Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Srovnání databází Oracle a PostgreSQL

Marek Flaška

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačních technologií

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Marek Flaška

Systémové inženýrství a informatika

Název práce
Srovnání databází Oracle a PostrgreSQL

Anglický název
Comparison of databases Oracle and PostgreSQL

Cíle práce

Práce je založena na studiu odborné literatury, online zdrojů a odborných technických diskuzí. Hlavním cílem práce je srovnání databází Oracle a PostgreSQL z různých hledisek a analyzovat jejich přednosti a slabá místa. Mezi tato hlediska patří například architektura, náročnost na hardware, podpora od výrobce a mnoho dalších. Dílčím cílem práce je pak obecné vyhodnocení toho, ve kterých případech se vyplatí používat jednu ze zkoumaných databází.

Metodika

Teoretická část diplomové práce je založena na studiu a analýze příslušných literárních nebo jiných odborných zdrojů. Jejím výstupem je obecný přehled řešené problematiky, který je následně konfrontován s praktickou částí. Ta je zaměřena na analýzu vybraných prvků jedné databáze a jejich porovnání s příslušnou alternativou v druhé databázi. Závěry diplomové práce jsou pak formulovány na základě spojení výsledků praktické a teoretické části.

Rozsah textové části

60 – 80 stran

Klíčová slova

architektura, funkcionalita, ladění, možnosti, Oracle, PostgreSQL, Relační databáze, srovnání

Doporučené zdroje informací

1. LONEY, Kevin. Oracle Database Kompletní průvodce. 1. vydání. Brno: Computer Press, a. s., 2010. 1368 s. ISBN 978-80-251-2489-5
2. SOLAŘ, Tomáš. Oracle Database 11g Hotová řešení.1. vydání. Brno: Computer Press, a. s., 2010. 1368 s. ISBN 978-80-251-2886-2
3. SMITH, Gregory. PostgreSQL 9.0 High Performance. 1. vydání. Birmingham: Packt Publishing, 2010. 468 s. ISBN 978-1-84951-030-1.
4. MATTHEW, Neil – STONES, Richard. Beginning Databases with PostgreSQL. 2 vydání. New York: Apress, 2005. 664 s. ISBN 1-59059-478-9.

Vedoucí práce

Ing. Alexandr Vasilenko

Elektronicky schváleno dne 31. 10. 2014

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 11. 11. 2014

Ing. Martin Pelikán, Ph.D.

Děkan PEF ČZU

V Praze dne 15. 01. 2015

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Srovnání databází Oracle a PostgreSQL" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.3.2015 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Alexandru Vasilenkovi za odborné vedení, cenné rady a informace, které mi poskytl během tvorby této diplomové práce.

Srovnání databází Oracle a PostgreSQL

Comparison of databases Oracle and PostgreSQL

Souhrn

Diplomová práce se věnuje problematice srovnání dvou vybraných relačních databází – Oracle a PostgreSQL. V teoretické části jsou definovány pojmy relační databáze, jazyk SQL, databázové objekty a mnoho dalších souvisejících. Přiblížení těchto pojmů je potřebné pro vlastní srovnání.

V teoretické části jsou také popsány historie obou databází, jejich architektury a společné znaky. Toto slouží jako znalostní základna, ze které se vychází v analytickém srovnání vlastností a funkcí, jež obě databáze nabízejí. Srovnání je založeno na vyhodnocení předem definovaných kritérií.

Finální hodnocení je pak k dispozici v psané i číselné formě.

Summary

This diploma thesis focuses on comparison of two chosen relational databases – Oracle and PostgreSQL. The theoretic part defines terms relational database, SQL language, database objects and many other related terms. Close up of these terms is required before the initial comparison.

The theoretical part also deals with the histories of both databases, their architectures and common traits. This serves as knowledge base, which is used for the analytical comparison of features and attributes that both databases offer. The comparison is based on evaluation of predefined criteria.

The final conclusion is available both in text and numeric form.

Klíčová slova: architektura, funkcionalita, ladění, možnosti, Oracle, PostgreSQL, Relační databáze, srovnání

Keywords: architecture, functionality, tuning, features, Oracle, PostgreSQL, Relational databases, comparison

Obsah

1. Úvod

Žijeme ve světě, který je protkán informačními technologiemi. Počítače a různé výpočetní systémy dnes používá téměř každý – od běžných občanů přes malé a velké firmy až ke státní správě. Všechny výše vyjmenované subjekty mají potřebu někde uchovávat svá data a ideálně je zabezpečit tak, aby se k nim nedostal nikdo nepovolaný.

Data lze ukládat pomocí *textových procesorů* do textových dokumentu, nebo pomocí *tabulkových procesorů* do tabulek, které se používají k finančním výpočtům a tvorbě analýz [1]. Tyto dva způsoby lze však použít pouze pro zpracovávání malého objemu dat. Zpracovávat velké objemy dat, systematicky je ukládat a komplexně je analyzovat umožňují až databáze.

Databáze (neboli datová základna) je kolekce vzájemně souvisejících datových položek, které jsou spravovány jako jediná jednotka [2]. Jiná definice říká, že databáze je určitá uspořádaná množina informací (dat) uložená na paměťovém médiu [3]. Systém řízení báze dat (SŘBD) je software dodávaný výrobcem databáze. Tento pojem vznikl překladem anglického *DataBase Management System*. Jde o systém sloužící k manipulaci s uloženými daty a přístupu k nim. Z hlediska pojmu „databáze“ už dnes není toto rozdělení tak striktní a tak se pojmem databáze běžně označují uložená data společně s řídicím systémem.

Existuje mnoho různých výrobců databázových systémů, jejichž produkty se liší mnoha parametry. Mezi nejznámější databázové systémy patří Oracle a MySQL od společnosti Oracle Corporation, Firebird od společnosti Borland [4], SQL Server od firmy Microsoft, DB2 od společnosti IBM nebo PostgreSQL vyvíjený komunitou jako *open source*. Existuje ještě celá řada dalších databází, které však nejsou tolik známé a používané.

Výše uvedené systémy řízení báze dat (dále jen SŘBD) mají mnoho společného – jsou postaveny na relačním modelu, jde tedy o relační databázové systémy (viz Kapitola 3.1). Data ukládají do tabulek, obsahují nástroje k zajištění integrity dat atd. V jiných parametrech se ale mnohé systémy řízení báze dat více či méně liší.

Právě na srovnání dvou vybraných systémů řízení báze dat (Oracle a PostgreSQL) se zaměřuje tato práce.

2. Cíl práce a metodika

2.1 Cíl práce

Cílem této práce je srovnat vybrané databázové systémy (Oracle a PostgreSQL) z hlediska zvolených parametrů. Jelikož vybrané SŘBD mají mnoho společného, je kladen důraz především na parametry lišící se v obou systémech. Cílem práce je tyto parametry najít, analyzovat, porovnat je s obdobnými parametry ve druhém SŘBD a vytvořit na základě těchto parametrů srovnání obou těchto systémů za použití matematických metod.

Dílčím cílem je vytvořit literární rešerši obsahující úvod do problematiky společně se srovnáním parametrů, které mají oba SŘBD podobné, nebo nemají zásadní vliv na výběr dat, nicméně jsou autorem považovány za důležité.

2.2 Metodika

Teoretická část práce je založena na studiu odborné literatury. Je v ní přiblížena problematika (relační databázové systémy, SŘBD Oracle a PostgreSQL) a zároveň jsou zde oba systémy srovnány v obecné rovině.

Z této obecné úrovně se následně vychází v praktické části, ve které je provedena srovnávací analýza vybraných konkrétních parametrů obou systémů. Jejím výstupem je vyhodnocení těchto parametrů na základě Metody kvantitativního párového srovnání kritérií (Saatyho metoda). Na základě matematického srovnání je vytvořeno i slovní zhodnocení, v jakých situacích se hodí spíše SŘBD Oracle a kdy naopak SŘBD PostgreSQL.

3. Přehled řešené problematiky

3.1 Relační databáze

3.1.1 Obecné principy

Koncept relačních databází byl poprvé popsán Dr. Edgarem F. Coddem v roce 1970. Hlavní jednotkou pro ukládání dat v relační databázi je tabulka, dvourozměrná struktura, která se skládá z řádků a sloupců, často také nazývaná *entita* [2]. Řádky tabulky se dají považovat za záznamy, zatímco sloupce představují atributy daného záznamu. Zde je vidět podobnost s výše zmiňovaným *tabulkovým procesorem,* nicméně databázové systémy nabízejí mnohem více funkcí a větší flexibilitu při ukládání dat a následné manipulaci s nimi.

Relace představují souvislosti mezi jednotlivými tabulkami databáze. Tabulky sice mohou existovat samostatně, nicméně hlavní funkcí databáze je ukládat související data [2]. Pokud například do jedné tabulky ukládám informace o knihách, v jiné mohu zase evidovat kategorie, do kterých jsou jednotlivé knihy zařazeny. Pomocí relace lze provázat mnoho tabulek a přitom v dotazu flexibilně zahrnout pouze ta data, která jsou relevantní. Právě tato snadná použitelnost a pochopitelnost byla jedním z důvodů, proč se relační model v 80. letech 20. století stal velmi populárním [5].

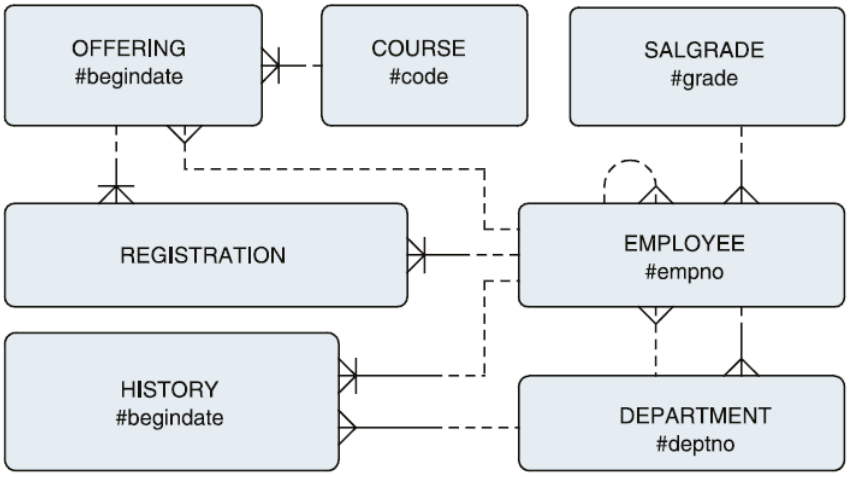
V samotné databázi jsou pak tyto relace reprezentovány systémem klíčů (*keys*), které jsou implementovány skrze *referenční integritní omezení (constraints)*. Existují *primární klíče* a *cizí klíče*.

V každé tabulce může existovat vždy pouze jeden primární klíč. Jde o sloupec (nebo skupinu sloupců), který obsahuje unikátní data (neopakující se) a neobsahuje hodnotu *NULL* (sloupec není prázdný) [6]. Jde o jakýsi jednoznačný identifikátor každého řádku. Struktura databáze (systém tabulek a vazeb mezi nimi) by měla být navržena tak, aby se primární klíč založeného záznamu nemusel nikdy měnit. Typickým příkladem primárního klíče je v praxi identifikační číslo (ID) nebo katalogové číslo výrobku.

Na rozdíl od primárního klíče může cizích klíčů v tabulce existovat více. Nejčastěji ukazují do jiných tabulek a je pomocí nich zajištěna integrita dat. Slouží k vytváření relací (vazeb) mezi tabulkami. Typy vazeb mohou být následující: žádná vazba, 1:1, 1:N anebo N:M [6]. Systém řízení báze dat pak kontroluje, zda pro všechny hodnoty cizího klíče existuje hodnota primárního klíče v nadřízené tabulce.

3.1.2 Diagramy ERD

Celkový návrh struktury databáze (z hlediska tabulek a vazeb mezi nimi) je nejčastěji zobrazován pomocí diagramu ERD – *Entity Relationship Diagram*. Každý obdélník v diagramu představuje tabulku, čáry mezi obdélníky pak představují vazby mezi nimi. Viz následující obrázek:



Obrázek 1: Příklad diagramu ERD [7]

V diagramu je celkem sedm entit (tabulek), jejichž názvy jsou zobrazeny velkými písmeny. Většina atributů (sloupců) byla kvůli zachování přehlednosti vynechána, zobrazeny jsou pouze klíčové atributy. Mezi entitami pak existují různé vztahy, například rozvětvená čára symbolizuje vztah 1:N. Na obrázku výše existuje následující interpretace vztahu mezi tabulkami *REGISTRATION* a *OFFERING*:

- Každá registrace (záznam v tabulce *REGISTRATION*) existuje vždy pouze na jednu nabídku kurzu (záznam v tabulce *OFFERING*).
- Avšak nabídka kurzu (záznam v tabulce *OFFERING*) může obsahovat žádnou, jednu, nebo mnoho registrací.

Zajímavostí je existence dvou vztahů mezi tabulkami oddělení (*DEPARTMENT*) a zaměstnanec (*EMPLOYEE*). Každý zaměstnanec pracuje právě pod jedním oddělením a zároveň každý zaměstnanec může být manažerem žádného, jednoho nebo více oddělení. Existuje zde tzv. *rekurzivní vazba*, kdy je entita ve vztahu sama se sebou [7].

3.1.3 Dotazovací jazyk SQL

SQL (*Structured Query Language*) je dotazovací jazyk, který se používá pro komunikaci s relačními databázemi. Řadí se mezi neprocedurální neboli deklarativní jazyky [2]. To znamená, že systému řízení báze dat sděluje uživatel pouze to, jaké výsledky požaduje, a nikoliv způsob, jakým je má systém získat.

Procedurální jazyky obsahují posloupnost příkazů, které se mají spustit v určeném pořadí, případně příkazy, jak toto pořadí změnit nebo je provádět cyklicky. Toto tedy jazyk

SQL nepodporuje, nicméně mnoho dodavatelů SŘBD nabízí procedurální rozšíření jazyka SQL do svých systémů – například jazyk PL/SQL (*Procedural Language/SQL*) společnosti Oracle nebo Transact-SQL v databázích firmy Microsoft [2]. Tato rozšíření jsou však považována za nové jazyky, byť umožňují provádět klasické příkazy jazyka SQL.

SQL příkazy lze dle funkčnosti rozdělit do několika kategorií. Tyto kategorie nelze přímo označit za samostatné jazyky, jelikož mají stejnou základní syntaxi a pravidla jako samotný jazyk SQL:

- Jazyk DDL (Data Definition Language).
- Jazyk DML (Data Manipulation Language).
- Jazyk DCL (Data Control Language).

Jazyk DDL slouží k vytváření databázových objektů – tabulek, pohledů, indexů atd. Zároveň umožňuje upravovat jejich strukturu. Typicky sem patří příkazy obsahující klíčová slova *CREATE, ALTER* a *DROP*.

Do jazyka DML se řadí příkazy, které umožňují uživatelům přidávat data do databáze, měnit je, nebo mazat stávající. Příkazy tohoto jazyka obsahují klíčová slova *UPDATE, INSERT* a *DELETE*.

Jazyk DCL pomáhá správcům databáze řídit přístup k datům v databázi a obecně řídit běh databáze. Obsahuje například příkazy pro spuštění či vypnutí databáze nebo pro přidělování příslušných oprávnění jednotlivým uživatelům (klíčová slova *GRANT, ALTER* a *REVOKE*) [8].

3.1.4 Další objekty v databázi

Kromě tabulek a klíčů se v databázi běžně vyskytuje mnoho dalších důležitých objektů, které mají velmi důležité funkce pro správný a především rychlý chod databáze.

Pohled (*view*) je databázový objekt, který poskytuje uživateli data ve stejné formě, jako tabulka. V podstatě je to pouze logická definice, jak tato data získat. Uživatel si uloží často používaný dotaz jako pohled a tím odpadne nutnost zadávat dotaz pokaždé znovu včetně všech možných kritérií obsažených v klauzuli *WHERE*. Pohled sám o sobě žádná data neobsahuje a ani data nikam neukládá (výjimkou je *materializovaný pohled*), pouze existuje jako náhled do tabulky/tabulek. SŘBD zároveň zajišťuje, že pohled je vždy aktuální – změny v původních tabulkách se projeví i v pohledech na nich postavených a obráceně [6].

Indexy jsou databázové objekty sloužící ke zrychlení dotazovacích a vyhledávacích procesů v databázi [9]. Tabulka sama o sobě obsahuje data v neseřazené podobě – jednou z klíčových vlastností relačních databází je to, že pozice řádku je nevýznamná. V případě hledání konkrétního záznamu je tedy potřeba projít všechny záznamy v dané tabulce (provést tzv. *Full table scan*). V případě malého počtu záznamů to nepředstavuje problém, nicméně v tabulkách obsahujících tisíce záznamů začne být tato operace velice náročná na výpočetní výkon. Z tohoto důvodu se začaly používat indexy. Tyto objekty obsahují informace o rozmístění indexovaných hodnot (záznamů) v tabulce. Tabulka pak není prohledávána přes všechny řádky, nýbrž se přímo přistupuje přes informace v indexu k řádkům relevantním. Lze říci, že index funguje jako rejstřík v knize.

Na první pohled by se mohlo zdát, že čím více indexů na tabulce existuje, tím jsou dotazy rychlejší. V praxi to takto bohužel nefunguje. Když je vytvořen nový index na nějaké tabulce, SŘBD vyhradí místo v paměti a uloží do něj informace o indexovaných sloupcích dané tabulky. DML operace upravující stávající data pak kromě samotných dat musejí aktualizovat i indexy, což vede ke zpomalení těchto operací a zároveň jsou zvýšeny nároky na paměťový prostor. Velký počet indexů může klidně způsobit, že paměť zabraná pro jejich chod je stejně velká jako paměť zabraná vlastními daty, což je pochopitelně nežádoucí situace [6].

Triggers (česky *spouštěče* nebo *aktivační události*) jsou objekty, jejichž cílem je reagovat na nějakou situaci, která v databázi nastane, a provést odpovídající akci. Spouštěče mohou existovat na úrovni celé databáze, nebo jen nad tabulkou. Typickou spouštěcí událostí jsou DML operace (*INSERT, UPDATE, DELETE*) nebo DDL operace (*CREATE, ALTER, DROP*). Obvyklým využitím jsou auditorské procesy – například při mazání řádků z tabulky (aktivační událost typu *ON DELETE* – viz kapitola 4.3.2) se do jiné tabulky ukládají informace o tom, kdo a kdy tuto operaci provedl.

Mnoho databází dále podporuje ukládání uživatelských funkcí napsaných v procedurálních nadstavbách (viz kapitola 3.4.4). Tyto funkce jsou nazývány pojmem UDF – *User Defined Function*. Vestavěné funkce jednotlivých systémů jsou pochopitelně rychlejší, než uživatelsky napsané, nicméně ne vždy systém obsahuje v základní verzi všechny funkce, které uživatel vyžaduje.

3.1.5 Databázové transakce

Databázová transakce je oddělená sada akcí, které je nutné zpracovat buď jako celek, nebo vůbec. Transakce se někdy označují jako pracovní jednotky, což zdůrazňuje jejich nedělitelnost [2]. Existují čtyři základní vlastnosti transakcí:

- Atomicita – transakce musí zůstat vcelku (je nedělitelná). To znamená, že buď proběhne celá kompletně, nebo neproběhne vůbec. Jestliže proběhne, systém změny potvrdí (potvrzení se označuje slovem *COMMIT*). Pokud transakce selže, je nutné změny vrátit do původního stavu (operace *ROLLBACK*).
- Konzistence – transakce musí převést databázi z jednoho konzistentního stavu do jiného. Pokud jsou například v bankovním systému převáděny peníze z jednoho účtu na druhý, vždy musí být z jednoho účtu smazány a na druhém připsány, případně musí zůstat na prvním účtu, nastane-li chyba. Jiný stav není přípustný.
- Izolace – každá transakce musí fungovat nezávisle na ostatních.
- Trvanlivost – změny, které transakce provede, musejí být zachovány i v případě vypnutí nebo chyby databáze.

V dnešní době podporují transakce téměř všechny SŘBD, byť se někdy vlastní způsob implementace transakcí u jednotlivých výrobců liší [2].

3.2 Systém řízení báze dat Oracle

3.2.1 Historie společnosti Oracle Corporation

SŘBD Oracle je produktem firmy Oracle Corporation. Jde o multiplatformní systém nabízející mnoho funkcí pro zpracování dat. Historie společnosti se datuje do roku 1977, kdy softwaroví inženýři Larry Ellison, Bob Miner a Ed Oates založili konzultantskou společnost Software Development Laboratories. Mimo jiné se podíleli na programování databáze pro CIA, která nesla krycí název Oracle [10].

Jedním z impulzů pro založení vlastní společnosti byl článek britského profesora Edgara F. Codda, který popisoval relační model dat. Z tohoto konceptu vycházel princip relačních databází. V tehdejší době byla tato technologie ostatními společnostmi na trhu přehlížena, nicméně Larry Ellison v ní viděl obrovský potenciál a příležitost.

V roce 1979 byl vydán jeho již přejmenovanou firmou (Relational Software) první komerční produkt označený Oracle V2. Zájem o něj byl však minimální. SŘBD sice podporoval základní SQL příkazy, ale nepodporoval transakce (viz kapitola 3.1.5). V roce 1982 přišla vylepšená verze Oracle V3, nicméně za prvních šest měsíců od uvedení produktu do prodeje se ho neprodal jediný kus [10].

Potencionální zákazníky přesvědčily až prezentace, které u nich tvůrci databázového systému prováděli. V roce 1983 se firma přejmenovala na Oracle. Během tohoto roku již vydělala 24 milionů dolarů a rostla i v následujících letech. SŘBD byla přepsán v jazyce C, portován na operační systém Unix a převeden na model klient/server.

Osmdesátá léta by se dala považovat za zlatý věk relačních databází. Společnost Oracle tehdy produkovala obrovské množství softwaru. Růst společnosti ustal až v roce 1990, kdy konečně začaly na trh přicházet konkurenční výrobky. Ellison se toho však nezalekl a díky jeho manažerským schopnostem byl Oracle za dva roky opět v černých číslech.

