Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Nové technologie a přístupy v Oracle Database 12c

Bakalářská práce

Autor: Radek Souček
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Barbora Tesařová, Ph.D.

Hradec Králové srpen 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 15. 8. 2015 Radek Souček

Poděkování

Rád bych poděkoval Ing. Barboře Tesařové, Ph.D. za vřelý přístup, ochotu, cenné rady a připomínky, kterými přispěla k vypracování této bakalářské práce.

Anotace

Předmětem této práce je seznámení s novými technologiemi a přístupy v nové verzi databázového systému Oracle Database 12c. V teoretické části práce popisuje multitenantní architekturu, která v sobě zahrnuje kontejnerovou databázi a zásuvnou databázi, automatickou optimalizaci dat a nové funkce z oblasti zabezpečení a tvorby databáze. Praktická část následně zobrazuje nové technologie a přístupy z teoretické části na ukázkových praktických příkladech.

Klíčová slova:

Oracle, Database, 12c, cloud, multitenantní architektura, konsolidace, kontejnerová databáze, zásuvná databáze, PDB, CDB, optimalizace, bezpečnost

Annotation

Title: New technologies and approaches in Oracle Database 12c

The object of this bachelor thesis is to introduce new technologies and approaches in the new version of database system called Oracle Database 12c. The theoretical part describes multitenant architecture which contains the pluggable database and the container database; automatic data optimization and new security and database creation functions. The practical part deals with realistic examples of the new technologies and approaches mentioned in the theoretical part.

Keywords:

Oracle, Database, 12c, cloud, multitenant architecture, consolidation, pluggable database, container database, PDB, CDB, optimization, security

Obsah

Úvod

Cílem práce je představit a prozkoumat nové důležité technologie a přístupy v nové majoritní verzi databázové platformy Oracle Database 12c. Oracle Database 12c představuje nové technologie v oblasti databázových systémů, kdy nová majoritní verze byla světu představena v červenci 2013 a označena písmenem „c“, které vyznačuje nejdůležitější prvek nové verze – navrženo pro cloud.

Ačkoliv cloud je nejvlivnější novinka nové verze, dotkne se tato práce cloudu pouze okrajově a zaměří se na jiné nové funkce, na první pohled méně známé, ale stejně důležité a umožňující právě přechod do cloudu. Primárně bude v této práci popsána nová multitenantní architektura, kde budou ukázány její základní prvky, vlastnosti i příklady práce s ní. Následně bude vysvětlen pojem konsolidace dat, který velmi úzce souvisí s multitenantní architekturou, kdy se jedná o zjednodušení a zefektivnění infrastruktury dat s cílem snížení nákladů dané společnosti, a na závěr budou popsány dodatečné prvky pro optimalizaci a kompresi dat. Předešlé funkce spolu úzce souvisí ve formě konsolidace v několika úrovních. Od multitenantní architektury a optimalizace dat přejde práce k novým funkcionalitám v oblasti zabezpečení jako je např. redigování dat a na závěr práce přejde k novým možnostem při tvorbě a úpravě struktury databáze jako je viditelnost sloupců tabulky a nová technika autoinkrementace.

Hlavním cílem této práce bude poukázání nejen na to, čím je nová architektura převratná, ale i proč ji využívat a nastínit důležité funkce této nové verze databázového systému, které pomohou společnostem používajícím tento systém zjednodušit práci, správu dat a ušetřit finance. V praktické části bude na praktických příkladech představena základní práce a správa multitenantní architektury, optimalizace dat, bezpečnosti a doplňující příklady pro tvorbu struktury databáze.

1 Teoretická část

Oracle Database 12c je nová majoritní verze systému řízení báze dat vydaná v červenci roku 2013. Označení v názvu 12c vyjadřuje, že se jedná již o 12. verzi od svého počátku roku 1978, kdy byla vydána první verze tohoto databázového systému, a písmeno „c“ vyjadřuje, že je databázový systém uzpůsoben pro nasazení do cloudu. Nejdříve je však potřeba vysvětlit, co cloud neboli celým názvem cloud computing vlastně znamená.

