.DATUM = VV.DATUM AND
VZ2.CAS_ODSTUP LIKE '%:%')) VYPOCET_HLAVNI


ORDER BY ROK;
```

Příloha I – *Univerzální SQL dotaz č. 9*

```
SELECT HLAVNI_VYPOCET.TYM, HLAVNI_VYPOCET.ZEME,
SUM(VYHRANA_KVALIFIKACE) AS VYHRANA_KVALIFIKACE,
SUM(NEJRYCHLEJSI_KOLO) AS NEJRYCHLEJSI_KOLO,
SUM(PRVNI_MISTO) AS PRVNI_MISTO, SUM(DRUHE_MISTO) AS DRUHE_MISTO,
SUM(TRETI_MISTO) AS TRETI_MISTO, VYPOCET_TYMY.PODIOVE_ZAVODY,
VYPOCET_TYMY.BODOVANE_ZAVODY, VYPOCET_TYMY.DOKONCENE_ZAVODY,
VYPOCET_TYMY.NEDOKONCENE_ZAVODY,
SUM(HLAVNI_VYPOCET.POCET_KOL) AS POCET_KOL,
SUM(HLAVNI_VYPOCET.POCET_BODU) AS POCET_BODU, VYPOCET_TYMY.POCET_ZAVODU,

(SELECT SUM(CASE WHEN POCET_SEZON > 0 THEN 1 END) FROM
(SELECT TT2.ID_TYMU AS TYM, COUNT(DISTINCT SL2.ROK) AS POCET_SEZON, SL2.ROK
FROM VYSLEDKY VV2
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
INNER JOIN KVALIFIKACE KK2 ON(VV2.ID = KK2.ID AND VV2.DATUM = KK2.DATUM)
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON(VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN PORADI PP2 ON (VZ2.PORADI = PP2.PORADI)
INNER JOIN TYMY TT2 ON (SL2.ID_TYMU = TT2.ID_TYMU AND SL2.ROK = TT2.ROK)
INNER JOIN ADRESY AA2 ON (TT2.ID_ADRESY = AA2.ID_ADRESY)
INNER JOIN ZEME ZZ2 ON (AA2.ID_ZEME = ZZ2.ID_ZEME)
GROUP BY TT2.ID_TYMU, SL2.ROK) VYPOCET_POCTU_SEZON
WHERE VYPOCET_POCTU_SEZON.TYM = HLAVNI_VYPOCET.TYM) AS POCET_SEZON FROM


(SELECT SL.ID_JEZDCE,
CONCAT(JJ.JMENO, CONCAT(' ', JJ.PRIJMENI)) AS JEZDEC,
SL.ID_TYMU AS TYM, ZZ.NAZEV AS ZEME,

(SELECT COUNT(KK3.Q3) FROM KVALIFIKACE KK3
INNER JOIN STARTOVNI_LISTINY SL3 ON (KK3.ID = SL3.ID)
INNER JOIN (SELECT KK2.DATUM, MIN(KK2.Q3) AS NEJRYCHLEJSI_KOLO_KVALIFIKACE
FROM KVALIFIKACE KK2
WHERE KK2.Q3 NOT LIKE '%0:%'
GROUP BY KK2.DATUM) VYPOCET_KVALIFIKACE
ON (VYPOCET_KVALIFIKACE.DATUM = KK3.DATUM)
WHERE SL3.ID_JEZDCE = SL.ID_JEZDCE
AND SL3.ID_TYMU = SL.ID_TYMU
AND KK3.Q3 = VYPOCET_KVALIFIKACE.NEJRYCHLEJSI_KOLO_KVALIFIKACE
GROUP BY SL3.ID_TYMU) AS VYHRANA_KVALIFIKACE,