Na začátku 21. století společnost upevňuje svou pozici četnými akvizicemi. 20. dubna 2009 koupila společnost Oracle firmu Sun Microsystems a udělala z ní svou dceřinou společnost [11]. Tím získala přístup k SŘBD MySQL, který by se dal považovat za jednoho z největších konkurentů databází Oracle. Z dalších akvizic lze zmínit například I-flex Solutions (900 milionů dolarů) nebo Hyperion solutions (3,3 miliardy dolarů). Sám Ellison je velice sebevědomý a vidí ve společnosti Oracle ještě větší potenciál, což potvrzuje svými slovy: *„IBM je minulost, Microsoft současnost a Oracle budoucnost“* [10].

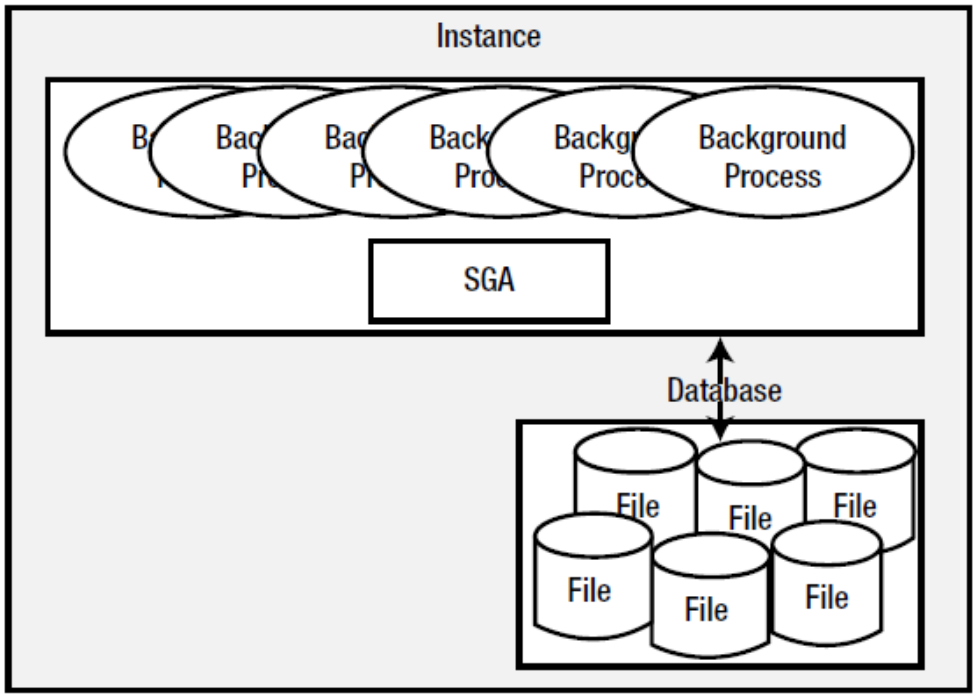
3.2.2 Struktura instance v SŘBD Oracle

V souvislosti s pojmem architektura databáze je třeba rozlišovat dva základní pojmy – databáze a instance. Byť jsou tyto dva pojmy velmi často zaměňovány a považovány za totožné, mají různý význam. Z hlediska architektury je databáze kolekce souborů fyzicky umístěných na disku [12]. Patří sem všechny databázové objekty uvedené v kapitole 3.1.4. Některé soubory mohou být umístěny i mimo databázi, například externí tabulky.

Oproti tomu instance je čistě logická. Instance se při startu databáze nahraje do operační paměti počítače. Jde o jakousi kolekci procesů, která komunikuje s paměťovou oblastí a s fyzickými soubory databáze umístěnými na disku. Instance běží v operační paměti počítače nebo serveru, oproti tomu databáze se fyzicky nachází pouze na pevném disku či jiném paměťovém médiu. Pokud uživatel pracuje s databází, ve skutečnosti komunikuje

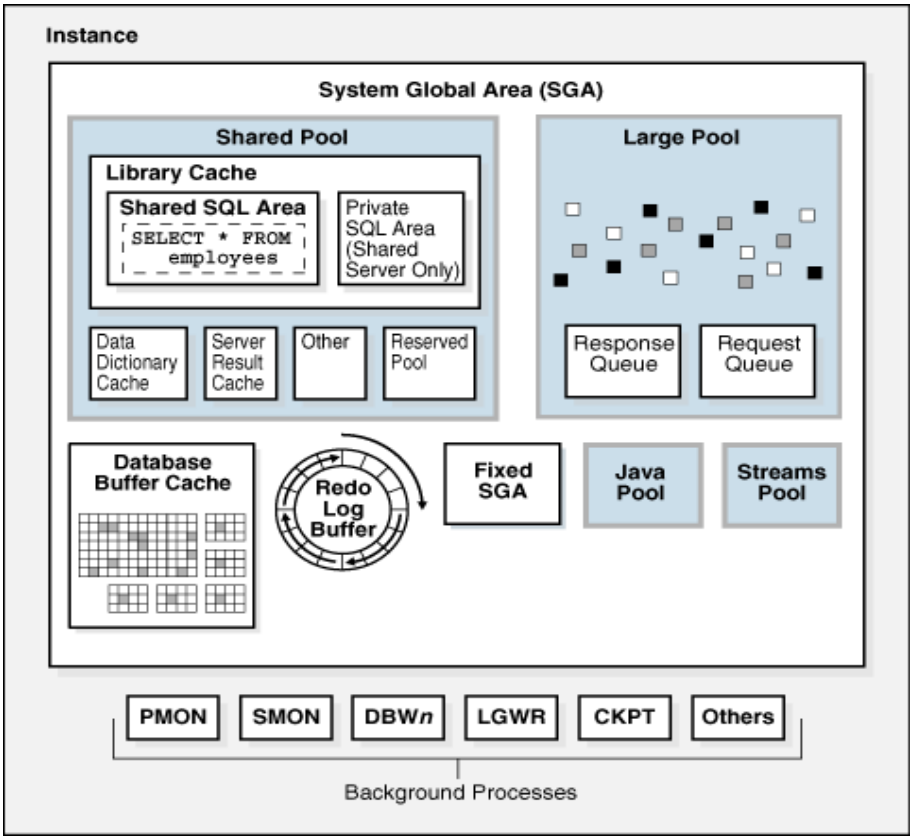
pouze s instancí, která následně obstarává požadované výsledky z fyzických úložišť databáze.

Jelikož instance běží v operační paměti, je pouze dočasná. Oproti tomu fyzická databáze uložená na disku přetrvá i po vypnutí instance. Důležité je, že jedna instance může obsluhovat právě jednu databázi. Při využití technologie Real Application Cluster (RAC) společnosti Oracle lze jednu databázi obsluhovat více instancemi [8].



Obrázek 2: Vztah instance a databáze v SŘBD Oracle [13]

Jak již bylo řečeno, instance se skládá z vyhrazené paměti a procesů běžících na pozadí. Sdílená paměť instance se nazývá System Global Area (SGA). Paměť SGA je segmentovaná, každá její část má svoji specifickou funkci [6]. Všechny procesy dané instance tuto paměť sdílejí. Velikost této paměti se dá nastavit pomocí inicializačního parametru, od verze Oracle 9i pak lze tento parametr měnit bez nutnosti restartovat instanci. Automaticky se tak mění velikost následujících segmentů paměti: *Shared pool*, *Large pool*, *Java pool*, velikost bufferu (*Buffer chache*) a *Streams pool* [8]. Dále do paměti SGA patří ještě *Redo log buffer*.



Obrázek 3: Struktura instance v SŘBD Oracle [14]

*Database Buffer Cach*e se používá při veškeré komunikaci s datovými soubory. Ukládají se v něm bloky dat získaných z databáze. Pokud je blok dat již uložen v bufferu (například z předchozího dotazu), je získán z paměti a není tak potřeba znovu přistupovat na disk k jeho získání. SŘBD Oracle obsluhuje tuto část paměti pomocí *last recently used* algoritmu (*LRU*) [8]. Pokud je potřeba přečíst blok, který není v paměti, je potřeba ho nejprve do paměti nahrát. Pokud je nějaký blok uživatelem změněn, nejprve se změní v paměti a až po určitém čase jsou provedeny změny na disku. Tím pádem uživatel nemusí čekat na to, až jsou jeho data fyzicky zapsána na disk. Smyslem práce s daty v paměti je omezit práci s pevným diskem. Pevné disky jsou nejpomalejší částí pracovní stanice a jakákoliv práce s nimi zpomaluje celý systém. Filozofie této struktury je tedy taková, že data se fyzicky zapisují/čtou až v případě, kdy je to nezbytně nutné.

Dalším segmentem SGA je *Redo log buffer*. Tato paměť slouží ke komunikaci s *redo logy*, což jsou soubory, do kterých se ukládají veškeré informace o činnosti databáze [6]. Buffer ukládá informace předtím, než jsou zapsány do fyzických redo logů na disku.

Toto má za následek zrychlení chodu databáze, jelikož redo logy jsou nejčastěji přistupované soubory v databázi.

Shared pool uchovává všechny konstrukce SQL dotazů, informace o kurzorech a všechna metadata k objektům, které mohou být sdíleny různými uživateli. Stejně tak jsou sem nahrávány PL/SQL funkce před zpracováním. Jejich výsledky jsou zde uchovávány taktéž. *Shared pool* je rozdělen ještě do několika menších oblastí. Nejdůležitější je *Shared SQL area*, která slouží k uchovávání všech informací o dotazech. *Data dictionary cache* slouží k uchovávání informací o objektech, s kterými se pracuje v paměti. Odpadá tak nutnost opakovaně nahlížet do datového slovníku (interní přehled všech objektů v databázi).

Large pool se používá pro operace, které jsou náročné na I/O (vstupní a výstupní) operace. Sem patří především zálohování a obnova (recovery) [8]. *Java pool* se používá při vykonávání kódu v Javě a pro objekty s ním spojené, například data v JVM (*Java Virtual Machine*). *Streams pool* pak slouží pro technologii Oracle Streams [6].

Vedle paměti SGA pak existuje ještě Program Global Area. Ta v sobě uchovává informace vždy pro konkrétní připojení (*session*) a alokuje se samostatně pro každý serverový proces [8].

Background procesy jsou takové procesy, které běží nezávisle na tom, co uživatel dělá. Jsou vytvořeny ihned po startu databáze bez ohledu na to, v jakém režimu byla databáze spuštěna.

PMON (*Process monitor*) dohlíží na uživatelské a serverové procesy, které pracují s databází. Pokud je například nějaký z nich ukončen abnormálně, postará se PMON o uvolnění zámků a smazání zdrojů z paměti – provede *ROLLBACK* operaci.

SMON (*System monitor process*) zajišťuje bezpečný chod instance. Provádí obnovu instance, pokud je nastartována po nějaké chybě.

DBWn (*Database writer process*) zapisuje bloky dat z bufferu v SGA paměti na disk, případně čte z disku do SGA. Jedna instance může mít až 20 DBW procesů pro manipulaci s daty – proto označení DWB*n* [8].

LGWR (*Log writer process*) zapisuje informace z redo log bufferu v SGA do všech kopií redo logů na disku. Zápis probíhá každé tři vteřiny, nebo při příkazu *COMMIT* a nebo je-li *redo log buffer* z jedné třetiny plný [6].

CKPT (*Checkpoint process*) provádí operaci zvanou *Checkpoint*. Při ní dochází k synchronizaci všech souborů v databázi kvůli zachování konzistence dat.

Existuje ještě několik dalších background procesů, nicméně není důvod vypisovat všechny. Pro představu chodu databáze postačí výše uvedené.

3.2.3 Struktura fyzické databáze Oracle

Fyzická struktura databáze Oracle zahrnuje *tablespaces* (tabulkové prostory), *control files* (kontrolní soubory), *redo log files* (redo log soubory), *archived logs* (archivní redo log soubory), *block change tracking files*, *Flashback logs* a *recovery backup files* (zálohy datových souborů).

Tabulkové prostory jsou logické struktury, do kterých jsou ukládána veškerá data. Každý tabulkový prostor se skládá z jednoho nebo více datových souborů (*datafile*). Každý datový soubor může patřit pouze jednomu tabulkovému prostoru. Při vytváření tabulky se musí určit, do jakého tabulkového prostoru se má tabulka uložit. Databáze Oracle se může skládat z cca 64 000 datových souborů. Při použití tabulkového prostoru typu *bigfile* (druhým typem je *smallfile*) mají bloky (viz dále) v 64-bitovém operačním systému velikost 32 KB a teoreticky lze dosáhnout úložného prostoru až 8 exabytů (jeden exabyte je milion terabytů) [8]. V databázi musí existovat minimálně dva tabulkové prostory – *SYSTEM* a *SYSAUX*, ty jsou nezbytné proto, aby databáze vyhověla svým vlastním interním požadavkům [12].

Kontrolní soubory tvoří společně s datovými soubory a redo log soubory tři základní pilíře fyzické databáze. Kontrolní soubor obsahuje informace o umístění a názvu všech ostatní souborů (datových a redo logů). Dále obsahují název databáze, informace o tabulkových prostorech a informace o provedených zálohách. Umístění souborů se nastavuje parametrem *CONTROL_FILES*, který využívá instance pro přístup ke kontrolním souborům. Doporučuje se mít alespoň jednu kopii kontrolního souboru umístěnou na jiném fyzickém disku. Při ztrátě lze sice kontrolní soubor obnovit, ale jde o velice náročný proces.

Datové soubory obsahují veškerá data umístěná v databázi – tabulky, indexy atd. Datový soubor je složen z Oracle bloků (*Oracle database block*), které jsou tvořeny bloky operačního systému – každý Oracle blok je tvořen čtyřmi bloky operačního systému. Velikost Oracle bloků lze nastavit na hodnoty v rozmezí od 2 KB do 32 KB [14] [8].

Redo log soubory zaznamenávají veškerou aktivitu v databázi – výsledky transakcí a interní operace databáze. Pokud například nastane chyba vedoucí k pádu instance, některé změny v blocích nemusí být fyzicky zapsány do datových souborů. Zaznamenání těchto změn v redo log souborech se následně použije k vrácení změn do stavu před chybou,

čímž se zachovává transakční integrita. Redo log souborů pochopitelně může (a mělo by) existovat více z důvodu jejich možné ztráty či poškození. Statickou část kontrolního souboru lze ještě opravit či napsat znovu, u redo log souboru toto však nelze. Proto se opět doporučuje mít redo log soubory ve více kopiích umístěných na různých fyzických discích. SŘBD Oracle pak synchronně zapisuje změny do všech souborů – zápis je považován za úspěšný až v momentě, kdy jsou změny uloženy ve všech redo log souborech [8].

Archivní redo log soubory jsou v podstatě zálohy online redo log souborů. Pokud databáze běží v režimu *ARCHIVELOG*, je po každém úspěšném zápisu (změně) v redo log souboru vytvořena kopie tohoto souboru. Díky tomu lze snadno dohledat historii veškerých změn v databázi před nečekanou poruchou. Pokud redo log soubory nejsou archivovány, Oracle je neustále prochází dokola a přepisuje je, čímž znemožňuje evidovat historii změn [8]. Archivování redo log souborů je tedy kruciální pro případnou obnovu databáze do určitého bodu v minulosti.

3.3 Databáze PostgreSQL

3.3.1 Historie PostgreSQL

Předchůdcem systému řízení báze dat PostgreSQL byl systém Ingres (*Interactive Graphics and Retrieval System*), který vytvořili studenti a zaměstnanci Kalifornské univerzity v Berkeley v letech 1977 – 1985. Zároveň zde působil Michael Stonebraker, který se zabýval vývojem objektově-relačního databázového serveru s název Postgres. Tento projekt převzala společnost Illustra a vytvořila z něj komerční produkt. Systém byl následně doplněn dvěma studenty (Jolly Chen a Andrew Yu) o podporu jazyka SQL a přejmenován na Postgres95 [1].

V roce 1996 bylo zřejmé, že existuje velká poptávka po Open Source SQL databázovém serveru. Byl proto sestaven tým lidí, který pokračoval ve vývoji systému Postgres95. Členy týmu byli Marc Fournier, Thomas Lockhart, Vladim Mikheev a Bruce Momijian [15]. Prvotním úkolem bylo stabilizovat kód získaný z Berkeley. Jelikož systém byl původně vytvořen v akademickém prostředí, nebyl až doteď vystaven širokému spektru reálných dotazů. Byl proto hodně nestabilní a jednotlivé sekce kódu byly hodně rozmanité.

V roce 1996 byl systém přejmenován na PostgreSQL. Vývojáři začali kromě opravování chyb přidávat i nové funkce. V roce 1997 pak byla uvolněna první verze PostgreSQL 6.0. Od té doby je systém udržován a vyvíjen skupinou dobrovolníků z celého světa, kteří pracují

ve svém volném čase. Společně komunikují pomocí internetu. Sám Bruce Momjian (spoluautor systému) uvádí, že bez internetu by tento databázový systém nikdy nedosáhl svého úspěchu [1].

PostgreSQL také často podporují různé komerční subjekty. Například v roce 2000 sponzorovala firma Great Bridge (společnost založená původními investory Red Hat) několik vývojářů PostgreSQL a poskytla komunitě mnoho zdrojů. V roce 2001 však musela firma ukončit svou činnost kvůli špatným podmínkám na trhu.

Dalším aktivním podporovatelem je společnost Command Prompt, která projekt podporovala od roku 2001. V roce 2005 začala PostgreSQL podporovat společnost Sun Microsystems. Její operační systém Sun Solaris z roku 2006 zahrnoval PostgreSQL. Podpora však skončila v roce 2008 poté, co tuto firmu převzala společnost Oracle Corporation [16].

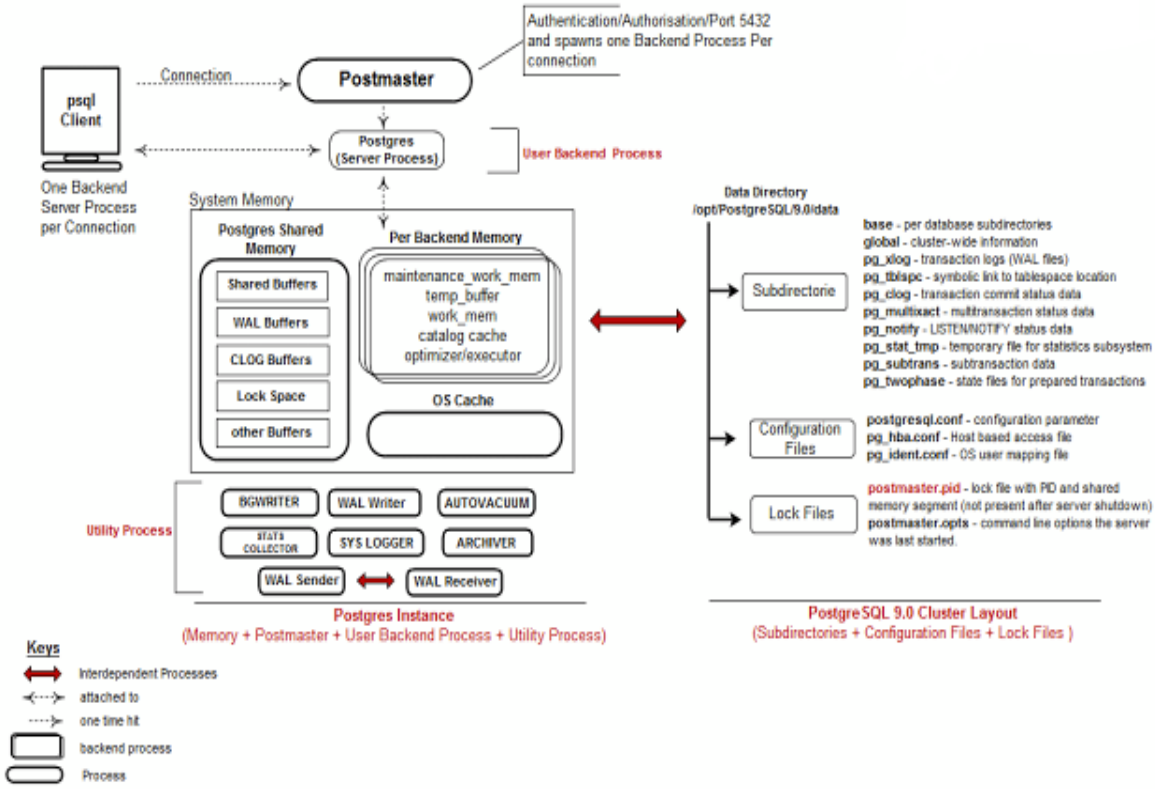
Zřejmě nejznámější firmou, která dnes PostgreSQL podporuje, je společnost EnterpriseDB. Ta na něm postavila svůj produkt Postgres Plus.

Komunita každoročně vydává velké a malé updaty, jejichž cílem je primárně opravovat chyby. SŘBD PostgreSQL není nikým vlastněn, je pouze podporován komunitou a komerčními subjekty. Běží pod BSD licencí, která umožňuje jeho bezplatné používání, modifikaci a distribuci pro komerční i nekomerční využití [17].

3.3.2 Struktura instance v SŘBD PostgreSQL

Hovořit o instanci v souvislosti se systémem řízení báze dat PostgreSQL je poněkud ošemetné. Architektura systému se sice skládá z procesů běžících v operační paměti a fyzických dat uložených na disku, ale termín instance se v souvislosti s SŘBD PostgreSQL příliš nepoužívá, byť ho některé zdroje používají [18].

Po spuštění databáze (technicky instance) se nejprve spustí proces zvaný *postmaster*. Ten obstarává všechna uživatelská připojení (*sessions*) a přiděluje jim proces zvaný *postgres*, který následně obsluhuje všechny uživatelské příkazy. Pro jednu databázi existuje vždy jeden proces *postmaster*. Procesů *postgres* může existovat více – dle počtu připojených uživatelů [19]. *Postmaster* zároveň po spuštění alokuje sdílenou paměť, do které mají přístup všichni uživatelé. Kromě přidělování procesů *postgres* uživatelům *postmaster* mnoho funkcí nemá, což je ovšem záměr – jeho selhání zpravidla způsobuje pád celé databáze, proto je hlavním cílem poskytnou mu co nejméně „důvodů“ pro pád [20].