1.1 Cloud computing

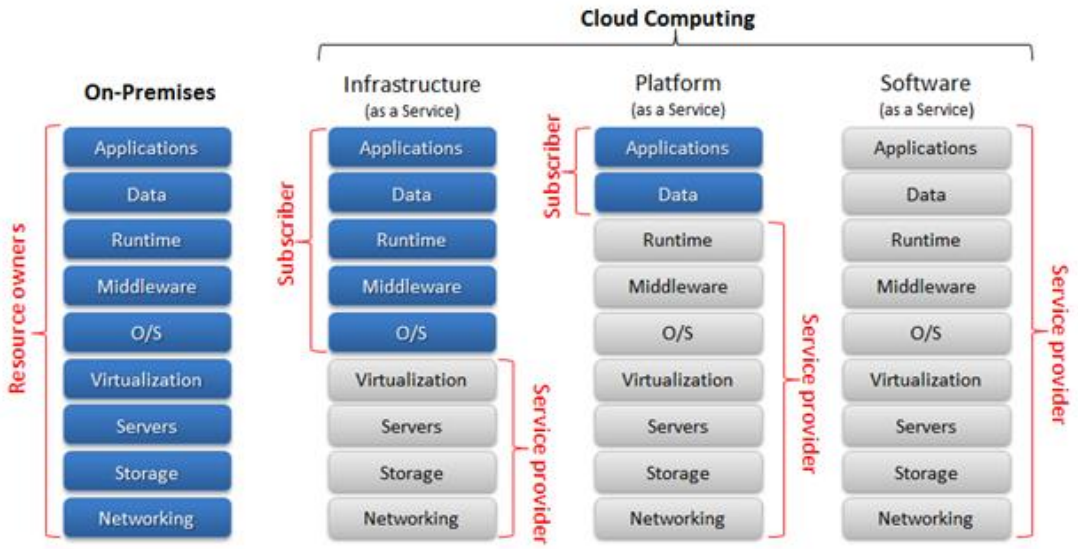
Napsat přesnou definici cloud computingu je velmi obtížné, jelikož v IT oblasti se momentálně vyskytuje několik mírně odlišných definic, ale velmi často se v odborné literatuře objevuje definice od institutu National Institute of Standards and Technology (NITS), která cloud computing popisuje jako model, který splňuje následujících pět charakteristik: [1]

- Samoobslužná služba na vyžádání
    - Zákazník požaduje využití výpočetních prostředků či celého prostředí bez lidské interakce s poskytovatelem.
- Přístup po síti
    - Zákazník má neomezený přístup odkudkoliv pomocí internetové sítě.
- Sdílení zdrojů
    - Výpočetní prostředky jsou rozděleny mezi více zákazníků v závislosti na klientských požadavcích.
- Vysoká pružnost
    - Zajistit dynamické přiřazování a odebírání prostředků zákazníkům.
- Měřitelná služba
    - Zákazník platí pouze za využité výpočetní prostředky.

Obecně lze shrnout definici jako standardizovaný a zautomatizovaný model, který je kdykoliv dostupný na vyžádání.

Cloud computing následně můžeme rozdělit do tří základních modelů neboli vrstev, které se liší v nabízených službách a rozdělení správy a odpovědnosti za dané služby:

- Software as a Service (SaaS)
- Platform as a Service (Paas)
- Infrastructure as a Service (IaaS)



Obrázek 1: Vrstvy cloud computingu znázorňující rozdělení odpovědnosti za poskytované služby (Zdroj: http://blogs.technet.com/b/yungchou/archive/2010/11/15/cloud-computing-primer-for-it-pros.aspx)

1.1.1 Oracle v cloudu

Ačkoliv se zpočátku Oracle cloud computingu bránil, nyní v něm vidí budoucnost a stává se jeho prioritou. O tom vypovídá i množství nabízených služeb na všech vrstvách cloud computingu od poskytování výpočetních zařízení (IaaS), specializovaných platforem (PaaS) až po množství specifických aplikací (SaaS) jako jsou např. aplikace pro plánování podnikových zdrojů (ERP systém), mzdové a personální aplikace (HRM systém) či různé marketingové aplikace.

V případě poskytování databází Oracle vytvořil ještě čtvrtou úroveň cloud computingu – Database as a Service. Tato úroveň se nachází na rozhraní mezi IaaS a PaaS úrovní, kdy zákazník obdrží spravovatelnou a konfigurovatelnou instanci databázového systému běžící na vyhrazeném virtuální serveru poskytovatele. [2]

1.1.2 Cesta ke cloudu

Proces transformace do cloudu se skládá ze tří hlavních kroků, kterými jsou standardizace, konsolidace a automatizace poskytování služeb. Oracle Database 12c je navržen právě tak, aby přinášel nové klíčové funkce v každém kroku této transformace.