(SELECT COUNT(NK3.CAS) FROM NEJRYCHLEJSI_KOLA NK3
INNER JOIN STARTOVNI_LISTINY SL3 ON (NK3.ID = SL3.ID)
INNER JOIN (SELECT KK2.DATUM, MIN(KK2.CAS) AS NEJRYCHLEJSI_KOLO_ZAVODU
FROM NEJRYCHLEJSI_KOLA KK2
WHERE KK2.CAS NOT LIKE '%0:%'
GROUP BY KK2.DATUM) VYPOCET_NEJRYCHLEJSIHO_KOLA
ON (VYPOCET_NEJRYCHLEJSIHO_KOLA.DATUM = NK3.DATUM)
WHERE SL3.ID_JEZDCE = SL.ID_JEZDCE
AND SL3.ID_TYMU = SL.ID_TYMU
AND NK3.CAS = VYPOCET_NEJRYCHLEJSIHO_KOLA.NEJRYCHLEJSI_KOLO_ZAVODU
GROUP BY SL3.ID_TYMU) AS NEJRYCHLEJSI_KOLO,

COUNT(CASE WHEN VZ.PORADI = '1' THEN VZ.PORADI END) AS PRVNI_MISTO,
COUNT(CASE WHEN VZ.PORADI = '2' THEN VZ.PORADI END) AS DRUHE_MISTO,
COUNT(CASE WHEN VZ.PORADI = '3' THEN VZ.PORADI END) AS TRETI_MISTO,
SUM(VZ.POCET_ABSOLVOVANYCH_KOL) AS POCET_KOL,
SUM(PP.BODY) AS POCET_BODU
FROM VYSLEDKY VV
```

```
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN VYSLEDKY_ZAVODU VZ ON(VV.ID = VZ.ID AND VV.DATUM = VZ.DATUM)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN PORADI PP ON (VZ.PORADI = PP.PORADI)
INNER JOIN TYMY TT ON (SL.ID_TYMU = TT.ID_TYMU AND SL.ROK = TT.ROK)
INNER JOIN ADRESY AA ON (TT.ID_ADRESY = AA.ID_ADRESY)
INNER JOIN ZEME ZZ ON (AA.ID_ZEME = ZZ.ID_ZEME)
GROUP BY SL.ID_JEZDCE, JJ.JMENO, JJ.PRIJMENI,
SL.ID_TYMU, ZZ.NAZEV) HLAVNI_VYPOCET


INNER JOIN
(SELECT TYM,
SUM(CASE WHEN PODIOVE_UMISTENI > 0 THEN 1 END) AS PODIOVE_ZAVODY,
SUM(CASE WHEN BODOVANE_ZAVODY > 0 THEN 1 END) AS BODOVANE_ZAVODY,
SUM(CASE WHEN DOKONCENE_ZAVODY > 0 THEN 1 END) AS DOKONCENE_ZAVODY,
SUM(CASE WHEN NEDOKONCENE_ZAVODY = POCET_ZAVODNIKU_V_ZAVODE THEN 1 END) AS
NEDOKONCENE_ZAVODY,
SUM(CASE WHEN POCET_ZAVODU > 0 THEN 1 END) AS POCET_ZAVODU FROM
(SELECT TT2.ID_TYMU AS TYM,
COUNT(CASE VZ2.PORADI WHEN '1' THEN VZ2.PORADI WHEN '2' THEN VZ2.PORADI
WHEN '3' THEN VZ2.PORADI END) AS PODIOVE_UMISTENI,
COUNT(CASE WHEN VZ2.PORADI IN ('1', '2', '3', '4', '5', '6', '7', '8', '9',
'10') THEN VZ2.PORADI END) AS BODOVANE_ZAVODY,
COUNT(CASE WHEN VZ2.PORADI <> 'NC' THEN VZ2.PORADI END) AS
DOKONCENE_ZAVODY,
COUNT(CASE WHEN VZ2.PORADI = 'NC' THEN VZ2.PORADI END) AS
NEDOKONCENE_ZAVODY,
COUNT(DISTINCT VZ2.DATUM) AS POCET_ZAVODU,
COUNT(VZ2.PORADI) AS POCET_ZAVODNIKU_V_ZAVODE, VZ2.DATUM
FROM VYSLEDKY VV2
INNER JOIN STARTOVNI_LISTINY SL2 ON (VV2.ID = SL2.ID)
INNER JOIN KVALIFIKACE KK2 ON(VV2.ID = KK2.ID AND VV2.DATUM = KK2.DATUM)
INNER JOIN VYSLEDKY_ZAVODU VZ2
ON(VV2.ID = VZ2.ID AND VV2.DATUM = VZ2.DATUM)
INNER JOIN PORADI PP2 ON (VZ2.PORADI = PP2.PORADI)
INNER JOIN TYMY TT2 ON (SL2.ID_TYMU = TT2.ID_TYMU AND SL2.ROK = TT2.ROK)
INNER JOIN ADRESY AA2 ON (TT2.ID_ADRESY = AA2.ID_ADRESY)
INNER JOIN ZEME ZZ2 ON (AA2.ID_ZEME = ZZ2.ID_ZEME)
GROUP BY TT2.ID_TYMU, VZ2.DATUM)
VYPOCET_ZAVODY GROUP BY TYM) VYPOCET_TYMY
ON (HLAVNI_VYPOCET.TYM = VYPOCET_TYMY.TYM)