Obrázek 4: Struktura instance v SŘBD PostgreSQL [20]

Jak je patrné z obrázku 4, vedle hlavních procesů (*postmaster* a *postgres*) běží v paměti ještě několik dalších obslužných procesů. BGWRITER (někdy označován pouze WRITER) slouží k zápisu dat do fyzických souborů. WAL Writer zapisuje data do WAL souborů. WAL (*Write-Ahead Logging*) soubory jsou soubory zajišťující transakční integritu. Filozofie je taková, že nejprve je nutné změny v datech zalogovat a až poté fyzicky přepsat daná data, jelikož v případě havárie vždy půjdou obnovit pomocí logů. Jde o jakousi obdobu redo log souborů z SŘBD Oracle [21].

STATS COLLECTOR slouží k získávání informací o objektech v databázi, které pak používá k monitorování výkonu. *CHECKPOINTER* je zodpovědný za tvorbu bodů pro obnovu databáze. Proces *AUTOVACUUM* spouští operaci *vacuum* (jejím cílem je „údržba“ databáze – například uvolňování místa zabraného smazanými řádky) v případě potřeby. Tyto (a další nezmíněné) procesy nejsou kruciální pro chod databáze a lze je v případě potřeby vypnout (na rozdíl od procesů uvedených v předchozím odstavci), nicméně je třeba dodat, že zastávají důležité funkce pro správný běh databáze [22].

Sdílená paměť je rozdělena do několika segmentů. Největší část zabírají *Shared Buffers*. Kdykoliv jsou čtena data z databáze, ukládají se do této vyrovnávací paměti. *WAL Buffers*

slouží k ukládání dat pro zápis do WAL souborů. *Lock space* slouží k ukládání zámků – paměťové struktury v paměti jsou chráněny „lehkými“ zámky, zatímco tabulky jsou chráněny pomocí „těžkých“. Dále se v paměti nachází ještě několik dalších bufferů [20].

Za zmínku stojí také to, že se ve sdílené paměti nenacházejí vyrovnávací paměti knihoven a datového slovníku. Což mimo jiné znamená, že pokud je nějaký dotaz zadán několikrát, musí být pokaždé naparsován a proveden dle exekučního plánu. V paměti se nenachází segment, který by ukládal výsledky tohoto typu dotazu pro opakované použití [20].

Vedle sdílené paměti existuje ještě jedna paměť – *process local memory*. Tu si alokuje každý proces (*postgres*) zvlášť a slouží pouze pro jeho interní potřeby. Ostatní procesy ji nevidí, tudíž nehrozí, že by ji mohly nějak poškodit.

3.3.3 Struktura fyzické databáze PostgreSQL

Z fyzického hlediska jsou všechny soubory potřebné pro chod databáze umístěny v adresáři obvykle pojmenovaném PGDATA. Jeho obvyklé umístění je *var/lib/pgsql/data*. Tento adresář obsahuje mnoho podadresářů a kontrolní soubory, viz tabulka:

`PG_VERSION`	Soubor obsahující hlavní číslo verze PostgreSQL
`base`	Obsahuje soubory databáze
`global`	Podadresář obsahující sdílené tabulky, jako `pg_database`
`pg_clog`	Podadresář obsahující data o stavu potvrzení (commit) transakcí
`pg_multixact`	Podadresář obsahující data o multitransakčním stavu (užívaná pro sdílené zámky řádků)
`pg_notify`	Podadresář obsahující data o stavu LISTEN/NOTIFY
`pg_serial`	Podadresář obsahující informace o potvrzených serializovatelných transakcích
`pg_stat_tmp`	Podadresář obsahující dočasné soubory statistického podsystému
`pg_subtrans`	Podadresář obsahující data o stavu subtransakcí
`pg_tblspc`	Podadresář obsahující symbolické odkazy do tabulkových prostorů
`pg_twophase`	Podadresář obsahující soubory pro připravené transakce
`pg_xlog`	Podadresář obsahující WAL (Write Ahead Log) soubory
`postmaster.opts`	Soubor zaznamenávající volby příkazové řádky, se kterou byl server naposledy spuštěn
`postmaster.pid`	Soubor zaznamenávající ID současného postmaster procesu (PID), cestu k adresáři clusterových dat, čas startu postmasteru, číslo portu, Cestu k socket Unixové domény (prázdná ve Windows), první validní

	listen_address (IP adresa nebo *, nebo prázdné, pokud se neposlouchá přes TCP), a ID segment sdílené paměti (po vypnutí soubor zmizí)

Tabulka 1: Obsah adresáře PGDATA [21]

Adresář base (pojmenován dle OID – Object identifier – databáze) je základním umístěním všech databázových souborů pro danou databázi. Nacházejí se zde například systémové katalogy. Každá tabulka a index jsou umístěny do samostatného souboru. Tabulkové prostory jsou reprezentovány adresářem obsahujícím relevantní objekty. Byť je tabulkový prostor logický konstrukt, fyzické soubory tabulek a indexů jsou umístěny v adresáři příslušného tabulkového prostoru. V adresáři tabulkového prostoru mohou být tabulky z různých databází (v případě používání clusteru). V tom případě jsou rozděleny do podadresářů, které jsou pojmenovány podle názvu a verze databáze [21].

3.4 Společné vlastnosti obou databází

Cílem této kapitoly je vytvořit stručný přehled toho, co mají oba SŘBD společné. Většinou je zde popsána základní funkcionalita, kterou by měli nabízet všechny dostupné systémy, případně některé odlišnosti, které však nejsou natolik markantní, aby se jimi bylo nutné zabývat v analytické části.

Některé oblasti, které lze téměř jistě zařadit do základní funkcionality systémů řízení báze dat, zde naopak popsány nejsou – to proto, že rozdíly mezi oběma systémy v řešení těchto funkcionalit byly natolik signifikantní, že jsou podrobně rozebrány v analytické části. To je záležitost především správy uživatelů (kapitola 4.3.1), aktivačních událostí (4.3.2) a zálohování (4.3.4).

3.4.1 Práce s tabulkami

Obě SŘBD pochopitelně nabízejí sadu příkazů pro práci s tabulkami. Příkazy typu *CREATE TABLE, ALTER TABLE, DROP TABLE* či *SELECT x FROM* jsou v obou systémech totožné. Řádky v tabulkách je možno měnit klasickým příkazem *UPDATE* a mazat pomocí příkazu *DELETE*. Výsledky dotazu *SELECT* je možno řadit pomocí klauzule *ORDER BY*. Omezit výčet vrácených řádků jde pomocí klauzulí *WHERE* a *HAVING*. V případě klauzule *HAVING* je použití totožné v obou systémech – tedy tehdy, pokud omezení vrácených řádků vyplývá z výsledků nějaké agregační funkce aplikované

na původní soubor. Oba SŘBD podporují poddotazy, včetně korelovaných. Stejně tak operátory *EXISTS, NOT EXISTS, NOT IN* a *IN*.

V obou systémech je možno tabulky spojovat pomocí primárních a cizích klíčů. Řádky z více tabulek lze získat pomocí klauzule *JOIN ON*. Podporovány jsou klasické *INNER JOINS* (spojení řádků, pro které existují hodnoty) a také *OUTER JOINS* (k řádkům, které nemají odpovídající hodnotu v druhé tabulce je přiřazena hodnota *NULL*). Rozlišujeme *LEFT OUTER JOIN* (ke všem řádkům z „levé“ tabulky se připojí hodnoty z „pravé“ tabulky, případně hodnoty *NULL*) a analogický *RIGHT OUTER JOIN*. Oba SŘBD také podporují *FULL OUTER JOIN* a *SELF JOIN* (napojení tabulky na sebe samu).

3.4.2 Datové typy a práce s nimi

V obou databázích lze pracovat se sloupci a proměnnými (v případě procedurálních rozšíření, viz kapitola 3.4.4) různých datových typů. Existují datové typy pro text (*CHAR, VARCHAR, VARCHAR2*), čísla (*NUMBER, INTEGER*), datum (*DATE*), objemná data (*BLOB, CLOB*) a „surová“ data (*RAW*).

Patrně nejzásadnějším rozdílem je neexistence datového typu *BOOLEAN* v SŘBD Oracle. Ten však lze vždy nahradit textovým řetězcem s odpovídajícími hodnotami. Poněkud odlišné jsou i funkce pro datový typ *DATE*, které se však liší především syntaxí. Ve výsledku je funkčnost nabízená oběma databázemi podobná.

Pro práci s řetězci je k dispozici celá řada funkcí. Funkce *RPAD* a *LPAD* pro doplňování řetězce znaky, *TRIM* pro jeho krácení, *LOWER* a *UPPER* pro konverzi velkých znaků na malé a obráceně, *SUBSTRING* pro hledání řetězce v jiném. Některé další funkce jsou uvedeny v kapitole 4.2.3. V obou databázích existuje ještě celá řada dalších funkcí pro práci s řetězci, nicméně nejsou již tolik signifikantní a používané.

Obě databáze obsahují i širokou škálu matematických funkcí pro práci s čísly. Základem je běžné sčítání, odčítání, násobení a dělení pomocí příslušných znamének. Dále jsou k dispozici funkce jako *ABS* (absolutní hodnota čísla), *FLOOR* (největší celé číslo menší než zadané), *CEIL* (nejmenší celé číslo větší než zadané), *POWER* (mocnina čísla), *SQRT* (druhá odmocnina čísla), *LOG* a *LN* (logaritmické funkce) a dále všechny trigonometrické funkce (*SIN, COS, TAN* apod.). Nechybí ani funkce pro zaokrouhlování (*ROUND*) a uříznutí části čísla za desetinnou čárkou (*TRUNC*). Dále všechny agregační funkce *AVG* (průměr), *COUNT* (počet), *SUM* (suma), *MAX* (maximum) a *MIN* (minimum).

3.4.3 Transakční zpracování

Transakce lze v obou databázích řídit pomocí vytváření uložených bodů (*SAVEPOINT*), potvrzování změn (*COMMIT*) a vracení změn (*ROLLBACK*). Operace *ROLLBACK* vrací všechny nepotvrzené změny, nebo ji lze použít k návratu k uloženému bodu (*ROLLBACK TO SAVEPOINT*). Funkcionalita v této oblasti je v obou databázích téměř totožná, dokonce i včetně nepovinného slůvka *work* v klauzuli *rollback – ROLLBACK [WORK]* … [29] [21]. Toto slovo nemá na funkcionalitu dotazu vůbec žádný vliv, existuje pouze kvůli dodržení standardů.

3.4.4 Procedurální rozšíření

SŘBD Oracle obsahuje procedurálně jazykovou nadmnožinu PL/SQL (PL - Procedural Language) [12]. Tento jazyk rozšiřuje standardní dotazovací jazyk SQL o procedurální funkce.

Kód jazyka PL/SQL se skládá z bloků. Blokem může být procedura (nevrací hodnotu), funkce (vrací hodnotu) nebo balík (sdružuje/zapouzdřuje funkce a procedury). Pokud blok kódu nemá název, jde o anonymní blok, který se nikam neukládá. První část bloku kódu je tzv. deklarace (*DECLARE*). Ta slouží k definování názvu a datového typu proměnných, které jsou v daném bloku použity. Lze jim nastavit i defaultní hodnotu.

V dalším oddíle se nacházejí spustitelné příkazy. Tento oddíl začíná klíčovým slůvkem *BEGIN*. Zde se pracuje se vstupními parametry a proměnnými pomocí klasických procedurálních nástrojů – podmínek a smyček (*WHILE, FOR*). Je zde tedy obsažena veškerá logika a funkcionalita daného bloku kódu.

Další oddíl slouží ke správě výjimek. Začíná klíčovým slovem *EXCEPTION* a lze v něm nadefinovat, jaká akce se provést, nastane-li v předešlém bloku nějaká chyba. Dají se použít předem definované výjimky v systému anebo uživatelsky definované. Nakonec je blok kódu zakončen slovem *END*.

Výsledný blok kódu PL/SQL může vypadat například takto:

```
CREATE FUNCTION prictipet (cislo IN NUMBER) RETURN NUMBER IS
   var_result NUMBER(10);
BEGIN
   var_result := cislo + 5;
   RETURN var_result;
END;
```

Tato modelová funkce vezme číslo na vstupu (parametr *cislo*) a přičte k němu pět. Následně vrátí výsledek. Slovo *DECLARE* zde bylo nahrazeno klauzulí *CREATE FUNCTION*, což znamená, že tato funkce bude uložena v databázi pro opětovné použití. Pomocí tohoto jazyka je možno psát aplikační logiku, vytvářet aktivační události (viz kapitola 4.3.2) a uživatelské funkce. Přístup k těmto funkcím je pak možno řídit pomocí přístupových oprávnění anebo viditelnosti balíků [29].

Databáze PostgreSQL nabízí svou vlastní obdobu výše uvedeného jazyka – jazyk PL/pgSQL. Taktéž zde hrají hlavní roli kódy bloku, které mají i stejné členění – oddíl deklarací *(DECLARE*), oddíl spustitelného kódu (*BEGIN*) a oddíl výjimek. Syntaxe je velmi podobná jazyku PL/SQL. Největším rozdílem je to, že zde nejdou vytvářet procedury, respektive jsou stejně jako funkce vytvářeny příkazem *CREATE FUNCTION*, a také to, že za každý blok kódu je nutno specifikovat, v jakém jazyce byl napsán. Kód jazyka PL/pgSQL může vypadat následovně [21]:

```
CREATE FUNCTION sales_tax(subtotal real) RETURNS real AS $$
BEGIN
    RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

3.4.5 Schémata

Pojem schéma lze v souvislosti s SŘBD Oracle vymezit jako kolekci všech objektů, které patří jednomu uživateli. Sem mohou patřit tabulky, funkce, balíky či pohledy. Vznik schématu je pevně svázán se vznikem uživatele, jiným způsobem vzniknout nemůže. Za schéma tedy lze považovat i všechny systémové tabulky a funkce, které zajišťují chod databáze – zpravidla patří uživateli *SYSTEM*. Analogicky lze považovat všechny objekty v databázi za schéma dané databáze. Uživatel může přistupovat (pomocí tečkové konvence) k objektům z jiného schématu pouze v případě, že k tomu má odpovídající oprávnění.

Oproti tomu v SŘBD PostgreSQL je schéma prostředek k logickému rozčlenění dat. Schéma může obsahovat tabulky či pohledy. Jedna tabulka však může být součástí více schémat. Pokud má uživatel dostatečná oprávnění, může vytvářet objekty ve schématu vlastněném jiným uživatelem. Tyto vlastnosti lze využít k řízení přístupu k datům. V SŘBD PostgreSQL je naprosto validní příkaz *CREATE SCHEMA*. Pomocí parametru *AUTHORIZATION* tohoto příkazu lze například vytvořit schéma, jehož vlastníkem bude jiný

uživatel [21]. Implicitně v databázi PostgreSQL také existuje schéma *public*, kam jsou automaticky ukládány tabulky vytvořené bez specifikace schématu. Důrazně se doporučuje neukládat do tohoto schématu tabulky.

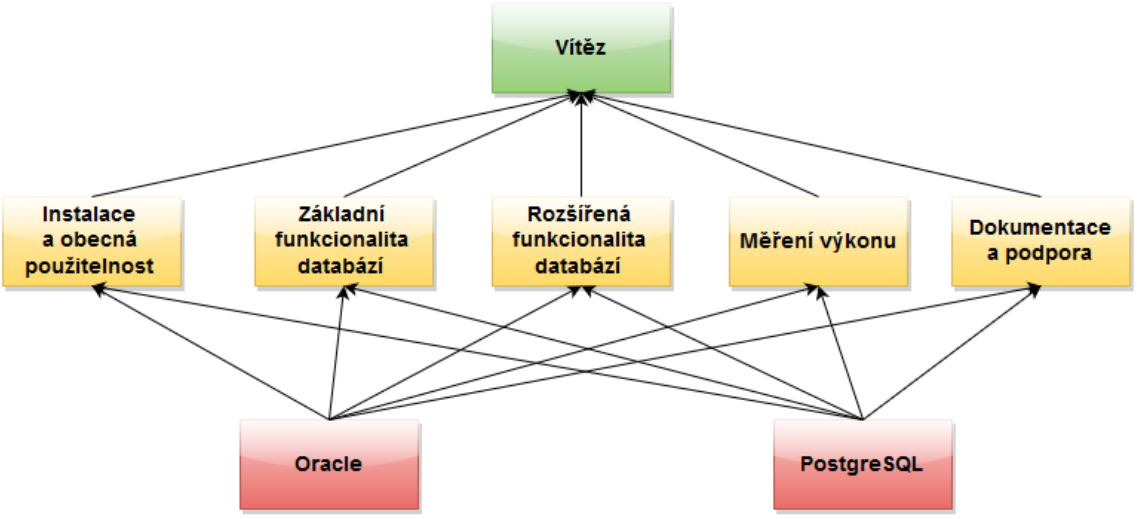
4. Analytická část

4.1 Metodika analytické části

4.1.1 Postup analýzy

V následující části práce jsou oba systémy řízení báze dat srovnány pomocí metod vícekriteriálního rozhodování. Jsou zde uvedena jednotlivá kritéria, pomocí kterých jsou obě databáze srovnávány. Tato kritéria jsou rozdělena do skupin. Váhy jsou jednotlivým kritériím přiděleny pomocí metody kvantitativního párového srovnávání kritérií (Saatyho metody, viz následující kapitola). V dalším kroku jsou pak váhy stejným způsobem přiřazeny i skupinám kritérií, které jsou pak mezi sebou porovnány pro výsledné pořadí. Je zde tedy použita metoda *AHP – Analytic Hierarchy Process*, jejímž autorem je taktéž Thomas Saaty [23].

V této metodě je rozhodovací proces znázorněn jako hierarchická struktura o několika úrovních. Úrovně jsou seřazeny od obecného ke konkrétnímu. Nejvyšší úrovní (první) je cíl analýzy, tedy vlastní rozhodnutí. Druhou úrovní jsou jednotlivé skupiny faktorů, které mají vliv na rozhodovací proces. Třetí úrovní jsou kritéria, která spadají do výše uvedených skupin. Poslední (čtvrtou) úrovní jsou pak jednotlivé varianty, v tomto případě systémy řízení báze dat.



Obrázek 5: Hierarchická struktura úlohy vícekriteriálního rozhodování [vlastní tvorba]

4.1.2 Saatyho metoda stanovení vah

Jak již bylo řečeno, k určení vah jednotlivých kritérií je použita metoda kvantitativního párového srovnávání (Saatyho metoda). Ta zkoumá preferenční vztahy dvojic kritérií. Kromě směru preference kritéria se určuje i velikost této preference. Tuto velikost je doporučeno vyjadřovat pomocí následující bodové stupnice (která je v práci použita):

Vyjádření preferencí	
číselné	slovní
1	kritéria jsou stejně významná
3	první kritérium je slabě významnější než druhé
5	první kritérium je silně významnější než druhé
7	první kritérium je velmi silně významnější než druhé
9	první kritérium je absolutně významnější než druhé
Hodnoty 2,4,6,8 slouží jako mezistupně pro citlivější vyjádření preferencí	

Tabulka 2: Bodová stupnice velikosti preference Saatyho metody [26]

Velikost preference i-tého kritéria oproti j-tému se pak zapíše do Saatyho matice S. Tato matice je čtvercová a její prvky obsahují odhady podílů vah kritérií, tedy kolikrát je jedno kritérium významnější než jiné [26].

Následně se spočítá Geometrický průměr a normovaná váha (Vážený geometrický průměr) pro každé kritérium dle následujících vzorců:

$$G_i = \sqrt[n]{x_1 * x_2 * \dots * x_n}$$

$$V_i = \frac{G_i}{\sum G}$$

Po získání vah jednotlivých kritérií jsou následně stejnou metodou získány váhy jednotlivých výsledků (kolikrát test dopadl lépe pro jeden SŘBD vůči druhému). Na základě vynásobení váhy kritéria s váhou výsledku je pak zjištěno výsledné pořadí. Vzhledem k tomu, že jsou kritéria rozdělena do skupin, jsou nejprve počítány výsledky uvnitř skupin a následně i výsledky mezi skupinami.

4.1.3 Testovací prostředí

Jako testovací prostředí byl zvolen operační systém CentOS 7 spuštěný virtuálně pomocí programu VirtualBox. CentOS (*Community Enterprise Operating System*) je linuxová distribuce založená na Red Hat Enterprise linuxu [27]. Zatímco Red Hat Enterprise Linux je k dispozici pouze platícím zákazníkům firmy Red Hat, CentOS je k dispozici zcela volně především i díky povinnosti firmy Red Hat umisťovat zdrojové kódy svého operačního systému na web. Oficiálně touto firmou CentOS podporován není, stojí za ním však skupina komunitních vývojářů, která vydává pravidelné aktualizace.

Verze CentOS 7 je nejnovější verzí tohoto systému. Byla vydána v červenci roku 2014. Pomocí programu VirtualBox byly vytvořeny dvě totožné instalace tohoto operačního systému, z nichž každá obsahovala jeden SŘBD (Oracle a PostgreSQL). Díky možnostem nastavení tohoto programu měly oba operační systémy k dispozici stejný virtuální hardware – jedno jádro 64-bitového procesoru AMD Vishera FX-6300 (frekvence 3,5 GHz), 2048 MB operační paměti DDR3 a zhruba 20 GB místa na disku.

SŘBD Oracle byl zvolen ve verzi 11g (release 11.2.0.4) Enterprise edition. Existuje i novější verze (12c), nicméně ta ještě není příliš rozšířená a vyladěná. Databáze je k dispozici zcela zdarma za podmínky, že bude použita pouze k testování, vývoji a prezentaci nekomerčních produktů a nebude použita k žádným jiným účelům (především komerčním a produkčním) [28].

SŘBD PostgreSQL byl použit ve verzi 9.3.5. Tato verze byla vydána v červenci roku 2014. Taktéž je k dispozici zcela zdarma.

4.2 Instalace a obecná použitelnost

Tato sekce si klade za cíl zhodnotit instalaci databáze a její provoz z uživatelského hlediska. Jde o první skupinu kritérií, která je použita ve výsledném srovnání.

4.2.1 Podporované platformy

Databázi Oracle 11g lze nainstalovat na těchto operačních systémech: Linux (Oracle Enterprise Linux, Linux for system Z, SUSE Linux, Red Hat Linux), Unix (Solaris, AIX, HP-UX), Windows (Server i běžné operační systémy), Mac OSX, z/OS (Operační systém firmy IBM, někdy také označován OS/390) a OpenVMS (Open Virtual Memory Systém) [24]. Nejsou podporovány systémy BSD (Berkeley Software Distribution), Android, AmigaOS, Symbian a iOS.

PostgreSQL podporuje všechny současné distribuce Linuxu, Unix (Solaris, AIX, HP-UX, IRIX, Tru64 Unix a UnixWare), BSD, Mac OSX, Windows a Android [21]. Nepodporuje systémy z/OS, iOS, OpenVMS, AmigaOS a Symbian.

	OpenVMS	BSD	Android	z/OS
Oracle	ANO	NE	NE	ANO
PostgreSQL	NE	ANO	ANO	NE

Tabulka 3: Srovnání podpory platforem obou databází [vlastní tvorba]

V tabulce výše jsou uvedené pouze platformy, v jejichž podpoře se oba systémy řízení báze dat liší. Jak vyplývá z tabulky, oba SŘBD mají rovnocenný počet nepodporovaných platforem oproti druhému. Zároveň je třeba dodat, že PostgreSQL lze na z/OS spustit přes virtualizované prostředí linuxu, nativně ovšem toto prostředí podporované není. Dá se tedy hovořit o jakési poloviční podpoře této platformy [25]. Když k tomu připočteme to, že SŘBD PostgreSQL oficiálně podporuje více linuxových distribucí, vychází v tomto srovnání s SŘBD Oracle o něco lépe.

4.2.2 Instalační proces

Následující řádky jsou subjektivního charakteru, vyjadřují spíše autorovy zkušenosti, než obecně platné informace, je tedy nutno brát je s určitou rezervou.

Instalace SŘBD Oracle 11g v linuxovém prostředí patří mezi náročnější procesy. Lze konstatovat, že uživatel bez alespoň základních zkušeností s linuxovým prostředím by ho nezvládl.

Prvním krokem je nastavení parametrů kernelu (jádra operačního systému) v souboru /etc/sysctl.conf. Následně je třeba vytvořit nového uživatele (běžně *oracle*) v systému a přiřadit ho do nově vytvořených skupin (*oinstall, dba, oper*). Dalším krokem je pak instalace balíků, které jsou potřebné pro chod databáze, ale nejsou obsaženy v základní distribuci daného operačního systému. Ty se mohou nainstalovat buď z instalačního DVD operačního systému, nebo je lze stáhnout příkazem *yum* z internetu, je-li tento příkaz podporován (například *yum -y install unixODBC*). Seznam všech potřebných balíků lze najít buď v oficiální dokumentaci databáze, nebo na internetu v různých instalačních návodech.

Dalším krokem je pak stažení instalačního souboru (obvykle má dvě části) databáze, rozbalení, a spuštění vlastního instalačního procesu. Během instalace je nutno spustit některé skripty umístěné v nově vytvořených adresářích jako uživatel *root*, což je opět operace, kterou naprosto běžný uživatel nemusí zvládnout. Po instalaci následuje i vytvoření a konfigurace vlastní databáze, byť tento krok je volitelný a je možno ho provést později.

Je třeba dodat, že grafická prostředí (například GNOME) pro distribuce operačních systému Linux instalaci SŘBD hodně zjednodušují – díky nim je možno použít Oracle Universal Installer, požadované soubory editovat přes grafický textový editor (nikoliv pouze přes příkazy v terminálu) apod. I tak se ale často během instalace vyskytnou chyby, které by se měly řešit – SŘBD Oracle se sice často nainstaluje úspěšně i při ignorování chyb, nicméně to může mít v budoucnu dopad na jeho funkčnost, což je nežádoucí.

Instalace SŘBD PostgreSQL 9.3 na témže operačním systému je mnohem jednodušší záležitost. Pomocí příkazu *yum* se automaticky stáhnou a nainstalují příslušné repositáře. Vhodné je aktualizovat i všechny stávající pomocí příkazu *yum -y update*. Dalším krokem je pak samotná instalace vlastního systému řízení báze dat, která proběhne zcela automaticky opět pomocí příslušného příkazu *yum*.

Instalace v testovacím prostředí ovšem nebyla bezproblémová – napoprvé se vyskytla chyba a bylo nutné doinstalovat jeden repositář ručně. Pak již instalace proběhla v pořádku. Na rozdíl od databáze Oracle zde instalační proces automaticky vytvoří uživatele *postgres*, který slouží jako hlavní řídící účet. Po nainstalování je třeba databázi spustit inicializovat a spustit, což je záležitost dvou příkazů:

```
/usr/pgsql-9.3/bin/postgresql93-setup initdb
systemctl start postgresql-9.3
```

Poté se lze již přihlásit jako uživatel *postgres* a započít práci.

Jak z předchozích řádků vyplývá, instalační proces je mnohem jednodušší u SŘBD PostgreSQL – k dokončení instalace je potřeba mnohem menší počet příkazů, vše je co nejvíce zautomatizováno, instalace je téměř bezproblémová. Opět je potřeba alespoň elementární znalost práce s linuxovým prostředím, ale k úspěšné instalaci stačí mnohem menší znalost prostředí, než v případě SŘBD Oracle.

Při instalaci databáze Oracle se nastavuje několik parametrů – například domovský adresář (*ORACLE_HOME*), heslo pro uživatele *SYSTEM, SID* (System ID) a několik dalších. Existuje zde tedy jakýsi požadavek, že uživatel musí vědět, co jednotlivé parametry znamenají a jaké jim má nastavit hodnoty. Dále je potřeba provést několik nastavení v systému ještě před vlastní instalací – toto celou dobu instalace prodlužuje a činí ji poněkud náročnější, než v případě databáze PostgreSQL.

Instalace na totožném operačním systému zabrala (od započetí práce přes vyřešení všech chyb až po přihlášení do databáze jako administrátor) v případě SŘBD Oracle zhruba tři hodiny, zatímco u SŘBD PostgreSQL to byla pouhá hodina. Časy jsou pochopitelně spíše ilustrativního charakteru, nicméně mohou posloužit pro pochopení náročnosti instalačního procesu.

4.2.3 Odlišnosti v syntaxi a užívání databáze

Byť syntaxe nepatří mezi důležité faktory při srovnání dvou databází (uživatel se ji vždy může doučit), přišlo autorovi vhodné upozornit na některé odlišnosti při práci s těmito SŘBD, jelikož často mohou mít dopad na celkový dojem ze systému.

SŘBD PostgreSQL umožňuje vytvořit novou tabulku se strukturou stávající tabulky příkazem:

```
CREATE TABLE tabulka2 ( LIKE tabulka1);
```

SŘBD Oracle toto neumožňuje, je nutno použít následující konstrukt:

```
CREATE TABLE tabulka2 AS SELECT * FROM tabulka1 WHERE 1<>1;
```

Rozdíl je i v limitování počtu zobrazených řádků z výsledku dotazu. V případě PostgreSQL stačí jednoduše zadat:

```
SELECT sloupce FROM tabulka ORDER BY sloupec LIMIT n;
```

V případě databáze Oracle je nutno použít pseudosloupec (umělý sloupec) *ROWNUM* nebo funkci *ROW_NUMBER*, což činí dotaz komplikovanějším – v případě opačné nerovnosti totiž dotaz nevrátí žádné řádky.

```
SELECT username FROM dba_users WHERE rownum < 10.
```

I vložení více řádků do tabulky jedním příkazem je v případě SŘBD Oracle mnohem komplikovanější záležitost. SŘBD PostgreSQL si vystačí s tímto dotazem:

```
INSERT INTO tabulka VALUES (0, 'hodnota0') , (1, 'hodnota1') , (2,
'hodnota2');
```

V SŘBD Oracle je nutno použít buď *UNION* (sjednocení množin) v kombinaci s příkazem *INSERT AS SELECT*, nebo příkaz *INSERT ALL*:

```
INSERT INTO tabulka
   SELECT 0,'hodnota0' FROM DUAL
         UNION ALL
   SELECT 1, 'hodnota1' FROM DUAL
         UNION ALL
   SELECT 2, 'hodnota2' FROM DUAL;

INSERT ALL
   INTO tabulka VALUES (0, 'hodnota0')
   INTO tabulka VALUES (1, 'hodnota1')
   INTO tabulka VALUES (2, 'hodnota2')
SELECT NULL FROM DUAL;
```

Další zajímavostí jsou tzv. umělé tabulky *(Dummy tables)*. V SŘBD Oracle je velice často využívána tabulka *DUAL*, jež je nutná například pro jednoduchý matematický výpočet:

```
SELECT 1+1 FROM DUAL;
```

Oproti tomu databáze PostgreSQL tyto tabulky nevyužívá, na vstup stačí zadat pouze:

```
SELECT 1+1;
```

Odlišná je i práce se sekvencemi (*SEQUENCE*) v obou systémech. Pro následující hodnotu se v SŘBD Oracle používá konstrukt *jmenosekvence.NextVal*, zatímco v PostgreSQL se používá *nextval('jmenosekvence')*.

Další odlišností je příkaz *MINUS* či *EXCEPT* při práci s množinami. Jediný rozdíl je v klíčkovém slůvku – SŘBD Oracle používá *MINUS*:

```
<Dotaz1> MINUS <Dotaz2>;
```

a databáze PostgreSQL používá *EXCEPT*:

```
<Dotaz1> EXCEPT <Dotaz2>;
```

Z hlediska funkčnosti vracejí oba dotazy stejný výsledek – výsledek dotazu *Dotaz1*, od kterého jsou odebrány výsledky dotazu *Dotaz2*.

Jak je vidět, mezi databázemi existuje hodně odlišností i na syntaktické úrovni, které pak ovlivňují každodenní práci s databází. Nedá se určit, který zápis některých příkazu je „lepší". Ale z uvedených příkladů je vidět, že databáze PostgreSQL se více zaměřuje

na jednoduchost a snadnou čitelnost příkazů, zatímco v případě databáze Oracle je občas nutné zadat k získání jednoduchého výsledku poměrně složitý dotaz.

4.2.4 Zhodnocení sekce Instalace a obecná použitelnost

Pro zhodnocení sekce bylo nejprve nutno spočítat váhu jednotlivých kritérií. Počet platforem, na kterých lze databáze nainstalovat je poměrně důležitý ukazatel. Naopak průběh instalace není považován za kruciální, jelikož SŘBD zpravidla instalujeme pouze jednou. Syntaxe leží svou důležitostí někde mezi těmito ukazateli – není zcela zásadní, nicméně má dopady na každodenní používání systému. Tabulka s výpočtem vah kritérií vypadá následovně:

	K1	K2	K3	G	V	Pořadí
K1	1	5	3	2,466212	0,636986	1.
K2	1/5	1	1/3	0,40548	0,104729	3.
K3	1/3	3	1	1	0,258285	2.
			suma:	3,871692		

K1 – Počet podporovaných platforem
K2 – Průběh instalace
K3 – Syntaktické rozdíly
G – Geometrický průměr
V – Vážený geometrický průměr (váha kritéria)

Tabulka 4: Výpočet vah kritérií sekce Instalace a obecná použitelnost [vlastní tvorba]

Jak z tabulky vyplývá, počet platforem je kritérium silně preferované před instalací a slabě preferované před syntaxí. Syntaxe je pak slabě preferována před průběhem instalace. Oba SŘBD pak byly ohodnoceny následovně (vychází se ze slovního hodnocení uvedeného v příslušných kapitolách):

	Oracle	PostgreSQL	G	V
Oracle	1	1/3	0,57735	0,25
PostgreSQL	3	1	1,732051	0,75
		suma:	2,309401	

Tabulka 5: Kritérium 1 – hodnocení z hlediska podporovaných platforem [vlastní tvorba]

	Oracle	PostgreSQL	G	V
Oracle	1	1/5	0,447214	0,166667
PostgreSQL	5	1	2,236068	0,833333
		suma:	2,683282	

Tabulka 6: Kritérium 2 – hodnocení z hlediska průběhu instalace [vlastní tvorba]

	Oracle	PostgreSQL	G	V
Oracle	1	1/2	0,707107	0,333333
PostgreSQL	2	1	1,414214	0,666667
		suma:	2,12132	

G – Geometrický průměr

V – Vážený geometrický průměr (váha)

Tabulka 7: Kritérium 3 – hodnocení z hlediska syntaxe [vlastní tvorba]

Jak vyplývá ze slovního hodnocení, ve všech kritériích v této sekci byl lepší SŘBD PostgreSQL. Nejvíce v případě druhého kritéria, kde byl silně preferovaný před SŘBD Oracle. Ve zbývajících kritériích již šlo pouze o slabou či minimální preferenci. Zhodnocení obou SŘBD za celou sekci (jsou zohledněny i váhy kritérií) je vidět na následující tabulce:

Kritérium	Váha	Oracle	PostgreSQL
K1	0,636986	0,25	0,75
K2	0,104729	0,166667	0,833333
K3	0,258285	0,333333	0,666667
	Celkem:	0,262796	0,737204

K1 – Počet podporovaných platforem

K2 – Průběh instalace

K3 – Syntaktické rozdíly

Tabulka 8: Zhodnocení sekce Instalace a obecná použitelnost [vlastní tvorba]

Výsledná hodnota se spočítá vždy jako součet všech výsledků násobených vahou daného kritéria. Je vidět, že v této sekci byl lepší systém řízení báze dat PostgreSQL (více je lépe).

4.3 Základní funkcionalita databází

Tato sekce se zaměřuje na zhodnocení vybraných klíčových vlastností obou SŘBD, které mají velký vliv na každodenní práci s databází. Jsou vybrány především funkce, které jsou implementovány odlišně v obou systémech (nebo v některé nejsou implementovány vůbec).

4.3.1 Správa uživatelů

V obou databázích je základním pilířem pro vytváření uživatelských účtů příkaz *CREATE USER*. Tento příkaz se dá zadat s mnoha různými parametry, jež se v obou databázích liší.

V SŘBD Oracle se pomocí parametrů volí jméno účtu, heslo, výchozí (*default*) tabulkový prostor, dočasný tabulkový prostor, kvóta na tabulkový prostor, profil a typ účtu (zamčený/otevřený) [29].

Uživatelskému účtu lze pak přidělovat práva – buď přímo, anebo pomocí rolí. Role je množina oprávnění, které mají všichni její vlastníci. Díky ní jde řídit přístupová práva mnoha uživatelů současně. Role se vytvoří pomocí příkazu *CREATE ROLE* a práva jí lze přidělit klasickým příkazem *GRANT*.

Vedle rolí pak v SŘBD Oracle existují ještě tzv. profily. Profil je množina limitů pro přístup k databázovým zdrojům. Nastavit lze například maximální počet připojení (*session*) daného uživatele, maximální délku připojení, maximální čas nečinnosti před odhlášením (*idle time*), limity na využití procesoru, životnost hesla a další. Profily se používají k řízení přístupu k databázi jako celku. Mohou zlepšit bezpečnost a výkon databáze.

V SŘBD PostgreSQL je od verze 8.1 naprosto odlišný přístup k uživatelům. V této verzi došlo ke spojení uživatelů (*users*) a skupin (*groups*) do rolí. SQL standard definuje uživatele a role jako dva odlišné koncepty, nicméně v PostgreSQL je uživatel a role ta samá entita. Příkazy *CREATEUSER* a *CREATE ROLE* jsou tedy ekvivalentní kromě toho, že příkaz *CREATEUSER* implicitně přidá právo na přihlášení (*LOGIN*). Stejně tak dříve používaný příkaz *CREATE GROUP* pro vytvoření skupiny je dnes pouze aliasem pro příkaz *CREATE ROLE* [21].

V SŘBD PostgreSQL může být tedy role buď uživatelem, nebo entitou obsahující další uživatele (a role). Podobně jako v SŘBD Oracle jí lze přiřadit oprávnění pro přístup k objektům. Pomocí parametru *INHERIT* lze pak nastavit, zda nějaká role zdědí práva skupiny (role), jíž je členem. Parametr *LOGIN* pak určuje, zda se může role přihlásit do databáze (a tedy se obrazně řečeno stát uživatelem).

Byť je přímo v oficiální PostgreSQL dokumentaci uvedeno, že tento přístup odporuje SQL standardům, je nutno konstatovat, že jde o přístup inovativní. Tímto sjednocením došlo ke zjednodušení procesu přidávání oprávnění jiným uživatelům/rolím a zároveň nepřineslo

žádné nevýhody. Problém může představovat až složitá struktura rolí uvnitř jiných rolí, které od sebe dědí různá práva, což ve výsledku může způsobit nepřehlednost struktury uživatelských oprávnění. Toto však závisí již na vlastní implementaci v praxi a na práci daného vývojáře, tudíž to nelze považovat za obecnou nevýhodu SŘBD PostgreSQL.

4.3.2 Aktivační události (Triggers)

SŘBD Oracle rozlišuje dle úrovně několik typů aktivačních událostí:

- Na úrovni řádků – klauzule *FOR EACH ROW*.
- Na úrovni příkazu – například při klauzuli *INSERT*.
- Na úrovni schématu – například při klauzuli *CREATE TABLE*.
- Na úrovni databáze – například při odhlášení uživatele.

Dále je v SŘBD Oracle rozlišeno, zda se aktivační událost provede před událostí (*BEFORE*) anebo až po ní (*AFTER*). Dále zde existuje klauzule *INSTEAD OF*. Aktivační události obsahující tuto klauzuli provedou nějakou definovanou akci místo původní.

Aktivační události se vytvářejí pomocí příkazu *CREATE TRIGGER*. Pomocí parametrů se specifikuje typ události a následně požadovaná akce. Ta je definována jako blok kódu PL/SQL (viz kapitola 3.4.4) – začíná klíčovým slůvkem *BEGIN* a končí slovem *END*. V tomto bloku je možno používat všechny možnosti, které nabízí procedurální jazyk PL/SQL, tedy větvení (*IF*), smyčky (*FOR, WHILE*), proměnné atd. Stejně tak je možno v tomto bloku volat jiné uložené PL/SQL procedury a funkce.

Ve verzi 11g Oracle umožnil vytvářet i tzv. složené aktivační události (*compound trigger*). V těch je možno vytvořit blok kódu pro každý okamžik z následujících časování: *BEFORE STATEMENT* (před příkazem), *BEFORE EACH ROW* (před každým řádkem), *AFTER EACH ROW* (za každým řádkem) a *AFTER STATEMENT* (po příkazu). Takto lze vytvářet složité příkazy, které sdílí data na úrovni řádku a příkazy a lze se tak vyhnout problém mutujících tabulek – tedy například situace, kdy aktivační událost *INSERT* provádí výpočet průměrné hodnoty a vkládané řádky současně tuto hodnotu mění [12].

Příkaz *CREATE TRIGGER* se používá i v SŘBD PostgreSQL. Stejně jako v systému Oracle lze vytvořit aktivační události typu *BEFORE, AFTER, INSTEAD OF* a *FOR EACH ROW*. Tyto aktivační události však lze vytvořit pouze na úrovni tabulky nebo pohledu, nelze je vytvářet na úrovni databáze. Stejně tak PostgreSQL nenabízí žádnou obdobu složených aktivačních událostí z databáze Oracle.

Při spuštění aktivační události lze pak zavolat uživatelsky definovanou funkci, napsanou například v jazyce PL/pgSQL, což je obdoba procedurálního jazyky PL/SQL databáze Oracle.

V tomto srovnání vítězí SŘBD Oracle, jelikož nabízí mnohem širší škálu možností práce s aktivačními událostmi – především ty na úrovni databáze lze považovat za velkou výhodu.

4.3.3 Audit

Audit je monitorování a nahrávání vybraných uživatelských akcí. Může být založen na jednoduchých akcích (typ SQL příkazu) nebo na kombinaci faktorů, které mohou zahrnovat například uživatelské jméno, čas, databázový objekt apod. Audit v žádném případě nenahrazuje běžné bezpečnostní politiky (autorizace, autentizace, oprávnění), měl by sloužit spíše jako doplněk či kontrola toho, že bezpečnostní politiky fungují správně.

SŘBD Oracle nabízí několik základních typů auditu [29]:

- Statement Auditing – například *AUDIT TABLE*. Tento příkaz bude zaznamenávat všechny příkazy *CREATE TABLE, DROP TABLE* a *SELECT FROM table*.
- Privilege auditing – například *AUDIT CREATE TABLE*. Na rozdíl od předchozího bodu je tento typ auditu zaměřen čistě na oprávnění (*CREATE TABLE*).
- Schema Object Auditing – například *AUDIT SELECT ON table*. Tento typ auditu bude zaznamenávat všechny operace typu *SELECT*, které proběhnou na tabulce *table*.
- Fine-Grained Auditing – monitoruje data podle obsahu, například *value > 1000*. Tento typ auditu bude zaznamenávat všechny řádky, které při nějaké DML operaci nabydou hodnoty větší než 1000. Tento typ auditu existuje zejména proto, že auditovat veškeré *SELECT* operace může být velmi náročné na výkon, a tak je lepší monitorovat pouze takové, které jsou klíčové z hlediska bezpečnosti databáze.

Všechny takto auditované operace se pak uchovávají buď v datovém slovníku dané databáze (*DBA_AUDIT_TRAIL* – tabulka *sys.aud$*), anebo v souborech operačního systému. Umístění těchto souborů se liší podle platformy, například na platformě Solaris je toto

umístění *$ORACLE_HOME/rdbms/audit*. Lze auditovat jak úspěšné, tak neúspěšné SQL příkazy.

SŘBD PostgreSQL takovouto funkcionalitu nenabízí. Lze namítnout, že obdobné funkcionality lze dosáhnout vytvořením odpovídajících aktivačních událostí (*Triggers*), které budou potřebné údaje zaznamenávat do příslušné tabulky. To ovšem znamená nutnost vytvořit další pomocné datové struktury tabulek kromě originálních a ručně vytvořit všechny požadované aktivační události, což zabere obrovské množství času. V případě změny struktury databáze pak bude potřeba změnit i související aktivační události. Oproti tomu v SŘBD Oracle je audit jakýchkoliv operací na jakémkoliv objektu nesrovnatelně jednodušší záležitostí.

4.3.4 Možnosti zálohování

Data v SŘBD Oracle je možno zálohovat následujícími způsoby [24]:

- Export/import
- „Studené“ Offline zálohy
- „Horké“ Online zálohy
- RMAN zálohy

Zálohovat pomocí exportu znamená v podstatě vytvořit soubor obsahující logickou strukturu databáze a samotná data. Tento soubor je pak možné nahrát zpět do opravené databáze. Od verze 10g je k dispozici novější nástroj Data Pump, který přináší lepší výkon a umožňuje exportovat soubory přes síť. Dnes již staré Export/Import nástroje jsou stále podporovány kvůli zpětné kompatibilitě.

Nástroj Data Pump provede dotazy nad databází (včetně datového slovníku) a jejich výstup zapíše do souboru ve formátu XML. Tento soubor je pak možno číst nástrojem Data Pump Import, který provede všechny příkazy uložené v tomto souboru. Tento způsob zálohy je však použitelný pouze pro menší databáze (do 50GB), poté už je vhodnější použít jiný nástroj.

Zálohovat databázi offline znamená vypnout databázi a udělat fyzickou zálohu všech datových souborů, logů a kontrolních souborů. Výhodou je, že při tomto způsobu zálohy nejsou zapotřebí údaje z archivních logů a ani databáze nemusí běžet v režimu *ARCHIVELOG*. Nevýhodou však je, že je nutné přerušit běh databáze, což může být problémem.

Pokud databáze běží v režimu *ARCHIVELOG*, provádí se archivace všech souborů protokolu a v rámci databáze se provádí záznam transakcí. Do online souborů protokolu je zapisováno cyklickým způsobem – když dojde k naplnění posledního souboru protokolu, začne se zapisovat opět do prvního. Při tomto typu zálohy je nutné přepnout tabulkové prostory do režimu zálohování. Výhodou tohoto způsobu zálohování je, že do určitého časového bodu v minulosti je možno vždy obnovit databázi bez nutnosti přerušení jejího běhu.

Poslední možností je pak využít k záloze nástroj RMAN – Recovery Manager. Tento nástroj je standardní součástí instalace SŘBD Oracle, který umožňuje zautomatizovat proces zálohování. Pomocí nástroje RMAN lze vytvořit kompletní či inkrementální zálohu, zálohovat tabulkový prostor, datové soubory nebo řídící soubor a soubor *SPFILE*. Kromě toho také RMAN nabízí možnost vytvořit tzv. kopii obrazu [30], což je bitově přesná kopie daného tabulkového prostoru nebo celé databáze. RMAN také obsahuje nástroje pro kompresi záloh pro úsporu místa na disku a podporu pro zálohování dat na pásková média. Samozřejmostí je pak možnost obnovit databázi, řídící soubor, tabulkový prostor či datový soubor pomocí tohoto nástroje.

V SŘBD PostgreSQL lze rozlišit tři základní přístupy při zálohování dat [21]:

- SQL dump
- Záloha souborů na úrovni systému
- Kombinace zálohy fyzických dat a WAL logů

Zálohování pomocí *SQL dump* je analogický způsob zálohování pomocí Exportu/Importu (či *Data pump*) u SŘBD Oracle. Nástroj *pg_dump* vytvoří textový soubor s SQL příkazy, který při spuštění na severu vytvoří databázi ve stejné podobě, v jaké byla při vytvoření tohoto souboru. Tento nástroj však nemá žádná zvláštní oprávnění – ke všem objektům, jež jsou předmětem zálohování, musí mít příslušná oprávnění. Proto je nejčastěji spouštěn administrátorem databáze (*superuser*). Nástroj *pg_dump* zapisuje na standardní výstup, tudíž je možné například v linuxovém prostředí využít externích nástrojů pro kompresi nebo rozdělit výsledný soubor na několik menších příkazem *split*.

Další možností zálohování dat je vytvoření fyzické kopie datových souborů databáze. Nevýhodou tohoto postupu je, že databáze musí být během kopírování souborů vypnuta. Zároveň nelze zálohovat pouze vybrané části databáze (například specifickou tabulku), protože informace v ní obsažené budou nepoužitelné bez změn potvrzovacích logů

(*pg_clog*), které obsahují potvrzený stav všech transakcí. Tímto způsobem tedy lze zálohovat a obnovovat pouze celý databázový cluster. Další nevýhodou může být to, že takto vzniklé zálohy budou mít mnohem větší velikost, než soubor vytvořený metodou SQL dump. Výhodou však je, že fyzická záloha datových souborů je zpravidla provedena rychleji [21].

Posledním způsobem zálohy databáze PostgreSQL je kombinace fyzické zálohy datových souborů a informací obsažených ve WAL lozích (*Write Ahead Log*). WAL logy zaznamenávají změny provedené v datových souborech. Pokud tedy máme zálohu fyzických souborů, je možné ji použít a následně pomocí WAL logů přehrát všechny změny provedené v databázi až do současného stavu.

Tento přístup má několik výhod. Není nutné mít vždy perfektně konzistentní zálohu datových souborů – WAL logy ji napraví (což ostatně dělají i v případě pádu databáze). Další výhodou je, že k úspěšnému obnovení stačí mít pouze jednu jedinou fyzickou zálohu datových souborů – o vše ostatní se postarají WAL logy (za předpokladu, že jsou neustále archivované). Díky tomu odpadá nutnost uchovávat velký počet záloh fyzických souborů, což může vést k obrovské úspoře místa zejména u velkých databází. Díky WAL logům je také možné obnovit databázi do jakéhokoliv bodě v čase – není vždy nutné přehrávat je až do konce (současnosti). WAL logy je také možno předávat jiné databázi obsahující stejné fyzické soubory – tím lze vytvářet kopii databáze, kterou je možno kdykoliv spustit v případě havárie originální databáze. Nevýhodou je, že tímto způsobem nelze zálohovat část databáze – vždy je nutno zálohovat celý databázový cluster, což může být náročné na úložný prostor.

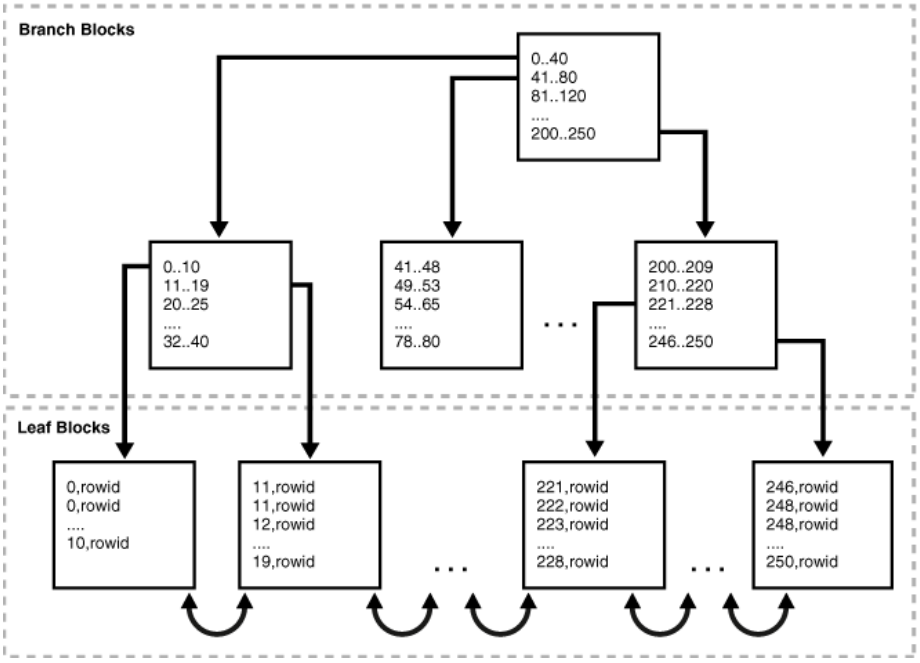
Celkově lze konstatovat, že SŘBD PostgreSQL nabízí spíše základní možnosti zálohování. Je velmi těžké zálohovat data za běhu systému – jednou z možností je například provozovat dvě totožné databáze pomocí WAL logů a přepínat je v případě pádu jedné z nich. Toto řešení však nemusí být v praxi vždy přijatelné, jelikož je náročné na výkon a úložný prostor. Oproti tomu SŘBD Oracle umožňuje provádět online zálohy dat poměrně jednoduchým způsobem. Zároveň nabízí kromě klasických metod i své vlastní specifické možnosti, jak předejít ztrátě dat.

4.3.5 Indexy

V SŘBD Oracle existují dva základní typy indexů – B-tree index a bitmapový index. Ostatní indexy (jedinečné indexy, clusterové indexy, bitmapové join indexy, indexy

s využitím funkcí, indexy s reverzním klíčem a textové indexy) jsou pouze variacemi na tyto dva typy indexů. Za běžný či základní je považován index typu B-tree.

B-tree index je datová struktura obsahující hodnotu indexovaných sloupců (na kterých je index vytvořen) a RowID (unikátní číslo řádku) této hodnoty, samozřejmě v seřazené podobě. Název indexu je odvozen od „stromové“ struktury. Řádky tabulky jsou uloženy v blocích. Oracle vždy musí přečíst celý blok včetně řádků, které jsou irelevantní. Proto je o úroveň výše uložen seznam bloků s první indexovanou hodnotou tohoto bloku a jeho adresou. Tuto úroveň si lze představit jako větev, která obsahuje listy (nejmenší bloky). O úroveň výše může být další „větev“ obsahující seznam menších větví. Tímto způsobem se dostaneme až ke kmeni stromu, tedy jakémusi rejstříku či nejobecnějšímu přehledu – opět ve formě jednoho bloku.



Obrázek 6: Příklad struktury B-tree indexu [29]

Indexy mají v SŘBD Oracle tři základní funkce:

- Rychle naleznout specifické řádky bez použití *Full Table Scanu* (projití celé tabulky).
- Vyhnout se obecně přístupu do tabulky – dotaz vracející pouze indexovaný sloupec nemusí vůbec přistupovat do indexované tabulky. V SŘBD Oracle lze vytvářet i tzv. *Index-organized tables*, což jsou tabulky obsahující řádky seřazené podle hodnot indexu (na rozdíl od klasických tabulek obsahujících neseřazené hodnoty).

- Vyhnout se řazení výsledků. Pokud nějaká operace (*JOIN, GROUP BY*) vyžaduje seřazené řádky, může použít hodnoty z indexu namísto řazení celé tabulky.

Jedinečný index (*Unique index*) je nejobvyklejší formou indexu, který využívá B-tree. Je obvykle používán pro implementaci primárního klíče v tabulce. Zajistí, že tabulka nebude v indexovaném sloupci obsahovat duplicitní hodnoty.

Duplicitní index je obdoba jedinečného až na to, že umožňuje vkládat duplicitní hodnoty do řádků. Tento typ indexu vytváří SŘBD Oracle implicitně příkazem *CREATE INDEX* v případě, kdy nejsou uživatelem zadány žádné další parametry.

Index s reverzním klíčem je typ indexu, který se využívá především v prostředí OLAP (*Online Transaction Processing*). Bajty hodnoty klíče tohoto indexu jsou uloženy v obráceném pořadí.

Indexy s využitím funkcí jsou taktéž typu B-tree. Jako hodnotu ukládají výstup nějaké funkce provedené na indexovaném sloupci. Příkladem může být funkce *upper(last_name),* která by při vhodné transformaci dotazů vracela správně výsledky pro dotaz:

```
SELECT last_name FROM employees WHERE last_name = 'SalouNovA';
```

Pokud je totiž příjmení v klauzuli *WHERE* ve výše uvedeném příkladu specifikováno s velkým písmenem uprostřed, běžný index by nevrátil žádnou hodnotu.

Bitmapové indexy se používají na sloupcích, které se často vyskytují v omezovací klauzuli *WHERE* a nabývají omezeného počtu různých hodnot. Bitmapový index mapuje rozdílné hodnoty sloupce oproti ostatním hodnotám. Největší přínos tedy mají i sloupců typu pohlaví (muž/žena), hodnocení (1,2,3,4,5), Ano/Ne apod. Zároveň by měly být vytvořeny na sloupcích, do nichž se nepřidávají nové hodnoty. Každá nová hodnota totiž bude vyžadovat vytvoření nové bitmapy.

V SŘBD Oracle lze vytvářet indexy na tabulkách a clusterech. Lze vytvářet indexy pro oddíly v rozdělených tabulkách (*Partitioned indexes*) a také doménové indexy (*Domain indexes*). Od verze 11g lze vytvářet neviditelné indexy – indexy, jež jsou implicitně vypnuty, pokud není uživatelem přímo řečeno, že se mají použít. Lze tak testovat výkon s indexem a bez indexu bez nutnosti index mazat klauzulí *DROP INDEX*.

V SŘBD PostgreSQL jsou k dispozici indexy typu B-tree, Hash, GiST, SP-GiST a GIN. Indexy typu B-tree mají stejnou filozofii jako v SŘBD Oracle a taktéž jsou základním (defaultním) typem indexu vytvářeným klauzulí *CREATE INDEX*.

Index typu Hash zvládne pouze srovnání pomocí jednoduché rovnosti (operátor „=“). Tento typ indexu se však v současnosti nedoporučuje používat, jelikož při pádu databáze je nutné jej znovu ručně nakonfigurovat [21].

Indexy typu GiST (*Generalized Search Tree*) nejsou jednotným typem indexu, nýbrž spíše indexovou infrastrukturou, která může obsahovat různé strategie indexování. Jedná se o jakési zobecnění B-tree indexu, GiST například nerozlišuje datové typy na sloupci.

SP-GiST indexy nabízejí podobně jako GiST indexy infrastrukturu podporující různé typy vyhledávání. Mezi tyto datové struktury patří například quadtrees, k-d trees či radix trees.

Indexy typu GIN jsou převrácené indexy, které zvládnou pracovat s hodnotami, které obsahují více klíčů, jako například pole.

Jednotlivé indexy se v SŘBD PostgreSQL liší především v algoritmu, který používají pro vyhledávání. Zároveň se každý z těchto typů indexů hodí pro jiný typ operátorů. Stejně jako v SŘBD Oracle lze i v SŘBD PostgreSQL vytvořit *UNIQUE* index. Musí však být typu B-tree. Lze také vytvářet indexy pro nějakou podmnožinu tabulky – tzv. *Partial index*.

Obecně lze říci, že SŘBD PostgreSQL nabízí o něco širší možnosti indexování než SŘBD Oracle – kromě klasického indexu typu B-tree nabízí i několik vlastních datových struktur. Z hlediska funkcionality však SŘBD Oracle zas tolik nezaostává, stejně jako PostgreSQL nabízí různé typy indexů, které mohou pomoci vylepšit rychlost zpracování většiny dotazů.

4.3.6 Zhodnocení sekce Základní funkcionalita databází

Váhy jednotlivých kritérií jsou vidět v následující tabulce. Za nejdůležitější jsou považována kritéria K1 (správa uživatelů) a K4 (možnosti zálohování), která jsou si rovnocenná svou významností. Tato dvě kritéria mají největší vliv na základní funkci databáze – ukládání dat a řízení přístupu k nim. Jako nejméně podstatné kritérium v párovém porovnání vyšel audit (K3), což je však nutno brát s rezervou, jelikož se svou váhou tolik neliší od zbývajících dvou kritérií – K2 (Aktivační události) a K5 (Indexy).

	K1	K2	K3	K4	K5	G	V	Pořadí
K1	1	3	3	1	3	1,933182045	0,319672	2.
K2	1/3	1	3	1/3	1/3	0,644394015	0,106557	4.
K3	1/3	1/3	1	1/5	1/2	0,406585136	0,067233	5.
K4	1	3	5	1	3	2,141127368	0,354058	1.
K5	1/3	3	2	1/3	1	0,922107911	0,15248	3.
					suma:	6,047396476		

K1 – Správa uživatelů
K2 – Aktivační události (Triggers)
K3 – Možnosti auditu
K4 – Možnosti zálohování
K5 - Indexy
G – Geometrický průměr
V – Vážený geometrický průměr (váha kritéria)

Tabulka 9: Výpočet vah kritérií sekce Základní funkcionalita databází [vlastní tvorba]

U prvního kritéria (správa uživatelů) nabízejí obě databáze podobné možnosti, čemuž odpovídají i výsledné váhy:

	Oracle	PostgreSQL	G	V
Oracle	1	1	1	0,5
PostgreSQL	1	1	1	0,5
		suma:	2	

Tabulka 10: Kritérium 1 – hodnocení správy uživatelů [vlastní tvorba]

Druhé kritérium představují aktivační události (*Triggers*). SŘBD Oracle zde nabízí mnohem širší možnosti při práci s nimi, čemuž odpovídá i hodnocení, kde tento SŘBD silně převyšuje SŘBD PostgreSQL:

	Oracle	PostgreSQL	G	V
Oracle	1	5	2,236068	0,833333
PostgreSQL	1/5	1	0,447214	0,166667
		suma:	2,683282	

Tabulka 11: Kritérium 2 – hodnocení aktivačních událostí [vlastní tvorba]

Další kritériem (K3) jsou možnosti auditu, které oba systémy nabízejí. V této oblasti dominuje SŘBD Oracle.

	Oracle	PostgreSQL	G	V
Oracle	1	7	2,645751	0,875
PostgreSQL	1/7	1	0,377964	0,125
		suma:	3,023716	

Tabulka 12: Kritérium 3 – hodnocení auditu [vlastní tvorba]

Důležitým kritériem bylo kritérium číslo čtyři – možnosti zálohování. Zde opět vyšel lépe SŘBD Oracle, ovšem ne tak zásadním rozdílem, jako v případě předchozího kritéria:

	Oracle	PostgreSQL	G	V
Oracle	1	4	2	0,661438
PostgreSQL	1/4	1	0,5	0,165359
		suma:	2,5	

Tabulka 13: Kritérium 4 – možnosti zálohování [vlastní tvorba]

Posledním kritériem v této sekci pak byly možnosti indexování. Zde vychází lépe SŘBD PostgreSQL především kvůli širší nabídce Indexů a možností práce s nimi.

	Oracle	PostgreSQL	G	V
Oracle	1	1/4	0,5	0,165359
PostgreSQL	4	1	2	0,661438
		suma:	2,5	

Tabulka 14: Kritérium 5 – indexy [vlastní tvorba]

Tabulka s výsledným vyhodnocením této sekce pak vypadá následovně:

Kritérium	Váha	Oracle	PostgreSQL
K1	0,319672	0,5	0,5
K2	0,106557	0,833333	0,166667
K3	0,067233	0,875	0,125
K4	0,354058	0,661438	0,165359
K5	0,15248	0,165359	0,661438
	Celkem:	0,566864	0,345403

```
K1 – Správa uživatelů
K2 – Aktivační události (Triggers)
K3 – Možnosti auditu
K4 – Možnosti zálohování
K5 - Indexy
```

Tabulka 15: Vyhodnocení sekce Základní funkcionalita databází [vlastní tvorba]

Z tabulky vyplývá, že v této sekci byl vítězem SŘBD Oracle.

4.4 Rozšířená funkcionalita

4.4.1 Dědičnost (Inheritance)

SŘBD PostgreSQL umožňuje vytvářet tabulky, které dědí atributy (sloupce) od jiné. Tento nástroj usnadňuje tvorbu datových modelů. Například chceme-li uchovávat informace o městech – každý stát obsahuje mnoho měst ale pouze jedno hlavní. Šlo by pochopitelně vytvořit sloupec s názvem *hlavniMesto* a v něm mít buď název státu anebo hodnotu *NULL*. V tom případě by ale naprostá většina měst měla v podstatě zbytečný atribut. SŘBD PostgreSQL umožňuje podědit atributy z jedné tabulky do druhé a tím se vyhnout tomuto problému:

```
CREATE TABLE mesta (
   nazev       text,
   pocetobyvatel     float,
);

CREATE TABLE hlavnimesta (
   stat        char(2)
) INHERITS (mesta);
```

Tabulka *hlavnimesta* převezme všechny atributy rodičovské tabulky (*mesta*). Klauzule *SELECT* pak bude vracet řádky z obou tabulek, pokud však nepřidáme klauzuli *ONLY* – pak lze specifikovat, z které podtabulky chceme vrátit řádky. Potomci dědí veškerá omezení typu *CHECK* aplikované na rodičovskou tabulku. Dědičnost se dá nastavit (a zrušit) také klauzulí *ALTER TABLE*.

Problém však představuje klauzule například *INSERT* – ta vkládá data vždy pouze do jasně specifikované tabulky. Neumí přesměrovat informace určené potomkovi do příslušné tabulky. Například následující příkaz by neproběhl:

```
INSERT INTO mesta (nazev, pocetobyvatel, stat) VALUES ('Praha',
1200000, 'CZ');
```

Obecně mají problém s dědičností klauzule, které jsou určeny pro jednu specifickou fyzickou tabulku – například příkaz pro údržbu databáze *VACUUM*. Dalším velkým problémem je to, že primární a cizí klíče se nedědí. Pokud by tedy byl v příkladu uvedeném výše sloupec *nazev* specifikován jako *PRIMARY KEY*, do tabulky *hlavnimesta* by šlo vkládat řádky s duplicitními názvy. Samozřejmě lze vytvořit *PRIMARY KEY* i v podtabulce,

to ovšem neřeší problém – řádky sice nebudou duplicitní v rámci tabulky *hlavnimesta*, ale mohou být duplicitní v rámci rodiče a potomka. Stejně tak cizí klíč je nutno specifikovat nejen v rodičovské tabulce, ale i v potomkovi.

SŘBD Oracle obdobnou funkcionalitu nenabízí. Jediný způsob, jakým by se dalo přiblížit požadované funkcionalitě, je použít tzv. objektově-relační přístup. V SŘBD Oracle je možno vytvářet tzv. typy. Pomocí klauzule *CREAE TYPE* je tak možnost seskupit několik příbuzných sloupců (například řádky adresy) do jednoho datového typu. Klasické tabulky pak mohou obsahovat klasické sloupce anebo takto vytvořené typy. K jednotlivým podtypům se pak přistupuje pomocí tečkové konvence [12], například:

```
SELECT mesta.nazev, mesta.hlavnimesta.stat FROM mesta;
```

Takto lze však vytvářet pouze nadtypy a podtypy, nejde přímo o dědičnost jako takovou. Pokud vývojáři PostgreSQL odstraní výše uvedené problémy s dědičností (například nepřebírání klíčů), mohlo by jít o velmi atraktivní doplněk k databázovým funkcím. Takto je SŘBD PostgreSQL obrazně řečeno na půli cesty k požadované funkcionalitě. To ovšem znamená mnohem dál, než je SŘBD Oracle, jelikož funkcionalitu typů z SŘBD Oracle PostgreSQL nabízí taktéž – ve formě složených typů (*Composite types*).

4.4.2 Rozdělování tabulek (Partitioning)

SŘBD Oracle ve verzi 11g přinesl možnost tzv. *Partitioningu* – rozdělování tabulky do oddílů. Toto řešení má několik výhod [12]:

- Může dojít ke zlepšení výkonu, jelikož ke zpracování některých dotazů bude muset SŘBD Oracle projít jen některý oddíl namísto celé tabulky.
- Oddíly mohou usnadnit organizaci tabulek. Načítání a odstraňování dat z oddílů bude pravděpodobně jednodušší, než obdobná operace na celé tabulce.
- Rozdělení do oddílů otevírá i nové možnosti při provádění záloh.

Rozdělení tabulky se realizuje v příkazu *CREATE TABLE* přidáním klauzule *PARTITION BY RANGE*. Parametr *RANGE* definuje jeden možný způsob rozdělení tabulky – podle rozsahu. Dle vybraného sloupce lze například všechny řádky začínající na písmena „a – m“ umístit do jednoho oddílu, a zbývající do druhého. Dalšími způsoby rozdělení jsou *BY LIST* (dle přesně uvedeného výčtu hodnot), *BY HASH* (na klíči oddílu se provede hashovací funkce) a *BY REFERENCE* (tabulka se rozdělí podle hodnot umístěných v jiné tabulce spojené přes cizí klíč).

V jednotlivých oddílech jde vytvářet i tzv. pododdíly (*subpartition*). V těch lze kombinovat všechny výše uvedené způsoby rozdělování. Pododdíly se vytvářejí přidáním klauzule *SUBPARTITION BY ...* za klauzuli *PARTITION BY*. Správa oddílů a pododdílů se pak provádí skrze příkaz *ALTER TABLE*. Oddíly je možno upravovat, mazat, zaměňovat či ořezávat.

V SŘBD PostgreSQL je rozdělování tabulek realizováno pomocí dědění (viz kapitola 4.4.1). Je tedy potřeba nejprve vytvořit rodičovskou tabulku a následně vytvořit požadovaný počet potomků, kteří budou z rodičovské tabulky přebírat atributy. Na těchto potomcích je pak potřeba implementovat omezení typu *CHECK* dle požadovaných hodnot oddílů. Z toho plyne, že databáze PostgreSQL umožňuje oddíly dělit pouze podle rozsahu (*RANGE*) a výčtu (*LIST*). Jiné způsoby rozdělení nejsou podporovány [21].

Rozdělení řádků do odpovídajících oddílů je pak potřeba implementovat pomocí aktivační události (*trigger*) typu *INSTEAD OF* (viz kapitola 4.3.2) anebo pomocí pravidla (*rule* -viz kapitola 4.4.4). Oddíly jsou pak spravovány stejně jako v databázi Oracle pomocí příkazu *ALTER TABLE* anebo pomocí vypínaní/zapínání dědění (*NO INHERIT*). Dobrým pomocníkem je pak v SŘBD PostgreSQL klauzule *SET constraint_exclusion = on*; Pokud je nastavená hodnota *OFF*, bude při vykonávání dotazu SELECT prohledán každý oddíl tabulky. Pokud je však nastavena hodnota *ON*, budou nejprve posouzeny existující omezení (*constraints*) na rozdělené tabulce za cílem prokázání, že některé oddíly nemá smysl procházet. Pokud tomu tak bude, plánovač dotazu vyškrtne oddíl z prováděcího plánu, čímž dojde ke zlepšení výkonu, respektive snížení času potřebného k jeho vykonání.

Z uvedených řešení vyplývá, že SŘBD Oracle v případě rozdělování tabulek nabízí lepší možnosti – zejména nativní podporu a širší škálu nastavení. V SŘBD PostgreSQL je sice rozdělování tabulek taktéž možné, nicméně je realizováno pomocí jiné funkce databáze (dědění), což nelze nazvat přímou podporou. Zároveň je celý proces vytvoření a správy oddílů mnohem pracnější záležitostí kvůli nutnosti vytváření spouštěčů či pravidel, než v databázi Oracle.

4.4.3 Technologie Flashback Query

Databáze v rámci konzistence dat zobrazuje pouze data, která byla v databázi potvrzena. SŘBD Oracle však nabízí technologii Flashback Query, která umožňuje dotazovat se na data v takové podobě, v jaké se nacházela v minulosti.

Při klasické obnově dat je potřeba nejprve obnovit celou databázi ze zálohy a poté aplikovat všechny změny provedené v databázi až po nějaký konečný bod. Jde o časově velice náročnou operaci, při které jsou použity údaje z archivního logu, redo logů a UNDO segmentů [6]. Oproti tomu technologie Flashback Query využívá své vlastní flashback logy, undo segmenty a odpadkový koš. Lze jí přirovnat k tlačítku „zpět“ v textovém editoru.

Flashback lze provést v několika různých oblastech. Pomocí příkazu *FLASHBACK TABLE* lze obnovit dřívější stav tabulky. Takto lze obnovit i smazanou tabulku.

V SŘBD Oracle tabulka při smazání nezmizí celá – její bloky se stále nacházejí v tabulkovém prostoru. Zároveň ji lze zobrazit dotazem do datového slovníku *RECYCLEBIN* (odpadkový koš). Teprve když je smazaná i z odpadkového koše (příkaz *PURGE*), dojde k jejímu smazání i z tabulkového prostoru. Poté je již tabulka nenávratně ztracena (nepočítáme-li zálohy). Do té doby je však možná její rychlá obnova.

Stejně jako v případě tabulky lze použít operaci Flashback i na celou databázi. Tento příkaz je ovšem mnohem náročnější na informace uložené v archivním logu a v undo segmentu. Pokud zde není uložen dostatek informací pro operaci Flashback, příkaz selže.

Flashback probíhá nejčastěji na základě SCN (*System change number*), časového údaje, nebo transakce. Zde je uveden příklad použití:

```
FLASHBACK TABLE nazevtabulky TO BEFORE DROP;
SELECT sloupec FROM nazevtabulky AS OF TIMESTAMP TO_TIMESTAMP(
‘2015-05-05 09:50:00’, ‘YYY-MM-DD HH:MI:SS’);
```

Pro používání této technologie je potřeba mít příslušná oprávnění (např. *FLASHBACK ANY TABLE*) a databáze musí používat funkci automatické správy obnovení – AUM (*Automatic Undo Management*) [12]. Celkově je tato technologie velice přínosná – umožňuje mnohem rychlejší nápravu chyb vzniklých například neuváženým potvrzením transakce. Bohužel je momentálně dostupná pouze v SŘBD Oracle – systém PostgreSQL 9.3 žádnou obdobnou funkcionalitu nenabízí.

4.4.4 Pravidla (Rules)

V obou srovnávaných systémech jsou k dispozici tzv. pravidla (*rules*), ovšem v každém SŘBD mají odlišnou funkcionalitu.

V SŘBD PostgreSQL lze vytvořit pravidlo pomocí klauzule *CREATE RULE*. To může záviset buď na tabulce, nebo na pohledu. Pomocí pravidel lze definovat alternativní akci, která se má provést při operacích typu *SELECT*, *INSERT*, *UPDATE* a *DELETE* na daném

objektu. Pravidlo typu *SELECT* může obsahovat pouze jednu akci (typu *INSTEAD*) a nesmí být podmíněné (na rozdíl od ostatních). Zároveň se tato akce provede až po příkazu, na rozdíl od ostatních typů pravidel, u nichž se akce vždy provede ještě před samotným spouštěcím příkazem. Pravidlo může vypadat například takto:

```
CREATE TABLE mujpohled;
CREATE RULE "nazev" AS ON SELECT TO mujpohled DO INSTEAD
   SELECT * FROM nazevtabulky;
```

Pomocí pravidel jsou implementovány veškeré pohledy v SŘBD PostgreSQL [21]. Pokud bude mít tabulka *mujpohled* stejné sloupce jako tabulka *nazevtabulky* z případu výše, jsou tyto dva příkazy ekvivalencí příkazu:

```
CREATE VIEW mujpohled AS SELECT * FROM nazevtabulky;
```

Pravidla jsou v SŘBD PostgreSQL hodně podobná aktivačním událostem (*trigger*). Hlavní rozdíl je, že pravidlo proběhne pouze jednou – většinou nějak upravuje zadaný dotaz anebo generuje jiný. Naopak aktivační událost proběhne jednou pro každý řádek. Pokud tedy manipulujeme s velkým počtem řádků, je lepší použít pravidlo, jelikož provedení jednoho příkazu navíc bude pravděpodobně rychlejší, než opakované provedení příkazu pro každý řádek zvlášť.

Jak již bylo řečeno, pomocí pravidel lze vytvářet pohledy. Pokud však pomocí pravidel ručně vytvoříme pohled, je potřeba také zajistit, aby se operace typu *INSERT* či *UPDATE* promítly i do originálních tabulek. To lze ošetřit například pomocí aktivační události, ovšem u pohledů vzniklých vícenásobným spojením půjde o velmi pracnou záležitost.

V SŘBD Oracle se pravidla vyskytují také, ovšem je to spíše v souvislosti s technologií Oracle Streams, která zajišťuje tok dat a událostí v rámci jedné databáze nebo z jedné databáze do druhé [30]. Pravidla se v databázi Oracle drží jednoduché struktury ECA (*Event – Condition – Action*). Pokud například dojde v jedné databázi k nějaké DML události, technologie Oracle Streams umožňuje replikovat tyto změny v databázi jiné.

Pravidla lze vytvářet pomocí balíku *DBMS_RULE_ADM*, konkrétně pak procedurou *CREATE_RULE*. Akce se definují pomocí jazyka PL/SQL. Existují buď samostatně, nebo v rámci množiny pravidel. Na rozdíl od PostgreSQL nejsou přímo přiřazeny k nějakému objektu, ale k nějaké události. Z toho plyne i odlišná funkcionalita a využití. Pravidla mají v SŘBD Oracle mnohem větší využitelnost, už jen proto, že je lze využít k zachytávání procesů v databázi obecně, a nikoliv pouze na určitém objektu. Neslouží zde pouze jako ekvivalent aktivačních událostí. Po splnění podmínky pravidla lze pak nastavit

i akci, která se má provést. Pravidla jsou spravována pomocí nástroje Rules Manager, který umožňuje tvořit i komplexní pravidla. Pravidla mohou být i složená a lze je rozřazovat do množin (*Rule sets*) [31].

Obecně je problematika pravidel v SŘBD Oracle mnohem širší a komplikovanější, než v SŘBD PostgreSQL. Jelikož je nutné vždy přesně definovat událost, je tvorba pravidel v SŘBD Oracle mnohem složitější. Zároveň ale platí, že v SŘBD PostgreSQL mají pravidla mnohem užší funkcionalitu a možnosti využití, než v SŘBD Oracle.

4.4.5 Hierarchické dotazy

V SŘBD Oracle je možno pomocí tzv. hierarchických dotazů (*Hierarchical query*) vyjádřit hierarchickou strukturu záznamů. To se hodí například v případě, že je potřeba zobrazit strukturu vedení firmy – nadřízené, podřízené apod. V SŘBD Oracle toto lze realizovat kombinací klauzulí *CONNECT BY* (specifikuje vztah mezi nadřazenými a podřízenými záznamy) a *START WITH* (specifikuje kořen stromu). Dále je možno využít pseudosloupec *LEVEL*, který umí zobrazit úroveň záznamu ve struktuře. Výsledek dotazu pak může vypadat například takto:

```
EMPLOYEE_ID LAST_NAME                 MANAGER_ID      LEVEL
----------- ------------------------- ---------- ----------
        101 Kochhar                          100          1
        108 Greenberg                        101          2
        109 Faviet                           108          3
        110 Chen                             108          3
        111 Sciarra                          108          3
        112 Urman                            108          3
        113 Popp                             108          3
```

Tabulka 16: Výsledek hierarchického dotazu [29]

Pomocí hesla *PRIOR* lze nastavit, zda se budu struktura vytvářet směrem od kořene k „listům“ (podřízeným záznamům), anebo obráceně.

SŘBD PostgreSQL přímo hierarchické dotazy nativně nepodporuje, nicméně výsledku je možno docílit pomocí rekurzivních dotazů. Nejprve se vytvoří pomocná tabulka s výpočty pomocí klauzule *WITH*, která může díky klauzuli *RECURSIVE* záviset na svém vlastním výstupu [21]. Toto řešení sice není tak uživatelsky přátelské, jako v případě SŘBD Oracle, nicméně lze dosáhnout stejných výstupů.

4.4.6 Složitá seskupení

Složitá seskupení jsou jednou ze složitějších funkcionalit, které nabízí SŘBD Oracle. Patří sem funkce *ROLLUP, CUBE, GROUPING* a *GROUPING SETS*.

Pomocí funkce *ROLLUP* je možno kromě klasického seskupení (například tržby jednotlivých zákazníků za jednotlivé měsíce získané klauzulí *GROUP BY*) zobrazit i mezivýpočty a celkové výpočty – tedy například součet tržeb za každý měsíc a součet tržeb za všechna období (měsíce). Výsledek dotazu může vypadat například takto:

MONTH	LAST_NAME	SALES
JAN	Sokol	500
JAN	Dolezal	200
JAN		700
FEB	Votrhanek	400
FEB	Podrazka	100
FEB		500
		1200

Tabulka 17: Výsledek dotazu s klauzulí GROUP BY ROLLUP [vlastní tvorba]

Vstupní hodnoty (za klauzulí *GROUP BY*) lze libovolně kombinovat. Klauzule *CUBE* pak provede výpočty a mezivýpočty všech možných kombinací sloupců uvedených v ní. V tabulce uvedené výše by tedy byly vypsány i tržby zákazníků za všechny měsíce. Pokud nechceme vypisovat úplně všechny kombinace mezivýpočtů, je možno je limitovat pomocí klauzule *GROUPING SETS*, čímž lze velice redukovat počet vrácených řádků.

SŘBD PostgreSQL v současnosti tuto funkcionalitu vůbec nenabízí, byť její zprovoznění je dle dostupných zdrojů v plánu vývojářů [32]. Lze sice pomocí množinových operací (*UNION*) dosáhnout stejných výsledků, jako v případě klauzule *ROLLUP* v SŘBD Oracle, nicméně jde o velice náročné a složité dotazy, jejichž tvorba zabere nesrovnatelně více času, než prosté zadání jednoho klíčového slova.

4.4.7 Zhodnocení sekce Rozšířená funkcionalita databází

Nejdůležitějšími kritérii v této sekci jsou K2 (Rozdělování tabulek) a K6 (Složitá seskupení). Možnost rozdělit tabulky do oddílu může přinést velmi znatelný nárůst výkonu. Složitá seskupení jsou pak důležitým nástrojem pro operace používané v datových skladech. Po těchto kritériích mají největší váhu kritéria K4 (Pravidla) a K5 (Hierarchické dotazy).

Tato kritéria sice nejsou kruciální pro chod databáze, nicméně mají své využití v některých specifických případech. Nejmenší váhu pak mají kritéria K1 (Dědičnost) a K3 (Technologie Flashback Query). Tato dvě kritéria sice usnadňují práci s databází, nicméně lze se obejít i bez nich.

	K1	K2	K3	K4	K5	K6	G	V	Pořadí
K1	1	1/7	1	1/3	1/3	1/7	0,36246	0,043078	6.
K2	7	1	7	4	5	2	3,537605	0,420439	1.
K3	1	1/7	1	1/3	3	1/5	0,552911	0,065713	5.
K4	3	1/4	3	1	2	1/3	1,069913	0,127158	3.
K5	3	1/5	1/3	1/2	1	1/3	0,5673	0,067423	4.
K6	7	1/2	5	3	3	1	2,323879	0,27619	2.
						suma:	8,414068		

K1 – Dědičnost
K2 – Rozdělování tabulek (Partitioning)
K3 – Technologie Flashback Query
K4 – Pravidla
K5 – Hierarchické dotazy
K6 – Složitá seskupení
G – Geometrický průměr
V – Vážený geometrický průměr (váha kritéria)

Tabulka 18: Váha kritérií sekce Rozšířená funkcionalita databází [vlastní tvorba]

Prvním hodnoceným kritériem bylo zpracování dědičnosti. SŘBD PostgreSQL nabízí mnohem širší možnosti v této oblasti. Byť se funkcionalita nedá považovat za dokonalou, dalece přesahuje možnosti databáze Oracle.

	Oracle	PostgreSQL	G	V
Oracle	1	1/5	0,447214	0,166667
PostgreSQL	5	1	2,236068	0,833333
		suma:	2,683282	

Tabulka 19: Kritérium 1 – dědičnost [vlastní tvorba]

Druhé kritérium je nejdůležitějším v této sekci – jde o možnosti rozdělování tabulek (*Partitioning*). Zde nabízí SŘBD Oracle mnohem lepší řešení, než SŘBD PostgreSQL.

V SŘBD PostgreSQL není *partitioning* nativně podporován, existuje pouze možnost zrealizovat ho pomocí jiných dostupných funkcí.

	Oracle	PostgreSQL	G	V
Oracle	1	6	2,44949	0,857143
PostgreSQL	1/6	1	0,408248	0,142857
		suma:	2,857738	

Tabulka 20: Kritérium 2 – rozdělování tabulek [vlastní tvorba]

Třetím hodnoceným kritériem je technologie Flashback Query. Tuto technologii nabízí pouze SŘBD Oracle, systém PostgreSQL žádnou obdobnou funkcionalitu neposkytuje.

	Oracle	PostgreSQL	G	V
Oracle	1	9	3	0,9
PostgreSQL	1/9	1	0,333333	0,1
		suma:	3,333333	

Tabulka 21: Kritérium 3 – technologie Flashback Query [vlastní tvorba]

Dalším kritériem byla pravidla (*rule*). Zde bylo hodnocení poněkud obtížnější, jelikož v obou SŘBD mají pravidla jinou funkci a jiné využití. I tak lze ale říci, že toto využití je o něco širší v SŘBD Oracle.

	Oracle	PostgreSQL	G	V
Oracle	1	2	1,414214	0,424264
PostgreSQL	1/2	1	0,707107	0,212132
		suma:	2,12132	

Tabulka 22: Kritérium 4 – pravidla (Rules) [vlastní tvorba]

Pátým hodnoceným kritériem v této sekci byly hierarchické dotazy. Zde opět vede SŘBD Oracle, nicméně rozdíl není tak markantní - především kvůli tomu, že v SŘBD PostgreSQL lze dosáhnout stejných výsledků, byť poněkud komplikovanější cestou.

	Oracle	PostgreSQL	G	V
Oracle	1	2	1,414214	0,424264
PostgreSQL	1/2	1	0,707107	0,212132
		suma:	2,12132	

Tabulka 23: Kritérium 5 – hierarchické dotazy [vlastní tvorba]

Posledním kritériem v této sekci byla složitá seskupení. SŘBD Oracle zde opět poskytuje mnohem jednodušší řešení. Získání obdobných výsledků v SŘBD PostgreSQL je opět mnohem komplikovanější proces.

	Oracle	PostgreSQL	G	V
Oracle	1	3	1,732051	0,519615
PostgreSQL	1/3	1	0,57735	0,173205
		suma:	2,309401	

Tabulka 24: Kritérium 6 – složitá seskupení [vlastní tvorba]

Výsledné zhodnocení této sekce pak vypadalo následovně:

Kritérium	Váha	Oracle	PostgreSQL
K1	0,043078	0,166667	0,833333
K2	0,420439	0,857143	0,142857
K3	0,065713	0,9	0,1
K4	0,127158	0,424264	0,212132
K5	0,067423	0,424264	0,212132
K6	0,27619	0,519615	0,173205
	Celkem:	0,652763	0,191646

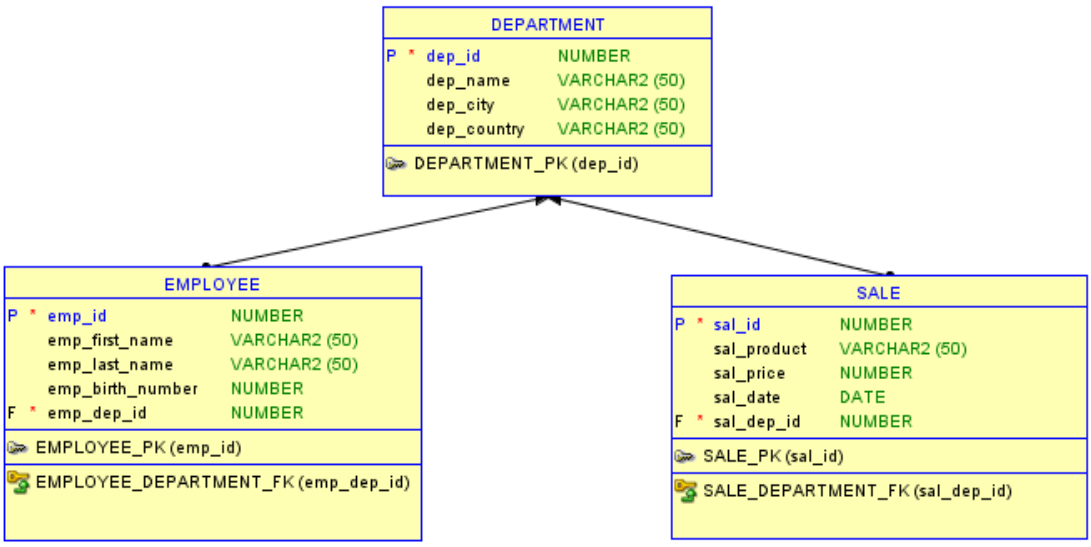
```
K1 – Dědičnost
K2 – Rozdělování tabulek (Partitioning)
K3 – Technologie Flashback Query
K4 – Pravidla
K5 – Hierarchické dotazy
K6 – Složitá seskupení
```

Tabulka 25: Vyhodnocení sekce Rozšířená funkcionalita databází [vlastní tvorba]

Z tabulky vyplývá, že v této sekci dosáhl lepších výsledků SŘBD Oracle.

4.5 Měření výkonu

Testování výkonu probíhalo na dvou oddělených testovacích prostředích (viz Kapitola 4.1.3) – každé pro jednu databázi. V každé databázi byla nainstalovaná totožná datová struktura – tři tabulky (*EMPLOYEE*, *DEPARTMENT*, *SALE*) obsahující primární klíč na sloupci *ID*, které byly spojené přes cizí klíče. Diagram tohoto datového modlu vypadal následovně:



Obrázek 7: Diagram datového modelu použitého k testování [vlastní tvorba]

Datový model vyjadřuje zjednodušenou strukturu firmy, která má pobočky (*DEPARTMENT*) v různých městech a zemích. V těchto pobočkách pracují zaměstnanci (*EMPLOYEE*), kteří prodávají produkty. Jednotlivé prodeje jsou evidovány v tabulce *SALE*. Pro tento datový model byl v obou databázích vytvořen samostatný tabulkový prostor. Jeho vlastníkem byl uživatel *NOVAK*, který byl taktéž použit pro spouštění SQL dotazů. Skript pro instalaci tohoto datového modelu je uveden v přílohách (8.1 a 8.2).

Pomocí skriptu byly taktéž tabulky naplněny – tabulka *EMPLOYEE* obsahovala 11 záznamů, tabulka *DEPARTMENT* 33 a tabulka *SALE* 582. Pro tvorbu hodnot primárních klíčů (sloupce *ID* v jednotlivých tabulkách) byly použity sekvence.

Samotné měření probíhalo vždy tak, že byl nejprve spuštěn operační systém a následně instance příslušné databáze. Následovalo přihlášení uživatele *NOVAK* přes konzoli a nastavení parametru *TIMING*, který zobrazuje čas potřebný k provedení dotazu. Na rozdíl od příkazu *EXPLAIN PLAN* zahrnuje i dobu potřebnou k zobrazení výsledků na terminálu, nikoliv pouze dobu potřebnou k provedení příkazu na serveru. V databázi tento parametr umožňuje zobrazovat čas v milisekundách s přesností na tři desetinná místa. V SŘBD Oracle lze však tímto parametrem zobrazit pouze desítky milisekund, tudíž výsledky z tohoto systému nejsou tak přesné. Problém šlo vyřešit testováním pomocí programu sqldeveloper, nicméně ten je poměrně náročný na systémové zdroje a mohl by ovlivnit výsledky měření. Kromě konzole nebyly spuštěny žádné jiné programy.

Následně byly do konzole zadány čtyři příkazy – každý z nich dvakrát. První příkaz běžel takzvaně „na surovo“ – po čerstvém startu operačního systému a databáze. Nebyla tedy k dispozici žádná data uložená ve vyrovnávacích pamětích operačního systému či databázové instance. Ostatní příkazy již měly k dispozici data (exekuční plány) dříve provedených příkazů (mezi příkazy nedocházelo k restartování systému). Cílem tohoto postupu bylo posoudit, jak se systém „přizpůsobí“ opakované práci s totožnými daty.

Jak je vidět dále, všechny příkazy měly totožnou syntaxi v obou databázích.

4.5.1 Sjednocení množin (UNION)

Prvním příkazem bylo klasické sjednocení dvou množin pomocí příkazu *UNION*. Použity byly sloupce *emp_dep_id* z tabulky *EMPLOYEE* a *dep_id* z tabulky *DEPARTMENT*. V tabulce *EMPLOYEE* byl limitující podmínkou rok narození (ilustračně odvozen z rodného čísla) a v tabulce *DEPARTMENT* byla limitující podmínkou země umístění. Příkaz vypadal následovně:

```
SELECT emp_dep_id FROM employee WHERE emp_birth_number BETWEEN
6000000000 AND 7000000000 UNION SELECT dep_id FROM department WHERE
dep_country = 'Czech republic' ORDER BY emp_dep_id;
```

Šlo o jednoduché spojení dvou množin ze dvou tabulek. Dotaz vracel 7 seřazených řádků z celkového možného počtu 33 krát 11 záznamů. Výsledky byly následující (v milisekundách):

	Oracle		PostgreSQL	
Příkaz	1. měření	2. měření	1. měření	2. měření
UNION přes dvě tabulky	40 ms	0 ms (zaokr.)	155 ms	0(0,465) ms

Tabulka 26: Výsledky spojení dvou množin [vlastní tvorba]

Jak z výsledků vyplývá, SŘBD Oracle tento dotaz napoprvé zvládl téměř 4x rychleji. Druhé měření již oba systémy zvládly za dobu pod jednu milisekundu (v případě SŘBD Oracle se kvůli zaokrouhlení zobrazila pouze nula).

4.5.2 Trojnásobné spojení (Three-way join)

Dalším příkazem bylo spojení všech tří tabulek přes hodnoty cizích klíčů. Jedinou limitující podmínkou bylo *ID* zaměstnance s hodnotou 2. Výsledek by se dal interpretovat

jako „všechny prodeje, které kdy uskutečnila pobočka zaměstnance s ID 2“. Příkaz vypadal následovně:

```
SELECT a.emp_first_name, a.emp_last_name, b.dep_name, b.dep_city,
c.sal_product, c.sal_price FROM employee a JOIN department b ON
a.emp_dep_id = b.dep_id JOIN sale c ON b.dep_id = c.sal_dep_id WHERE
a.emp_id = 2;
```

Dotaz vracel 96 řádků z celkového počtu všech řádků všech tabulek – 11 krát 33 krát 582. Výsledky měření vypadaly následovně:

	Oracle		PostgreSQL	
Příkaz	1. měření	2. měření	1. měření	2. měření
Three way JOIN	120 ms	120 ms	60 ms	1,5 ms

Tabulka 27: Výsledky trojnásobného spojení tabulek [vlastní tvorba]

V tomto měření dopadl jednoznačně hůře SŘBD Oracle. Nejen že na zpracování tohoto dotazu potřeboval dvakrát více času, ale ani nedošlo k jeho snížení při opakovaném zadání dotazu. Oproti tomu SŘBD PostgreSQL dokázal při druhém měření snížit čas na pouhých 1,5 ms.

4.5.3 Agregační funkce

Cílem tohoto dotazu bylo vyzkoušet jednoduchý SQL dotaz obsahující agregační funkci, v tomto případě sumu (*SUM*). Dotaz vracel sumy prodejů za jednotlivé země, které byly celkem čtyři. Dotaz vypadal následovně:

```
SELECT a.dep_country, SUM(b.sal_price) FROM department a JOIN sale b
ON a.dep_id = b.sal_dep_id GROUP BY a.dep_country;
```

Dotaz musel projít všech 582 řádků tabulky *SALE* a seskupit je do čtyřech řádků dle tabulky *DEPARTMENT*. Doby potřebné k jeho provedení byly následující:

	Oracle		PostgreSQL	
Příkaz	1. měření	2. měření	1. měření	2. měření
Agregační funkce	40 ms	0 ms (zaokr.)	63 ms	2,6 ms

Tabulka 28: Výsledky agregační funkce [vlastní tvorba]

V tomto měření měly oba systémy podobné výsledky. Čtyřicet milisekund oproti šedesáti v případě prvního měření a jednotky milisekund v případě druhého. Opět je třeba připomenout, že SŘBD Oracle vrací výsledky pouze v desítkách milisekund, tudíž přesná hodnota bude pravděpodobně ležet mezi 0 až 4 milisekundami.

4.5.4 Poddotaz

Posledním testovaným příkazem byl SQL dotaz, který obsahoval řádky z tabulky *SALE* limitované názvem produktu (začínající na písmeno „h“) a identifikačním číslem oddělení realizovaným jako výsledek poddotazu. Příkaz vypadal takto:

```
SELECT sal_product, sal_date, sal_price FROM sale WHERE sal_product
LIKE 'H%' AND sal_price > 20000 AND sal_dep_id = (SELECT dep_id FROM
department WHERE UPPER(dep_name) = 'MAIN DEPARTMENT') ORDER BY sal_date;
```

Příkaz vracel 56 řádků z celkového počtu 582 krát 11 (tabulky *DEPARTMENT* a *SALE*). Záznamy byly také řazeny podle data uskutečnění prodeje. Výsledky měření vypadaly takto:

	Oracle		PostgreSQL	
Příkaz	1. měření	2. měření	1. měření	2. měření
Poddotaz	10 ms	10 ms	56 ms	0 (0,701) ms

Tabulka 29: Výsledky dotazu obsahujícího poddotaz [vlastní tvorba]

Je zajímavé, že SŘBD Oracle sice zvládl provést tento dotaz napoprvé mnohem rychleji, než SŘBD PostgreSQL, nicméně tento čas nedokázal zredukovat při druhém měření, kdy již PostgreSQL dosahoval velmi dobrých výsledků. Zhodnotit toto měření je tedy poněkud obtížné, jelikož vždy bude záležet na situaci, v jaké je tento typ dotazu použit - opakovaná práce s výsledky versus jednorázový dotaz. V první situaci bude podávat lepší výsledky SŘBD PostgreSQL, v druhé pak SŘBD Oracle.

4.5.5 Zhodnocení sekce Měření výkonu

Za nejdůležitější kritérium je v této sekci považováno trojnásobné spojení (*Three way join*). Získávání informací z tabulek, jež jsou vzájemně propojeny přes systém klíčů, je jedna z nejběžnějších operací, jako lze na databázi provádět. Z toho plyne i nejvyšší váha tohoto kritéria. S odstupem pak následují agregační funkce, jelikož jsou taktéž velice často používané. Nejnižší váhu pak získaly shodně sjednocení množin a poddotazy.

	K1	K2	K3	K4	G	V	Pořadí
K1	1	1/3	1/2	1	0,638943	0,137942	3.
K2	3	1	3	3	2,279507	0,492125	1.
K3	2	1/3	1	2	1,07457	0,23199	2.
K4	1	1/3	1/2	1	0,638943	0,137942	3.
					4,631963		

K1 – Sjednocení množin

K2 – Trojnásobné spojení

```
K3 – Agregační funkce
K4 – Poddotaz
G – Geometrický průměr
V – Vážený geometrický průměr (váha kritéria)
```

Tabulka 30: Váhy kritérií sekce Měření výkonu [vlastní tvorba]

Prvním měřeným dotazem bylo sjednocení množin. Zde podával mnohem lepší výsledky SŘBD Oracle.

	Oracle	PostgreSQL	G	V
Oracle	1	4	2	0,8
PostgreSQL	1/4	1	0,5	0,2
		suma:	2,5	

Tabulka 31: Kritérium 1 – Sjednocení množin [vlastní tvorba]

Druhým hodnoceným kritériem bylo kritérium nejdůležitější – trojnásobné spojení. Zde poněkud překvapivě podával mnohem lepší výsledky SŘBD PostgreSQL. SŘBD Oracle nedokázal snížit čas potřebný k provedení dotazu ani při jeho opětovném provádění.

	Oracle	PostgreSQL	G	V
Oracle	1	1/6	0,408248	0,142857
PostgreSQL	6	1	2,44949	0,857143
		suma:	2,857738	

Tabulka 32: Kritérium 2 – Trojnásobné spojení [vlastní tvorba]

Další na řadě bylo třetí kritérium – agregační funkce. Při měření dotazu toho typu podaly oba systémy hodně podobné výsledky – rozdíly jsou minimální.

	Oracle	PostgreSQL	G	V
Oracle	1	2	1,414214	0,666667
PostgreSQL	1/2	1	0,707107	0,333333
		suma:	2,12132	

Tabulka 33: Kritérium 3 – Agregační funkce [vlastní tvorba]

Posledním hodnoceným kritériem v této sekci bylo měření času poddotazu. Zde bylo opět hodnocení poněkud obtížnější, jelikož SŘBD Oracle měl mnohem lepší čas prvního měření, nicméně stejnou hodnotu měl i ve druhém měření, ve kterém již SŘBD PostgreSQL podal mnohem lepší výsledek. Nakonec byl SŘBD PostgreSQL vyhodnocen jako lepší, především díky mnohem lepšímu výsledku druhého měření.

	Oracle	PostgreSQL	G	V
Oracle	1	1/3	0,57735	0,272166
PostgreSQL	3	1	1,732051	0,816497
		suma:	2,309401	

Tabulka 34: Kritérium 4 – Poddotaz [vlastní tvorba]

Výsledné vyhodnocení této sekce pak vypadalo následovně:

Kritérium	Váha	Oracle	PostgreSQL
K1	0,137942	0,8	0,2
K2	0,492125	0,142857	0,857143
K3	0,23199	0,666667	0,333333
K4	0,137942	0,272166	0,816497
	Celkem:	0,372861	0,63937