Jednotlivé kroky transformace do cloudu:

- Standardizace
    - Prvním a základním krokem je standardizace, která má za cíl snížení složitosti a sjednocení celého IT prostředí společnosti, od hardware, software, po služby a procesy. Klíčovou vlastností standardizace je sjednocení různorodých prostředí ve společnosti, což má za následek snížení administrační režie a následně tedy i provozní náklady. [3] Právě v této fázi přichází Oracle Database 12c s hlavním prvkem nové verze – multitenantní architekturou. Jedná se o sdílenou architekturu, která umožňuje spravovat až 252 databází jako jedinou. Multitenantní architektura bude detailně vysvětlena v následující kapitole.
- Konsolidace
    - V obecném pojetí může být konsolidace vysvětlena jako zjednodušování infrastruktury a snižování nákladů v organizaci jak investičních tak i provozních, o což se pokouší společnosti v zájmu úsporných opatření či zefektivnění chodu společnosti. Již v předchozím kroku při standardizaci došlo k přechodu na sdílenou multitenantní architekturu, kdy na jednom serveru může být více databází, tím pádem dochází k lepšímu využití výpočetních zdrojů, kdy následně není potřeba takové množství serverů, tudíž dochází například k nižší spotřebě energií, zmenšení rozměrů datacenter či snížení režijních nákladů na správu a chod. Konsolidace databáze bude následně upřesněna v následujících kapitolách v souvislosti s multitenantní architekturou. [4]

- Poskytování služeb

    - Poslední krok transformace by měl zvyšovat rychlost a flexibilitu prostředí. Ideálním způsobem je zautomatizovaný samoobslužný portál poskytující koncovým uživatelům služby a správu databáze. Jako příklad lze uvést požadavek programového vývojáře na novou databázi, kdy bez samoobslužného portálu by musel vznést požadavek na databázové oddělení a vyčkat několik hodin či dní na vyřízení žádosti. Cílem této fáze je umožnit uživatelům rychlejší přístup ke službám a automatizace operací z důvodů ušetření času při provádění triviálních operací. Oracle Database 12c v tomto kroku přichází s několika novinkami. I v této fázi se znovu objevuje multitenantní architektura umožňující rychlé vytváření, přesouvání a klonování databází. Dalším prvkem usnadňujícím práci je Automatická optimalizace dat, která má na starosti monitorování, kompresi a přesun dat. [4]

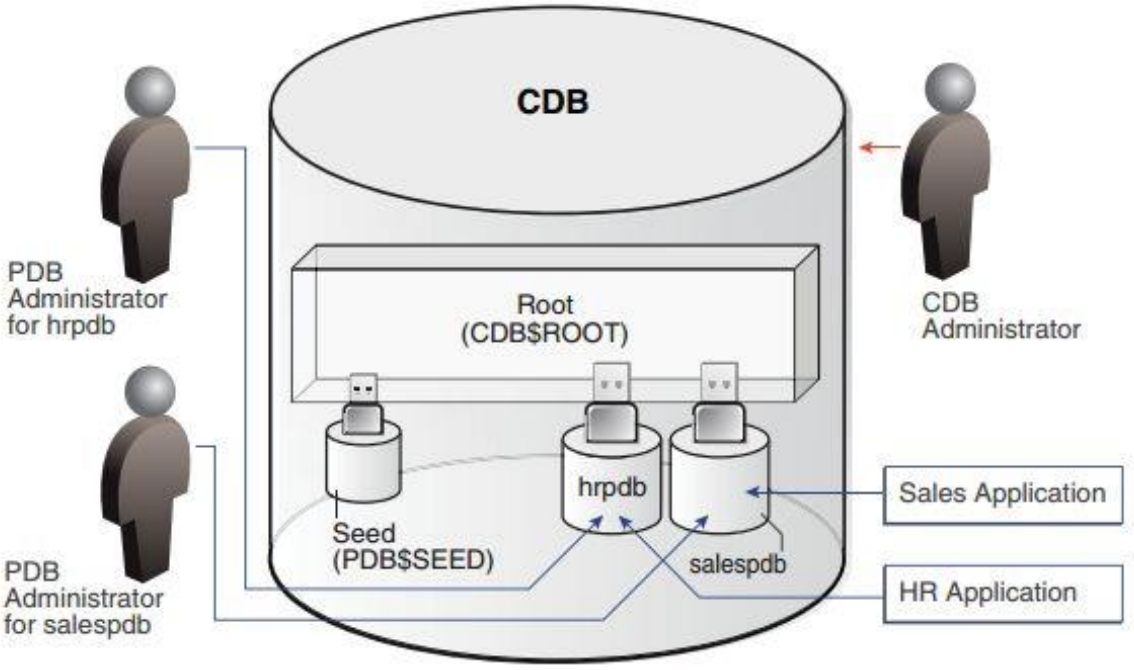
Po shrnutí předešlých poznatků lze napsat, že nejdůležitějším prvkem Oracle Database 12c při cestě do cloudu je multitenantní architektura.