GROUP BY HLAVNI_VYPOCET.TYM, HLAVNI_VYPOCET.ZEME,
VYPOCET_TYMY.PODIOVE_ZAVODY, VYPOCET_TYMY.BODOVANE_ZAVODY,
VYPOCET_TYMY.DOKONCENE_ZAVODY, VYPOCET_TYMY.NEDOKONCENE_ZAVODY,
VYPOCET_TYMY.POCET_ZAVODU ORDER BY TYM;
```

Příloha J – *Univerzální SQL dotaz č. 10*

```
SELECT ROK, TYM, MOTOR, SASI,
(SELECT JEZDEC FROM
(SELECT SL.ROK, SL.ID_JEZDCE, SL.ID_TYMU, TT.NAZEV,
CONCAT(JJ.JMENO, CONCAT(' ', JJ.PRIJMENI)) AS JEZDEC,
(SELECT COUNT(*)+1 FROM STARTOVNI_LISTINY SL2
WHERE SL2.ID_TYMU = SL.ID_TYMU AND SL2.ROK = SL.ROK
AND SL2.ID_JEZDCE > SL.ID_JEZDCE) AS PILOT_CISLO
FROM STARTOVNI_LISTINY SL
INNER JOIN TYMY TT ON (SL.ID_TYMU = TT.ID_TYMU AND SL.ROK = TT.ROK)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)) VYPOCET
WHERE NAZEV = VYPOCET_HLAVNI.TYM AND ROK = VYPOCET_HLAVNI.ROK AND
PILOT_CISLO = 1) AS PILOT_C_JEDNA,
(SELECT JEZDEC FROM
(SELECT SL.ROK, SL.ID_JEZDCE, SL.ID_TYMU, TT.NAZEV,
CONCAT(JJ.JMENO, CONCAT(' ', JJ.PRIJMENI)) AS JEZDEC,
(SELECT COUNT(*)+1 FROM STARTOVNI_LISTINY SL2
WHERE SL2.ID_TYMU = SL.ID_TYMU AND SL2.ROK = SL.ROK
AND SL2.ID_JEZDCE > SL.ID_JEZDCE) AS PILOT_CISLO
FROM STARTOVNI_LISTINY SL
INNER JOIN TYMY TT ON (SL.ID_TYMU = TT.ID_TYMU AND SL.ROK = TT.ROK)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)) VYPOCET
WHERE NAZEV = VYPOCET_HLAVNI.TYM
AND ROK = VYPOCET_HLAVNI.ROK AND PILOT_CISLO = 2) AS PILOT_C_DVA,
BODY, PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO, CTVRTE_MISTO, PATE_MISTO FROM