```
K1 – Sjednocení množin
K2 – Trojnásobné spojení
K3 – Agregační funkce
K4 – Poddotaz
```

Tabulka 35: Vyhodnocení sekce Měření výkonu [vlastní tvorba]

Jak je vidět, v této sekci dosáhl lepšího výsledku SŘBD PostgreSQL.

4.6 Dokumentace a podpora

Cílem této sekce je zhodnotit kvalitu dokumentace a komunitní podporu pro dané systémy. Kvalitní dokumentace je základním kamenem při práci se systémem a také při řešení problémů, s čímž může pomoc i široká komunita uživatelů.

U SŘBD Oracle lze v případě zakoupení licence očekávat i oficiální placenou podporu, nicméně tato možnost není v porovnání zohledněna, jelikož jsou porovnávány volně dostupné systémy.

Je jasné, že charakter této sekce je hodně subjektivní a stejně tak tuto sekci nelze považovat za kruciální faktor při volbě SŘBD, ale svou roli určitě má.

4.6.1 Dokumentace

Online dokumentace systému řízení báze dat PostgreSQL je přístupná na adrese http://www.postgresql.org/docs/9.3/static/index.html. Je poměrně přehledně rozčleněna do jednotlivých kapitol, které jako celek popisují všechny části databáze. Za velkou výhodu

považuji možnost volby verze na vrchu stránky, pomocí níž lze snadno přepínat mezi dokumentacemi pro jednotlivé verze SŘBD PostgreSQL.

Některé sekce však mohou být trochu matoucí. Například funkcionalita spojení dvou tabulek pomocí klauzule JOIN je popsána v kapitole 2.6, ovšem tato kapitola je brána jako *tutorial* (výukový program). Viz věta „**Exercise:** There are also right outer joins and full outer joins. Try to find out what those do.“ [21]. Lépe je pak funkcionalita popsána v kapitole 7.2.1, kde již lze najít všechny možné případy klauzule *JOIN*.

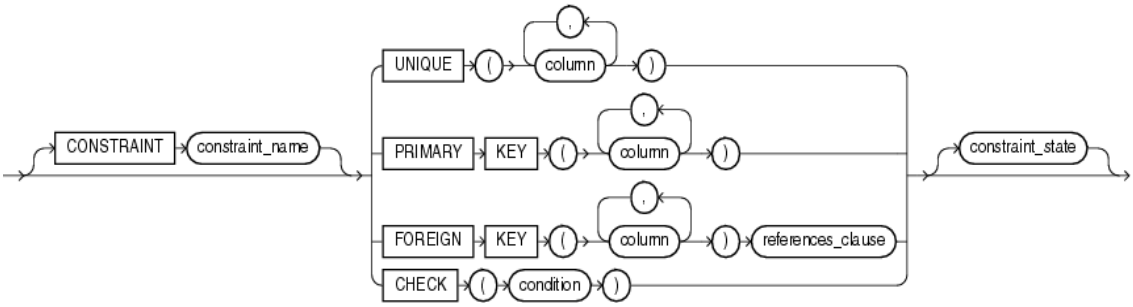
Vyhledávání v dokumentaci sice funguje, ovšem většinou je rychlejší zadat hledaná slova do vyhledávače (např. Google), než se je pokoušet „ručně“ vyhledat v dokumentaci.

Dále je v dokumentaci možno spatřit určitou nekonzistenci v podrobnostech určitých témat. Některé obecně známé problematiky jsou v dokumentaci rozepsány poměrně hodně podrobně (například klauzule *FROM*), což není nutně špatně, a naopak některé složitější funkcionality jsou v dokumentaci popsány až příliš obecně (namátkou struktura databáze, funkcionalita indexů). V takových případech je pak nutno vyhledávat relevantní informace z odborných (i neodborných – např. internetové diskuze) zdrojů mimo oficiální dokumentaci.

Celkově lze dokumentaci SŘBD PostgreSQL zhodnotit jako jasnou a přehlednou pro naprostou většinu uživatelů. Občas je trochu obtížnější se v ní zorientovat (například při hledání souvisejících témat) a občas lze narazit na výše uvedený problém se špatně odhadnutou mírou podrobností. I tak se jedná o velkého pomocníka při práci se systémem.

Dokumentace systému řízení báze dat Oracle verze 11g je přístupná na adrese http://docs.oracle.com/cd/E11882_01/index.htm. Členění do kapitol a podkapitol je přehledné a logické. Na rozdíl od dokumentace PostgreSQL zde lze nelze jednoduše přepínat mezi verzemi databáze, vždy je nutné znovu vyhledat dokumentaci pro odpovídající verzi.

Dokumentace je obecně mnohem propracovanější a detailnější než v případě PostgreSQL. Za jednu z největších výhod lze považovat grafické zpracování parametrů jednotlivých příkazů pomocí diagramu. Například z něj lze jasně a snadno vyčíst přesnou syntaxi daného příkazu.



Obrázek 8: Ilustrace grafického zpracování klauzule v dokumentaci SŘBD Oracle [29]

Stejně jako v případě SŘBD PostgreSQL platí, že pokud hledáme specifický problém, je nejrychlejší cestou jeho zadání do vyhledávače. Nutno však podotknout, že vyhledávání uvnitř dokumentace systému Oracle je propracovanější a přehlednější. Vyhledávání totiž vrací kromě kapitol obsahujících zadané slovo i klíčová slova dokumentace. Například při hledání slova *JOIN* vrátí dokumentace SŘBD Oracle jako jeden z prvních výsledků kapitolu „Joins“, zatímco dokumentace PostgreSQL při zadání totožného slova vrátí poněkud nepřehledný seznam kapitol, v nichž se slovo vyskytlo.

Celkově lze dokumentaci SŘBD Oracle zhodnotit jako mnohem obsáhlejší a vydatnější, než dokumentaci SŘBD PostgreSQL. To ovšem neznamená, že by dokumentace systému PostgreSQL byla nedostačující. Jen jí občas chybí jakási pomyslná přidaná hodnota.

4.6.2 Podpora

Vzhledem k velké rozšířenosti a popularitě SŘBD Oracle je možné mnohé problémy řešit pomocí komunity uživatelů. Většina běžných problémů (chybových upozornění apod.) je často rozebrána na různých diskuzních fórech a komunitních blozích. Mezi nejznámější patří *Ask Tom* (https://asktom.oracle.com/pls/apex/f?p=100:1:0) a *Oracle Communities* (https://www.oracle.com/communities/index.html). Mnoho užitečných informací lze najít i na webu *Stack Overflow* (http://stackoverflow.com/), který se zaměřuje řešení chyb a problémů v různých jazycích a systémech obecně.

Při zakoupení licence je pak možno využívat i oficiální podpory společnosti Oracle. Zároveň existuje ve světě i v České republice mnoho firem, jejichž hlavním předmětem činnosti je technická podpora související s technologiemi společnosti Oracle.

Vývojáři PostgreSQL na svém webu zveřejňují tzv. *Mailing lists*, což jsou přepisy dotazů z emailových konverzací mezi vývojáři a uživateli [32]. Zde lze najít mnoho užitečných

informací, ovšem jejich zobrazení není příliš přehledné. Hlavním zdrojem informací při vyskytnuvším se problému tak často bývá již zmíněná webová stránka *Stack Overflow*.

Jelikož PostgreSQL je systém vyvíjený komunitou, nenabízí žádnou formu placené podpory. Na webu je však uveden seznam firem pro jednotlivé kontinenty, které podporu pro SŘBD PostgreSQL nabízejí.

4.6.3 Zhodnocení sekce Dokumentace a podpora

V této sekci byla hodnocena pouze dvě kritéria. Kvalitní dokumentace (K1) je považována za důležitější parametr, než je podpora (K2). Váha kritérií vypadala následovně:

	K1	K2	G	V	Pořadí
K1	1	5	2,236068	0,833333	1.
K2	1/5	1	0,447214	0,166667	2.
			2,683282		

K1 – Dokumentace
K2 – Komunitní podpora
G – Geometrický průměr
V – Vážený geometrický průměr (váha kritéria)

Tabulka 36: Váha kritérií sekce Dokumentace a podpora [vlastní tvorba]

Prvním hodnoceným kritériem byla dokumentace. Ta byla vyhodnocena jako lepší v případě SŘBD Oracle.

	Oracle	PostgreSQL	G	V
Oracle	1	3	1,732051	0,75
PostgreSQL	1/3	1	0,57735	0,25
		suma:	2,309401	

Tabulka 37: Kritérium 1 – dokumentace [vlastní tvorba]

Druhým kritériem byla podpora databází. Jelikož do srovnání nebyla zahrnuta kvalita placené podpory společnosti Oracle, mají oba systémy podobné výsledky. I tak ale platí, že větší rozšířenost SŘBD Oracle zvyšuje pravděpodobnost, že v případě problému bude rychle nalezeno jeho odpovídající řešení.

	Oracle	PostgreSQL	G	V
Oracle	1	2	1,414214	0,666667
PostgreSQL	1/2	1	0,707107	0,333333
		suma:	2,12132	

Tabulka 38: Kritérium 2 – podpora [vlastní tvorba]

Tabulka s vyhodnocením této sekce pak vypadala následovně:

Kritérium	Váha	Oracle	PostgreSQL
K1	0,833333	0,75	0,25
K2	0,166667	0,666667	0,333333
	Celkem:	0,736111	0,263889

K1 – Dokumentace

K2 – Komunitní podpora

Tabulka 39: Vyhodnocení sekce Měření výkonu [vlastní tvorba]

Jak je vidět, z této sekce vyšel lépe SŘBD Oracle.

5. Shrnutí výsledků

5.1 Matematické vyhodnocení

Pro vyhodnocení všech sekcí bylo nejprve potřeba určit jejich váhy. Největší vahou byla po párovém porovnání ohodnocena sekce 2 – Základní funkcionalita databází (S2). Tato sekce obsahuje oblasti, které se používají v každodenním provozu databáze a jejich zpracování je proto důležitým parametrem.

S velkým odstupem byla jako druhá nejdůležitější vyhodnocena sekce Dokumentace a podpora (S5), především kvůli dokumentaci. Bez kvalitní dokumentace nelze provozovat databázi. Stejně tak jsou nepoužitelné její pokročilé funkce, pokud nejsou dostatečně přehledně zdokumentovány a vysvětleny.

Po této sekci následovala s již malým odstupem sekce Rozšířená funkcionalita databází (S3). Některé nadstandardní funkce, které oba systémy nabízejí, mohou výrazně usnadnit práci s nimi či zlepšit výkon, proto lze tuto sekci považovat za signifikantní.

Čtvrtou nejdůležitější sekcí byla sekce Měření výkonu (S4). Výkon je sice důležitým ukazatelem, ale lze ho testovat různými způsoby. Zabývat se všemi takovými způsoby by dalece přesahovalo rozsah této práce a stejně by pravděpodobně nebyly vždy zcela průkazné.

Vždy budou existovat typy příkazů, se kterými si daný SŘBD poradí lépe než konkurenční a opačně. Proto je potřeba brát jakékoliv testy výkonu s určitou rezervou.

Nejmenší váhu pak měla sekce Instalace a obecná použitelnost (S1). Jak je již v sekci řečeno, databázi instalujeme zpravidla jednou (pokud si odmyslíme různé havárie), tudíž nejde o příliš významný ukazatel. Stejně tak u obecné použitelnosti nelze vyloženě určit, které řešení (například z hlediska syntaxe) je lepší než druhé.

	S1	S2	S3	S4	S5	G	V	Pořadí
S1	1	1/7	1/5	1/3	1/3	0,316474	0,043709	5.
S2	7	1	7	5	4	3,965018	0,547625	1.
S3	5	1/7	1	2	1/5	0,778371	0,107504	3.
S4	3	1/5	1/2	1	1/2	0,684255	0,094505	4.
S5	3	1/4	5	2	1	1,496278	0,206657	2.
					suma:	7,240396		

S1 – Instalace a obecná použitelnost
S2 – Základní funkcionalita databází
S3 – Rozšířená funkcionalita databází
S4 – Měření výkonu
S5 – Dokumentace a podpora
G – Geometrický průměr
V – Vážený geometrický průměr (váha kritéria)

Tabulka 40: Váhy jednotlivých hodnocených sekcí [vlastní tvorba]

Po dosazení dílčích výsledků z předchozích sekcí získáme výsledné hodnocení. V hodnocení jsou tedy zohledněny váhy jednotlivých sekcí a výsledky obou porovnávaných databází:

Kritérium	Váha	Oracle	PostgreSQL
S1	0,043709	0,262796	0,737204
S2	0,547625	0,566864	0,345403
S3	0,107504	0,652763	0,191646
S4	0,094505	0,372861	0,63937
S5	0,206657	0,736111	0,263889
	Celkem:	0,57945	0,356935
	Umístění:	1.	2.

Tabulka 41: Výsledné zhodnocení obou databází [vlastní tvorba]

Tabulka ukazuje výsledné pořadí obou systémů – lze tedy konstatovat, že SŘBD Oracle je na základě srovnávaných kritérií lepší, než SŘBD PostgreSQL.

5.2 Slovní vyhodnocení

K matematickému vyhodnocení srovnání systémů řízení báze dat Oracle a PostgreSQL je třeba doplnit několik informací. SŘBD Oracle vyšel lépe ze srovnání, které bylo postaveno na vybraných kritériích. Ta byla ohodnocena určitou váhou vzhledem k jejich důležitosti. Nejdůležitější sekcí byla Základní funkcionalita databází, ve které SŘBD Oracle dosahoval lepších výsledků. Patří sem hlavně širší možnosti zálohování, více nástrojů při práci s aktivačními událostmi a možnost auditovat procesy v databázi. Všechny tyto faktory mají vliv především na bezpečnost ukládaných dat. V této sekci měl SŘBD PostgreSQL lepší výsledky pouze v možnostech indexování.

K celkovému vítězství SŘBD Oracle přispěla také jeho kvalitní dokumentace (lépe zpracovaná než v případě SŘBD PostgreSQL). Nutno podotknout, že dokumentace byla hodnocena především způsobem „více je lépe“. Autor této práce například preferuje přesný popis problematiky rozebraný do detailů. Jinému uživateli však může vyhovovat „odlehčená“ forma dokumentace, kterou nabízí SŘBD PostgreSQL. Záleží na osobních preferencích. To, že v tomto srovnání dopadl SŘBD Oracle lépe neznamená, že by SŘBD PostgreSQL měl nekvalitní dokumentaci.

SŘBD Oracle dominoval i sekci číslo tři – Rozšířené funkcionalitě. Zde SŘBD Oracle nabízí mnoho funkcí, které by se daly považovat za nadstandardní či inovativní (například technologie Flashback Query) a které mohou velice usnadnit práci s databází. Dominance SŘBD Oracle v této sekci se dala předpokládat, jelikož tento systém je proslulý svou robustností. To však paradoxně bývá často zmiňováno jako jeho nevýhoda – ne každý uživatel využije všechny funkce, které tato databáze nabízí, a ty pak zůstanou nevyužité. Takový uživatel by se pak měl poohlédnout po „menších“ databázích (méně robustních).

SŘBD PostgreSQL poněkud překvapivě podával lepší výsledky v sekci Měření výkonu. Ve všech měřeních opakovaného průběhu dotazu dokázal mnohonásobně snížit čas potřebný k jeho provedení. SŘBD Oracle takto konzistentní výsledky nepodával. V internetových diskuzích lze často narazit na názor, že Oracle podává nejlepší výkonnostní výsledky až při velmi složitých operacích nad obrovským počtem (miliony) řádků. Takové měření by však dalece přesahovalo rozsah této práce a stejně tak ne každý uživatel vyžaduje od databáze takovéto extrémní výkony.

SŘBD PostgreSQL má také mnohem jednodušší proces instalace. V tomto ohledu jasně předčil SŘBD Oracle, jehož instalace je proces na několik (i více než čtyři) hodin. SŘBD

PostgreSQL také podporuje o něco větší počet platforem, než databáze Oracle, ovšem rozdíl není příliš markantní. Obecně lze říci, že oba SŘBD lze provozovat na opravdu velkém množství platforem. Dále lze některá méně významná řešení také vyhodnotit lépe ve prospěch SŘBD PostgreSQL (například nepoužívání *dummy* tabulek, jednoduché vkládání více řádků do tabulky jedním příkazem).

Jak vyplývá z předchozích řádků, srovnání obou systémů hodně ovlivňují preference jednotlivých kritérií, které může mít každý uživatel nastaven jinak. V praxi může nastat případ, kdy bude mít uživatel naprosto odlišné preference kritérií, než jaké jsou použity v této práci. I tak se ale autor práce snažil vycházet ze základní funkcionality relačních databází – bezpečně uchovávat data, umožňovat flexibilní práci s nimi a řídit přístupová oprávnění k těmto datům. Proto byly jednotlivé váhy sekcí, respektive i konkrétních kritérií zvoleny tak, jak byly zvoleny.

5.3 Problematika ceny

Z hodnocených parametrů byla záměrně vynechána cena. SŘBD Oracle je zdarma pouze k nekomerčním účelům. V minulosti bylo nutné zakoupit licenci i kvůli pouhému nainstalování databáze. Společnost Oracle však přešla k mírnější podobě licenčních ujednání (i díky tlaku konkurence) a její systém je tak možno libovolně provozovat k testování a vývoji nekomerčních produktů.

V práci srovnávaná verze Oracle Enterprise Edition je nejdražší verzí tohoto systému. Při komerčním využití je potřeba zakoupit licenci k používání (*right to use*) produktu a dále platit za podporu pro daný produkt. Cena licence se odvíjí buď podle počtu koncových uzlů, které přistupují do systému (například v běžné prodejně pokladna, samoobslužný terminál, rozhraní ve skladu atd.), anebo podle počtu procesorů, které systém využívá. Dle edice se pak v tomto případě cena odvíjí buď podle počtu fyzických procesorů (Standard Edition) anebo počtu jader (Enterprise Edition). Cena licence této edice pro více jádrové procesory se pak odvíjí dle tzv. *multi-core* faktoru [24].

Nejlevnější možná licence Oracle Enterprise Edition (i s podporou) začíná buď na $9,975 ročně při omezení 25 koncovými uzly (při zvolení této metodiky tvoření ceny), anebo na $19,950 ročně pro jeden procesor (bez omezení počtem uzlů) [33]. Do této ceny nejsou započítány daně.

Měsíční plat databázového administrátora systému Oracle pak začíná na hranici 40.000 Kč [34]. Pokud k tomu připočteme cenu licence (za jeden procesor) a podpory, dostaneme se na částku okolo 80.000 Kč měsíčně za provoz databáze, a to pouze v případě použití nejlevnější možné licence. Jelikož společnost Oracle cílí především na větší podniky, je jasné, že v takovém případě bude tato částka mnohem vyšší (klidně i několikanásobně).

Obecně je velmi těžké určit, pro jaké podniky se hodí daný systém řízení báze dat. Vždy bude záležet na konkrétních požadavcích dané firmy – už jen cena licence SŘBD Oracle se odvíjí podle počtu uživatelů pracujících se systémem. Je jasné, že tento systém (v Enterprise Edition) využijí spíše podniky, které kladou důraz na bezpečný a nepřetržitý provoz databáze s přístupem mnoha uživatelů a například využijí i některé složitější analytické funkce. Typickým příkladem může být firma Škoda Auto či jakákoliv bankovní instituce.

Menší podniky (které si například mohou dovolit vypínat systém v noci kvůli provedení záloh) pak mohou sáhnout po levnější edici SŘBD Oracle anebo zvolit konkurenční systém – například v této práci srovnávaný PostgreSQL. Lze předpokládat, že plat administrátora tohoto systému se bude pohybovat v podobných cenových relacích jako v případě SŘBD Oracle. Problémem však může být dostupnost takovýchto lidí – je třeba si uvědomit, že systém PostgreSQL není zas tolik rozšířený, tudíž bude existovat menší počet lidí, kteří s ním umějí pracovat.

Pokud vezmeme v úvahu modelovou situaci typu jednoduchého elektronického obchodu, lze konstatovat, že v tomto případě se nejspíše nevyplatí kupovat licenci společnosti Oracle. SŘBD PostgreSQL zde s přehledem zvládne všechny požadované funkce za podstatně nižší měsíční náklady.

6. Závěr

Ze srovnání dvou systémů řízení báze dat (Oracle a PostgreSQL) vyšel vítězně systém Oracle. Toto srovnání bylo provedeno na základě předem definovaných parametrů. SŘBD Oracle vyšel ze srovnání lépe především díky lepšímu zpracování zálohování, rozdělování tabulek, auditu a aktivačních událostí. Zároveň se opírá o velmi kvalitní dokumentaci, která je dnes nezbytná při používání jakéhokoliv systému.

SŘBD PostgreSQL podával lepší výsledky mimo jiné v možnostech indexování a v měření výkonu. Zároveň ho lze zhodnotit jako uživatelsky přátelštější a celkově odlehčenější (byť se také jedná o poměrně složitý systém). Zvládne naprostou většinu

běžných úkonu, které vykonáváme při práci s databází. Jeho největší výhodou je, že k jeho užívání není potřeba zakoupit žádnou licenci.

Právě cena licence produktu společnosti Oracle je věc, která toto srovnání komplikuje. I v obecné rovině požaduje každý uživatel (podnik) od systému řízení báze dat odlišné funkce. A od těchto funkcí se pak odvíjí i cena licence SŘBD Oracle. Zatímco z hlediska funkcionality obou srovnávaných systémů lze najít modelové situace, při kterých lze poměrně jednoznačně doporučit volbu jednoho z těchto systémů, z ekonomického hlediska je takovéto obecné doporučení velice obtížné, jelikož vždy bude záležet na konkrétní situaci a přesných požadavcích uživatele.

7. Seznam použitých zdrojů

7.1 Seznam literatury

1. MOMJIAN, Bruce. *PostgreSQL: praktický průvodce*. 1. vyd. Brno: Computer Press, 2003, xiii, 402 s. ISBN 80-722-6954-2.

2. OPPEL, Andrew J. *SQL bez předchozích znalostí: [průvodce pro samouky]*. Vyd. 1. Brno: Computer Press, 2008, 240 s. ISBN 978-80-251-1707-1.

3. Databáze. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-01-17]. Dostupné z: http://cs.wikipedia.org/wiki/Datab%C3%A1ze

4. Firebird. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-01-17]. Dostupné z: http://cs.wikipedia.org/wiki/Firebird

5. Relační databáze. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-01-19]. Dostupné z: http://cs.wikipedia.org/wiki/Rela%C4%8Dn%C3%AD_datab%C3%A1ze

6. SOLAŘ, Tomáš. *Oracle Database 11g: hotová řešení*. Vyd. 1. Brno: Computer Press, 2010, 288 s. K okamžitému použití. ISBN 978-80-251-2886-2.

7. HAAN, Lex de, Tim GORMAN, Inger JØRGENSEN a Melanie CAFFREY. *Beginning Oracle SQL : for Oracle database 12c*. Third edition. xxv, 410 pages. ISBN 978-1-4302-6557-3.

8. GREENWALD, Rick, Robert STACKOWIAK a Jonathan STERN. *Oracle essentials: Oracle database 11g*. 4th ed. Sebastopol, [CA]: O'Reilly, c2008, xviii, 386 p. ISBN 978-059-6514-549.

9. Index (databáze). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-01-19]. Dostupné z: http://cs.wikipedia.org/wiki/Index_%28datab%C3%A1ze%29

10. KAPOUN, Jan. Historie Oracle Corporation. In: *CIO Business World.cz* [online]. 2008 [cit. 2015-01-20]. Dostupné z: http://businessworld.cz/veda-a-historie/historie-oracle-corporation-1801-p1973

11. Sun Microsystems. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2015 [cit. 2015-01-20]. Dostupné z: http://cs.wikipedia.org/wiki/Sun_Microsystems

12. LONEY, Kevin. *Oracle Database: kompletní průvodce*. Vyd. 1. Brno: Computer Press, 2010, 1368 s. ISBN 978-80-251-2489-5.

13. KYTE, Thomas a Darl KUHN. *Expert Oracle Database Architecture*. New York: Apress, 2014. ISBN 978-143-0262-985.

14. ASHDOWN, Lance a Thomas KYTE. ORACLE CORPORATION. *Oracle Database Concepts, 11g Release 2 (11.2)* [online]. 2014 [cit. 2015-01-21]. Dostupné z: http://docs.oracle.com/cd/E11882_01/server.112/e40540/title.htm

15. PostgreSQL. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-01-22]. Dostupné z: http://cs.wikipedia.org/wiki/PostgreSQL

16. Historie projektu PostgreSQL. In: STĚHULE, Pavel. *Root.cz* [online]. 2012 [cit. 2015-01-22]. Dostupné z: http://www.root.cz/clanky/historie-projektu-postgresql/

17. PostgreSQL. STĚHULE, Pavel. *PostgreSQL* [online]. 2014 [cit. 2015-01-22]. Dostupné z: http://postgres.cz/wiki/PostgreSQL

18. SHAW, Peter. SYNCFUSION. *Postgres succinctly* [online]. 2013 [cit. 2015-01-22]. Dostupné z: http://www.syncfusion.com/resources/techportal/ebooks/postgres

19. MOMJIAN, Bruce. *PostgreSQL: introduction and concepts*. Boston, MA: Addison-Wesley, 2001, xxviii, 461 p. ISBN 02-017-0331-9.

20. SHAIK, Baji. PostgreSQL architecture. In: *Simple PostgreSQL blog* [online]. 2013 [cit. 2015-01-22]. Dostupné z: http://bajis-postgres.blogspot.cz/2013/10/postgresql-architecture.html

21. PostgreSQL 9.1.14 Documentation. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL: The world´s most advanced open source database* [online]. 2013 [cit. 2015-01-22]. Dostupné z: http://www.postgresql.org/docs/9.1/static/index.html

22. PostgreSQL 9.0 Memory & Processes. In: *Relational Database Technologies: About Open and Close source databases* [online]. 2011 [cit. 2015-01-22]. Dostupné z: http://raghavt.blogspot.in/2011/04/postgresql-90-memory-processes.html

23. JABLONSKÝ, Josef. Metody vícekriteriálního rozhodování a HTA. In: *CzechHTA* [online]. 2013 [cit. 2015-01-27]. Dostupné z: http://czechhta.cz/wp-content/uploads/2013/02/Metody-v%C3%ADcekriteri%C3%A1ln%C3%ADho-rozhodov%C3%A1n%C3%AD-a-HTA.pdf

24. *Oracle FAQ* [online]. 2008 [cit. 2015-01-27]. Dostupné z: http://www.orafaq.com/

25. Comparison of relational database management systems. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2015, 2015 [cit. 2015-01-27]. Dostupné z: http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems

26. FRIEBELOVÁ, Jana, Jana KLICNAROVÁ a Ludvík FRIEBEL. JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH. *Rozhodovací modely v praxi* [online]. 2006 [cit. 2015-01-27]. Dostupné z: http://www2.ef.jcu.cz/~jfrieb/rmp/index.php

27. CentOS. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2014 [cit. 2015-01-29]. Dostupné z: http://cs.wikipedia.org/wiki/CentOS

28. OTN developer License Terms. ORACLE CORPORATION. *Oracle | Hardware and Software, Engineered to Work Together* [online]. 2014 [cit. 2015-01-29]. Dostupné z: http://www.oracle.com/technetwork/licenses/standard-license-152015.html

29. Oracle Database Online Documentation 11g Release 2 (11.2). ORACLE CORPORATION. *Oracle Help Center* [online]. 2014 [cit. 2015-02-01]. Dostupné z: http://docs.oracle.com/cd/E11882_01/index.htm

30. BRYLA, Bob a Kevin LONEY. *Mistrovství v Oracle Database 11g*. Vyd. 1. Brno: Computer Press, 2009, 700 s. ISBN 978-80-251-2189-4.

31. YALAMANCHI, Aravind a Rod WARD. Oracle Database Rules Manager and Expr ession Filter Developer's Guide, 11 g Release 2 (11.2). *Oracle Help Center* [online]. 2011 [cit. 2015-02-08]. Dostupné z: http://docs.oracle.com/cd/E11882_01/appdev.112/e14919.pdf

32. PostgreSQL: Community. *PostgreSQL: The world´s most advanced open source database* [online]. 2015 [cit. 2015-02-21]. Dostupné z: http://www.postgresql.org/community/

33. Oracle Store. ORACLE CORPORATION. *Oracle | Hardware and Software, Engineered to Work Together* [online]. 2015 [cit. 2015-03-09]. Dostupné z: https://shop.oracle.com/pls/ostore/f?p=dstore:home:0

34. *Jobs.cz - Inspirujeme k úspěchu - nabídka práce, volná pracovní místa, brigády i vzdělávání a rozvoj* [online]. 2015 [cit. 2015-03-09]. Dostupné z: http://www.jobs.cz/

7.2 Seznam obrázků

7.3 Seznam tabulek

8. Přílohy

8.1 Instalační skript datové struktury pro měření výkonu (Oracle)

```
CREATE TABLE EMPLOYEE (
   emp_id                  NUMBER not null,
   emp_first_name          VARCHAR2(50),
   emp_last_name           VARCHAR2(50),
   emp_birth_number        NUMBER(10),
   emp_dep_id              NUMBER,
   constraint PK_EMPLOYEE PRIMARY KEY (emp_id)
);

CREATE TABLE DEPARTMENT (
   dep_id                  NUMBER not null,
   dep_name                VARCHAR2(50),
   dep_city                VARCHAR2(50),
   dep_country             VARCHAR2(50),
   constraint PK_DEPARTMENT PRIMARY KEY (dep_id)
);

CREATE TABLE SALE (
   sal_id                  NUMBER not null,
   sal_product             VARCHAR2(50),
   sal_price               NUMBER,
   sal_date                DATE,
   sal_dep_id              NUMBER,
   constraint PK_SALE PRIMARY KEY (sal_id)
);

ALTER TABLE EMPLOYEE add constraint FK_EMP_REF_DEP foreign key
(emp_dep_id) references DEPARTMENT (dep_id) on delete cascade;

ALTER TABLE SALE add constraint FK_SAL_REF_DEP foreign key
(sal_dep_id) references DEPARTMENT (dep_id) on delete cascade;

CREATE SEQUENCE emp_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE dep_seq START WITH 1 INCREMENT BY 1;
```

```
CREATE SEQUENCE sal_seq START WITH 1 INCREMENT BY 1;
```

8.2 Instalační skript datové struktury pro měření výkonu (PostgreSQL)

```
CREATE TABLE EMPLOYEE (
   emp_id                  NUMERIC not null,
   emp_first_name          VARCHAR(50),
   emp_last_name           VARCHAR(50),
   emp_birth_number        NUMERIC(10),
   emp_dep_id              NUMERIC,
   constraint PK_EMPLOYEE PRIMARY KEY (emp_id)
);

CREATE TABLE DEPARTMENT (
   dep_id                  NUMERIC not null,
   dep_name                VARCHAR(50),
   dep_city                VARCHAR(50),
   dep_country             VARCHAR(50),
   constraint PK_DEPARTMENT PRIMARY KEY (dep_id)
);

CREATE TABLE SALE (
   sal_id                  NUMERIC not null,
   sal_product             VARCHAR(50),
   sal_price               NUMERIC,
   sal_date                DATE,
   sal_dep_id              NUMERIC,
   constraint PK_SALE PRIMARY KEY (sal_id)
);

ALTER TABLE EMPLOYEE add constraint FK_EMP_REF_DEP foreign key
(emp_dep_id) references DEPARTMENT (dep_id) on delete cascade;

ALTER TABLE SALE add constraint FK_SAL_REF_DEP foreign key
(sal_dep_id) references DEPARTMENT (dep_id) on delete cascade;

CREATE SEQUENCE emp_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE dep_seq START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE sal_seq START WITH 1 INCREMENT BY 1;
```