1.2 Multitenantní architektura

Pojem multitenantní architektura (ang. multitenant architecture) je možno volně přeložit jako „více nájemná architektura“, což může být obecně vysvětleno jako architektura pro sdílení např. technických zdrojů více uživateli. Důležitým prvkem multitenantní architektury je izolace uživatelů, aby se vzájemně neovlivňovali z důvodů bezpečnosti a ochrany dat.

Autoři Avril a Hardie [5] popisují multitenantní architekturu jako novou možnost pro zjednodušení konsolidace dat bez nutnosti změn v již existujících aplikacích a správu více databází, kdy v jednom multitenantním databázovém kontejneru označovaném CDB (Container Database) na jednom počítači může být připojeno mnoho tzv. zásuvných databází označených jako PDB (Pluggable Database). Databáze vytvořené před vypuštěním verze Oracle Database 12c se označují jako non-CDB. Důležitou vlastností multitenantní architektury není jen konsolidace

databází, hromadná správa databází, ale i možnost řízení a rozdělování technických zdrojů jednotlivým databázím. Bryn Llewellyn [6] vyobrazuje zjednodušeně CDB databázi jako super databázi obsahující spoustu sub databází.



Obrázek 2: Ukázka multitenantní architektury zobrazující jednu CDB databázi obsahující dvě uživatelské zásuvné PDB databáze (Zdroj: [7])

1.2.1 Multitenantní architektura vs. virtualizace

Velmi důležité je si uvědomit rozdíl mezi multitenantní architekturou a virtualizací a nezaměňovat tyto pojmy, i když mají podobný hlavní cíl – efektivní využití technických prostředků.

V případě serverové virtualizace se jedná o jeden fyzický server, na kterém běží jeden nebo více instancí virtuálních logických serverů obsahující stejné či rozdílné operační systémy a aplikace, kdy zákazník využívá pro své účely jednu instanci.

Zatímco v případě multitenantní architektury se jedná o jeden fyzický server s jednou aplikací, kterou sdílí více zákazníků. V případě multitenantní architektury je nutné zajistit bezpečnost a izolaci jednotlivých zákazníků. V Oracle Database 12c si lze představit zákazníka jako vlastníka jedné PDB databáze. [7]

1.2.2 Konsolidace databáze

Pojem konsolidace databáze je silně spojený s předešlou multitenantní architekturou. Autoři Ashdown a Kyte [7] definují tento pojem takto: „Proces konsolidace dat z více databází do jedné databáze na jednom počítači se nazývá konsolidace databáze.“(přeloženo autorem).

Konsolidace databáze je jednou z největších přednostní multitenantní architektury. Díky hlavní myšlence multitenantní architektury, kdy jedna kontejnerová databáze může obsahovat mnoho zásuvných databází, dochází k redukci počtu počítačových serverů. Umožňuje sdílení technických zdrojů, pamětí a souborů, což má za následek snížení nákladů a zároveň zefektivnění využití technických prostředků.

Z konsolidace technických prostředků vychází i další výhoda, kterou je jednodušší a rychlejší správa, údržba a monitoring databáze z fyzického i logického hlediska, kdy se CDB administrátor stará o jednu fyzickou databázi CDB s desítkami PDB databází. Díky datové struktuře probíhají aktualizace a zálohování na kontejnerové úrovni, z toho plyne, že aktualizace a záplaty se aplikují pouze na kontejnerovou databázi, tudíž není potřeba aktualizovat každou zásuvnou databázi zvlášť. Popřípadě je možno vytvořit nový aktualizovaný a zazáplatovaný kontejner a pouze všechny zásuvné databáze ze zastaralé kontejnerové databáze přesunout do nové aktualizované kontejnerové databáze. Tento přístup nazývaný Oracle „Správa mnoha databází jako jedné“ může být aplikován i na oblast týkající se zálohování, kdy administrátor spustí zálohování na kontejnerové úrovni a provede se záloha včetně všech zásuvných databází. [5]

Výsledkem je tedy zvýšená pracovní efektivita, snížená časová náročnost a možnost snížení počtu zaměstnanců databázového oddělení zásluhou redukce technických prostředků a efektivní správy databází.

1.2.3 Kontejnerová databáze

Každá kontejnerová databáze CDP obsahuje několik druhů kontejnerů. Prvním kontejnerem je kořenový kontejner nazývající se CDB$ROOT a obsahující metadata pro správu zásuvných databází a databázi společných uživatelů, což je databáze známých uživatelů ve všech ostatních kontejnerech. [8]

Dalším kontejnerem je „seed PDB“ neboli v překladu semínko PDB databáze nazývané PDB$SEED. Jedná se o systémovou šablonu PDB databáze, kterou využívá CDB databáze k vytvoření nové PDB databáze. V této šabloně PDB databáze nelze přidávat ani upravovat objekty. [7]

Poslední položkou CDB databáze jsou samotné PDB databáze vytvořené podle uživatelských potřeb. Po vytvoření CDB databáze kontejner neobsahuje žádnou uživatelskou PDB databázi a maximální počet zásuvných databází je stanoven na 252.