(SELECT ROK, TYM, MOTOR, SASI, BODY, PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO,
CTVRTE_MISTO, PATE_MISTO, MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST,
MAXIMUM_DRUHYCH_MIST, MAXIMUM_TRETICH_MIST, MAXIMUM_CTVRTYCH_MIST,
(SELECT MAX(POCET_PATYCH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID_TYMU, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS
POCET_DRUHYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '3' THEN PP3.PORADI END) AS
POCET_TRETICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '4' THEN PP3.PORADI END) AS
POCET_CTVRTYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '5' THEN PP3.PORADI END) AS POCET_PATYCH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID_TYMU, SL3.ROK) VYPOCET
ON (VYPOCET.ID_TYMU = SL2.ID_TYMU AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_CTVRTYCH_MIST.ROK
AND VYPOCET.BODY = VYPOCET_CTVRTYCH_MIST.MAXIMUM_BODU
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_CTVRTYCH_MIST.MAXIMUM_PRVNICH_MIST
AND VYPOCET.POCET_DRUHYCH_MIST = VYPOCET_CTVRTYCH_MIST.MAXIMUM_DRUHYCH_MIST
AND VYPOCET.POCET_TRETICH_MIST = VYPOCET_CTVRTYCH_MIST.MAXIMUM_TRETICH_MIST
AND VYPOCET.POCET_CTVRTYCH_MIST =
VYPOCET_CTVRTYCH_MIST.MAXIMUM_CTVRTYCH_MIST) AS MAXIMUM_PATYCH_MIST FROM

(SELECT ROK, TYM, MOTOR, SASI, BODY, PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO,
CTVRTE_MISTO, PATE_MISTO, MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST,
MAXIMUM_DRUHYCH_MIST, MAXIMUM_TRETICH_MIST,
(SELECT MAX(POCET_CTVRTYCH_MIST) FROM STARTOVNI_LISTINY SL2
```

```
INNER JOIN (SELECT SL3.ID_TYMU, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS
POCET_DRUHYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '3' THEN PP3.PORADI END) AS
POCET_TRETICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '4' THEN PP3.PORADI END) AS
POCET_CTVRTYCH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID_TYMU, SL3.ROK) VYPOCET
ON (VYPOCET.ID_TYMU = SL2.ID_TYMU AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_TRETICH_MIST.ROK
AND VYPOCET.BODY = VYPOCET_TRETICH_MIST.MAXIMUM_BODU
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_TRETICH_MIST.MAXIMUM_PRVNICH_MIST
AND VYPOCET.POCET_DRUHYCH_MIST = VYPOCET_TRETICH_MIST.MAXIMUM_DRUHYCH_MIST
AND VYPOCET.POCET_TRETICH_MIST = VYPOCET_TRETICH_MIST.MAXIMUM_TRETICH_MIST)
AS MAXIMUM_CTVRTYCH_MIST FROM

(SELECT ROK, TYM, MOTOR, SASI, BODY, PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO,
CTVRTE_MISTO, PATE_MISTO, MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST,
MAXIMUM_DRUHYCH_MIST,
(SELECT MAX(POCET_TRETICH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID_TYMU, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS
POCET_DRUHYCH_MIST,
COUNT(CASE WHEN PP3.PORADI = '3' THEN PP3.PORADI END) AS POCET_TRETICH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3 ON (VV3.ID = VZ3.ID AND VV3.DATUM =
VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID_TYMU, SL3.ROK) VYPOCET
ON (VYPOCET.ID_TYMU = SL2.ID_TYMU AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_DRUHYCH_MIST.ROK
AND VYPOCET.BODY = VYPOCET_DRUHYCH_MIST.MAXIMUM_BODU
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_DRUHYCH_MIST.MAXIMUM_PRVNICH_MIST
AND VYPOCET.POCET_DRUHYCH_MIST = VYPOCET_DRUHYCH_MIST.MAXIMUM_DRUHYCH_MIST)
AS MAXIMUM_TRETICH_MIST FROM