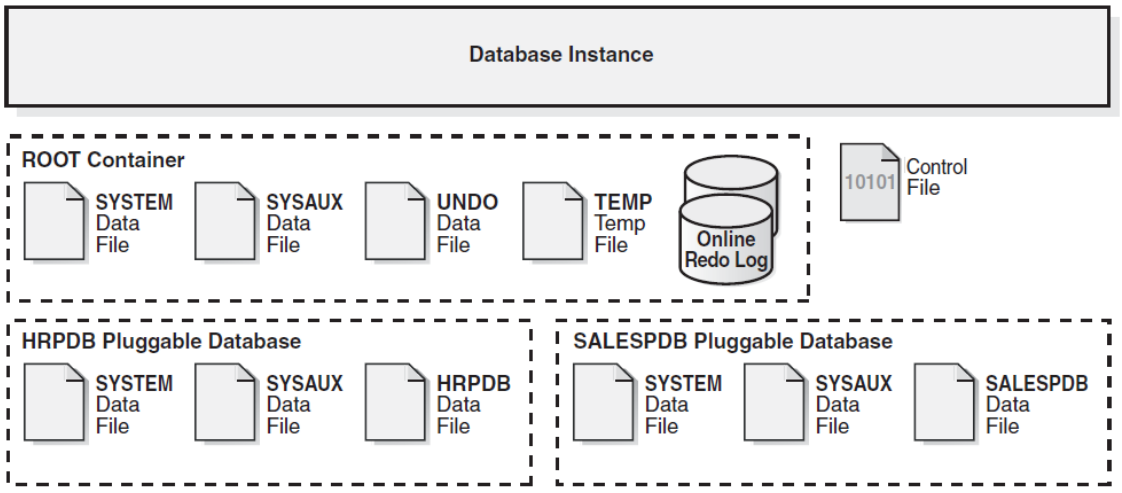
1.2.4 Architektura CDB

Z hlediska fyzické struktury databáze CDB databáze obsahuje stejnou strukturu jako non-CDB databáze s výjimkou tabulkových prostor a datových souborů náležících k PDB databázím.

CDB databáze obsahuje: [7]

- jeden kontrolní soubor (control file):
    - Jedná se o malý binární soubor, který vzniká v okamžiku vytvoření databáze a obsahuje název databáze, jména a umístění přidružených datových souborů a redo log souborů, čas vytvoření databáze, sekvenční číslo logu pro návrat a checkpoint informace.
- jeden online redo log
    - Obsahuje dva a více redo log souborů, které zaznamenávají všechny změny do databázových souborů nebo kontrolního souboru. Slouží k ochraně dat v paměti, která ještě nebyla zapsána do datových souborů, před ztrátou.
- undo soubory (undo data files)
    - Undo soubory obsahují záznamy transakčních akcí, které ještě nebyly úspěšně ukončeny, pomocí kterých lze navrátit databázi do konzistentního stavu např. v případě pádu a zvrátit tak účinek SQL dotazu.

- systémové datové soubory (datafiles)
    - Systémové soubory tvoří hlavní rozdíl mezi multitenantní architekturou a non-CDB databází. Non-CDB databáze obsahuje pouze jednu sadu systémových datových souborů, zatímco v multitenantní architektuře obsahuje CDB databáze a každá její PDB databáze vlastní sadu systémových souborů.
- a jeden nebo více dočasných souborů (temp files)
    - Při vytvoření databáze obsahuje pouze jeden dočasný tabulkový prostor zvaný TEMP, který používají v nativním nastavení všechny PDB databáze daného CDB kontejneru. Dočasné tabulkové prostory obsahují objekty schématu během trvání relaci. Neobsahuje žádné permanentní objekty. V případě potřeby lze vytvořit lokální dočasný soubor v jednotlivých PDB databázích.



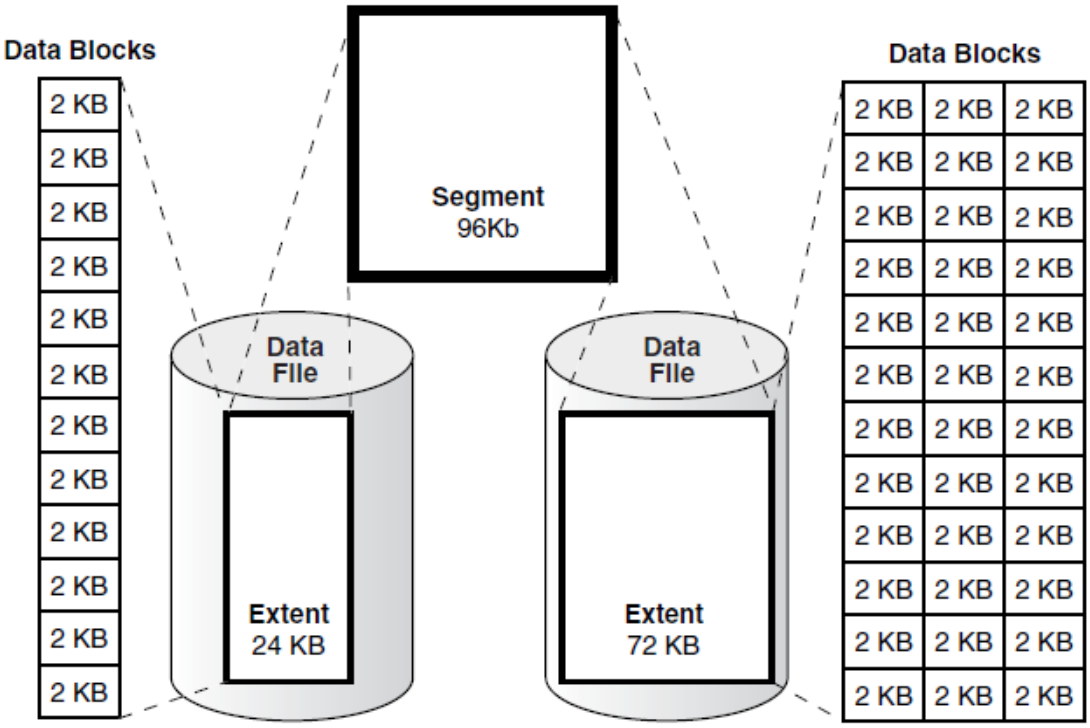
Obrázek 3: Fyzická struktura CDB a PDB (Zdroj: [7])

1.2.5 Zásuvná databáze

PDB databáze je uživatelsky vytvořená množina schémat, objektů a logicky souvisejících struktur. Databázové schéma je logický kontejner pro datové struktury zvané objekty schématu, kdy každý uživatel databáze vlastní svoje databázové schéma pojmenované dle jeho jména. Mezi základní objekty schématu patří tabulky a indexy. Další objekty schématu jsou např. pohledy, sekvence,

synonyma či PL/SQL subprogramy a skripty. Oracle Database dále obsahuje non-schéma objekty, které jsou uloženy mimo schéma a mezi ně patří například uživatelé a role. Databázové objekty jsou ukládány pomocí logických struktur databáze, které alokují logické místo pro všechny data v databázi. Oracle Database rozděluje logické struktury na čtyři základní hierarchické jednotky: [7]

- Datový blok
    - Data jsou uložena v datových blocích, kdy datový blok je nejmenší logickou jednotkou odpovídající specifickému počtu bajtů na fyzickém disku.
- Úsek
    - Úsek je určitý počet logicky souvisejících datových bloků použitých k uložení specifického typu informace.
- Segment
    - Segment je zase soubor úseků alokovaných pro databázový objekt. Každý databázový objekt se skládá z jediného segmentu.
- Tabulkový prostor
    - Databáze je rozdělena do logických struktur zvaných tabulkové prostory. Tabulkový prostor je logický kontejner pro segmenty, kdy tabulkový prostor se skládá nejméně z jednoho datového souboru. Všechny úseky musí být z jednoho tabulkového prostoru, ale zároveň segment může být složen z úseků z různých datových souborů v případě, že tabulkový prostor neobsahuje pouze jeden datový soubor.



Obrázek 4: Logická struktura objektu (Zdroj: [7])

PDB databáze se vytváří zkopírováním semínka, což je rychlá a nenáročná úloha, nebo klonováním již existující PDB. Největší výhodou PDB databáze, podle níž je vytvořen i název této databáze, je schopnost jednoduchého odpojení a následného připojení k jinému CDB kontejneru díky struktuře a rozdělení datového slovníku. [7] Připojování a odpojování PDB databází se uplatní zejména v případě migrace databáze na novou platformu, hardware, operační systém či nový aktualizovaný databázový systém.