(SELECT ROK, TYM, MOTOR, SASI, BODY, PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO,
CTVRTE_MISTO, PATE_MISTO, MAXIMUM_BODU, MAXIMUM_PRVNICH_MIST,
(SELECT MAX(POCET_DRUHYCH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID_TYMU, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS
POCET_PRVNICH_MIST,
COUNT(CASE WHEN PP3.PORADI = '2' THEN PP3.PORADI END) AS POCET_DRUHYCH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID_TYMU, SL3.ROK) VYPOCET
ON (VYPOCET.ID_TYMU = SL2.ID_TYMU AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_PRVNICH_MIST.ROK
```

```
AND VYPOCET.BODY = VYPOCET_PRVNICH_MIST.MAXIMUM_BODU
AND VYPOCET.POCET_PRVNICH_MIST = VYPOCET_PRVNICH_MIST.MAXIMUM_PRVNICH_MIST)
AS MAXIMUM_DRUHYCH_MIST FROM

(SELECT ROK, TYM, MOTOR, SASI, BODY, PRVNI_MISTO, DRUHE_MISTO, TRETI_MISTO,
CTVRTE_MISTO, PATE_MISTO, MAXIMUM_BODU,
(SELECT MAX(POCET_PRVNICH_MIST) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID_TYMU, SL3.ROK, SUM(PP3.BODY) AS BODY,
COUNT(CASE WHEN PP3.PORADI = '1' THEN PP3.PORADI END) AS POCET_PRVNICH_MIST
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID_TYMU, SL3.ROK) VYPOCET
ON (VYPOCET.ID_TYMU = SL2.ID_TYMU AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = VYPOCET_MAXIMUM_BODU.ROK AND VYPOCET.BODY =
VYPOCET_MAXIMUM_BODU.MAXIMUM_BODU) AS MAXIMUM_PRVNICH_MIST FROM

(SELECT SL.ROK, TT.NAZEV AS TYM, TT.MOTOR, TT.SASI, SUM(PP.BODY) AS BODY,
COUNT(CASE VZ.PORADI WHEN '1' THEN VZ.PORADI END) AS PRVNI_MISTO,
COUNT(CASE VZ.PORADI WHEN '2' THEN VZ.PORADI END) AS DRUHE_MISTO,
COUNT(CASE VZ.PORADI WHEN '3' THEN VZ.PORADI END) AS TRETI_MISTO,
COUNT(CASE VZ.PORADI WHEN '4' THEN VZ.PORADI END) AS CTVRTE_MISTO,
COUNT(CASE VZ.PORADI WHEN '5' THEN VZ.PORADI END) AS PATE_MISTO,
(SELECT MAX(VYPOCET.BODY) FROM STARTOVNI_LISTINY SL2
INNER JOIN (SELECT SL3.ID_TYMU, SL3.ROK, SUM(PP3.BODY) AS BODY
FROM VYSLEDKY VV3
INNER JOIN VYSLEDKY_ZAVODU VZ3
ON (VV3.ID = VZ3.ID AND VV3.DATUM = VZ3.DATUM)
INNER JOIN STARTOVNI_LISTINY SL3 ON (VV3.ID = SL3.ID)
INNER JOIN PORADI PP3 ON (VZ3.PORADI = PP3.PORADI)
GROUP BY SL3.ID_TYMU, SL3.ROK) VYPOCET
ON (VYPOCET.ID_TYMU = SL2.ID_TYMU AND VYPOCET.ROK = SL2.ROK)
WHERE SL2.ROK = SL.ROK) AS MAXIMUM_BODU
FROM VYSLEDKY VV
INNER JOIN VYSLEDKY_ZAVODU VZ ON (VV.ID = VZ.ID AND VV.DATUM = VZ.DATUM)
INNER JOIN STARTOVNI_LISTINY SL ON (VV.ID = SL.ID)
INNER JOIN JEZDCI JJ ON (SL.ID_JEZDCE = JJ.ID_JEZDCE)
INNER JOIN TYMY TT ON (SL.ID_TYMU = TT.ID_TYMU AND SL.ROK = TT.ROK)
INNER JOIN PORADI PP ON (VZ.PORADI = PP.PORADI)
GROUP BY SL.ROK, TT.NAZEV, TT.MOTOR, TT.SASI) VYPOCET_MAXIMUM_BODU)