PDB databáze obsahuje pouze informace, které jsou specifické pouze pro ni. Obsahuje datové soubory, kde jsou uložena uživatelská data, a může obsahovat dočasné soubory pro práci s daty, jako je např. třídění dat. [7]

Oracle garantuje PDB/non-CDB kompatibilitu, která zaručuje uživateli, že jeho aplikace zobrazí stejný výsledek v PDB databázi i v tradiční non-kontejnerové databázi (non-CDB). [7]

1.2.6 Datový slovník

Datový slovník se skládá z metadat databáze, která obsahují informace o definicích každého objektu schémat v databázi (např. tabulky, indexy, pohledy aj.), jménech uživatelů, rolích, oprávněních, ale obsahuje i informace o množství přiděleného prostoru pro objekty schémat. Z pohledu fyzické struktury je datový slovník uložen v tabulkovém prostoru SYSTEM a skládá se z tabulek a pohledů, z čehož plyne, že se lze na datový slovník dotazovat pomocí SQL jazyka. Pohledy jsou rozděleny do několika kategorií dle prefixů zobrazujících objekty dle oprávnění přístupu.

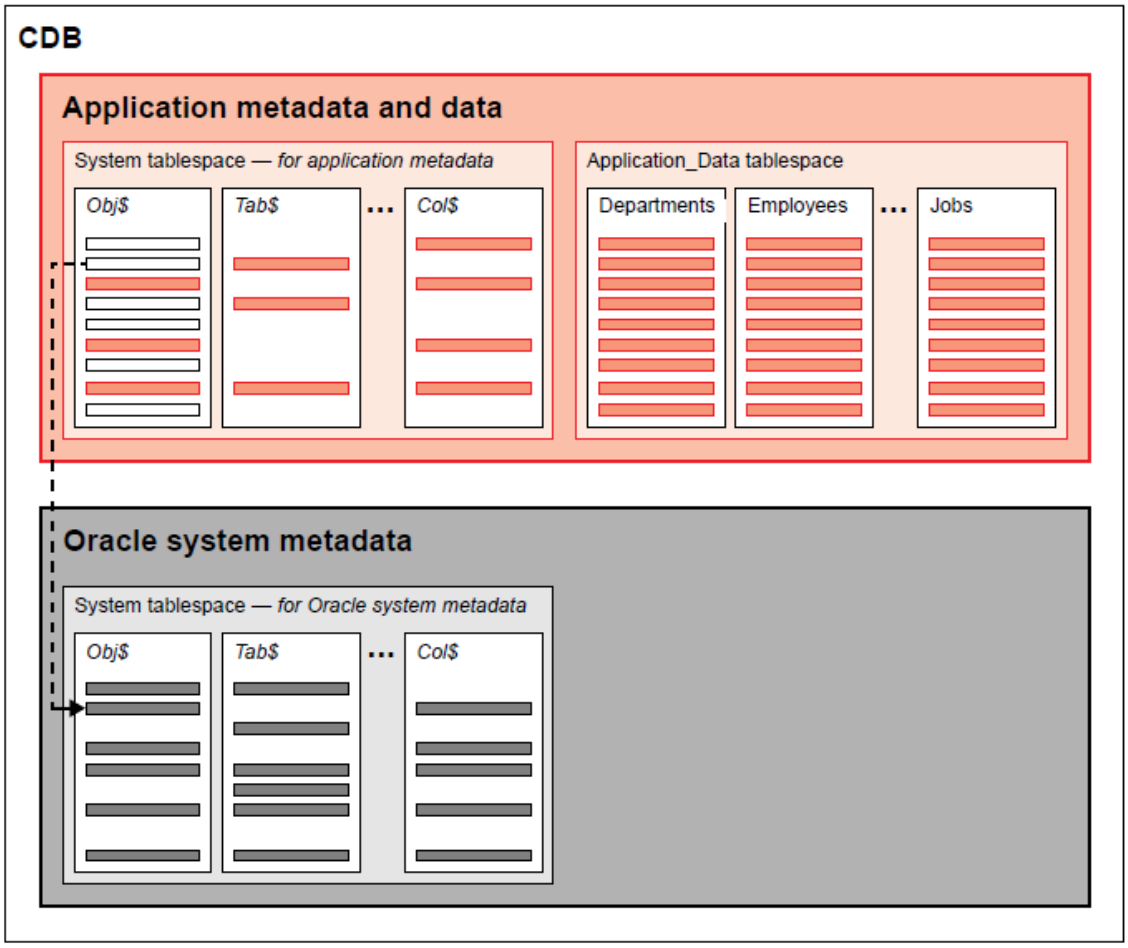
Prefixy pohledů datového slovníku v non-CDB databázi:

- USER_ - vrací všechny objekty vlastněné uživatelem
- ALL_ - vrací všechny objekty, ke kterým má uživatel oprávnění, včetně těch, kterým je vlastníkem
- DBA_ - slouží databázovým administrátorům a vrací úplně všechny objekty