VYPOCET_PRVNICH_MIST)

VYPOCET_DRUHYCH_MIST)

VYPOCET_TRETICH_MIST)

VYPOCET_CTVRTYCH_MIST)

VYPOCET_HLAVNI


WHERE VYPOCET_HLAVNI.BODY = VYPOCET_HLAVNI.MAXIMUM_BODU AND
VYPOCET_HLAVNI.PRVNI_MISTO = VYPOCET_HLAVNI.MAXIMUM_PRVNICH_MIST AND
VYPOCET_HLAVNI.DRUHE_MISTO = VYPOCET_HLAVNI.MAXIMUM_DRUHYCH_MIST AND
VYPOCET_HLAVNI.TRETI_MISTO = VYPOCET_HLAVNI.MAXIMUM_TRETICH_MIST AND
VYPOCET_HLAVNI.CTVRTE_MISTO = VYPOCET_HLAVNI.MAXIMUM_CTVRTYCH_MIST AND
VYPOCET_HLAVNI.PATE_MISTO = VYPOCET_HLAVNI.MAXIMUM_PATYCH_MIST
ORDER BY ROK;
```

Příloha K – Seznam a popis adresářů na DVD

VYSLEDKY_UNIVERZALNICH_DOTAZU – složka obsahuje pro každý databázový systém výsledky univerzálních dotazů

VYSLEDKY_OPTIMALIZOVANYCH_DOTAZU – složka zahrnuje pro každý databázový systém výsledky optimalizovaných dotazů

EXEKUCNI_PLANY_UNIVERZALNICH_DOTAZU – složka obsahuje pro každý databázový systém exekuční plány jednotlivých univerzálních dotazů

EXEKUCNI_PLANY_OPTIMALIZOVANYCH_DOTAZU – složka obsahuje pro každý databázový systém exekuční plány jednotlivých optimalizovaných dotazů, pro každý optimalizovaný dotaz existuje jeden sql skript postupného vývoje dotazu, další skripty slouží pro změny struktury databázového modelu (partitioning)

DBFIT-COMPLETE-3.2.0 – složka obsahuje celý program DbFit, všechny univerzální a optimalizované dotazy společně s výsledky a přidané soubory .jar s JDBC pro každý jednotlivý databázový systém

SQL_CREATE_DDL – obsahuje skripty potřebné k vytvoření databázových modelů na jednotlivých databázových systémem (potřeba změnit ve všech souborech absolutní cesty)

DATA – složka obsahuje následující soubory:

- SQL skripty obsahující data pro každý databázový systém (Oracle == PostgreSQL)
- soubory CSV pro generování SQL skriptů
- ZAKLADNI_DATA.xlsx – data získaná ze stránek zabývajícími se Formulí 1
- soubory TXT (vygenerované ID) pro generování SQL skriptů
- Generate_data.jar – soubor, který vygeneruje SQL skripty ()

DBFIT_VYSLEDKY_UNI_A_OPT_DOTAZU – složka obsahuje soubory typu CSV a SQL, které obsahují výsledky univerzálních a optimalizovaných dotazů (tyto výsledky jsou použity pro program DbFit)

SQL_DOTAZY – složka obsahuje kompletní souhrn univerzálních a optimalizovaných dotazů i souhrn univerzálních a optimalizovaných dotazů přímo pro program DbFit

DATABAZOVY_MODEL – složka obsahuje vytvořený databázový model v programu Oracle SQL Developer Data Modeler