Datový slovník je důležitou součástí Oracle databáze. Je používán databází pro zjišťování informací o všech uživatelích, objektech schémat a struktur pro ukládání. Zároveň databáze zajišťuje změny datového slovníku po každém DDL (Data Definition Language) příkazu. [7]

1.2.6.1 Datový slovník v multitenantní architektuře

Výhody multitenantní architektury, jako je například jednoduché odpojení PDB databáze a přesunutí do jiné CDB databáze, vyplývají především z architektury datového slovníku. V multitenantní architektuře je datový slovník rozdělen mezi kořenový kontejner a PDB databáze, kdy Oracle označuje datový slovník jako virtualizovaný a horizontálně rozdělený. Část datového slovníku v kořenovém kontejneru obsahuje pouze systémová metadata, která jsou společná pro všechny PDB databáze, zatímco každá PDB databáze má svoji část datového slovníku pouze s aplikačními metadaty, uživatelskými daty a tzv. metadatovými ukazateli do datového slovníku v kořenovém kontejneru pro zkompletování celého datového slovníku. [6]



Obrázek 5: Horizontální rozdělení datového slovníku v multitenantní architektuře (Zdroj: [6])

Rozdělení a sdílení části datového slovníku mezi kořenovým kontejnerem a jednotlivými zásuvnými databázemi má dva hlavní cíle: [7]

1) Zamezení redundance dat, kdy se jednotlivé PDB databáze odkazují na sdílenou část datového slovníku, tudíž není potřeba uchovávat jedny a ty samé data na více místech.

2) Jednodušší aktualizace databáze, jelikož systémová data jsou uložena pouze na jednom místě v kořenovém kontejneru, tudíž není potřeba aktualizovat každou PDB databázi zvlášť.

Ke změně dochází i v případě dotazování se na pohledy, kdy byl do multitenantní architektury přidán další prefix CDB_, který je nadřazený všem ostatním prefixům a vrací úplně všechny objekty z CDB databáze i všech PDB databází, zatímco již výše zmiňovaný prefix DBA_ vrací všechny objekty z konkrétního kontejneru nebo

PDB databáze. Zároveň uživatel může upřesnit SQL dotaz pomocí identifikačního čísla každého kontejneru (CON_ID) v klauzuli WHERE.

Tabulka 1: ID hodnoty kontejnerů v multitenantní architektuře (Zdroj: [7])

ID kontejneru	Kontejner
0	CDB nebo non-CDB
1	CDB$ROOT
2	PDB$SEED
Ostatní ID	PDB databáze

1.2.7 Uživatelé a role

V multitenantní architektuře jsou využívány dvě skupiny uživatelských účtů a rolí, které se nazývají společní uživatelé (Common users), resp. společné role (Common roles) a místní uživatelé (Local users), resp. místní role (Local roles).

1.2.7.1 Společní a místní uživatelé

Společný uživatel je databázový účet, který má stejnou a unikátní identitu v kořenovém kontejneru i ve všech již vytvořených nebo v budoucnu vytvořených PDB databází. V případě, že má uživatel potřebná privilegia, se může připojit k jakémukoliv kontejneru. Společné uživatele je možno rozdělit na další dvě skupiny – uživatelsky vytvořené, které musí začínat znaky C## nebo c##, a účty dodávané společností Oracle – např. účet SYS a SYSTEM. [7]

Místní uživatel na rozdíl od společného uživatele spadá pouze pod jedinou PDB databázi, ve které byl vytvořen společně s vlastním schématem. Jedná se tedy o jedinečný účet pouze v rámci jedné PDB databáze, který nemusí na rozdíl od společného uživatele začínat speciálními znaky. [7] Stěžejní vlastností místního účtu je, že se nemůže připojit k žádné jiné PDB databázi. Jediné řešení, jak získat přístup k objektům z jiné PDB databáze, je použití databázového odkazu (database link), který umožňuje spojení mezi dvěma databázemi a vytváří jednu logickou databázi. Jinými slovy databázový odkaz umožňuje získat lokálnímu uživateli jedné databáze přístup k datům vzdálené databáze. [9]

1.2.7.2 Společné a místní role

V obecném pojetí je role skupina oprávnění pro snadnější přidělování oprávnění a jeho správu. Stejně jako společný uživatel, tak společná role se nachází v kořenovém kontejneru i ve všech již vytvořených či v budoucnu vytvořených PDB databází. Stejná pravidla platí i v případě pojmenování a rozdělení rolí na uživatelsky vytvořené, začínající prefixem C## nebo c## a unikátním názvem či dodávané účty společností Oracle – např. role DBA obsahující všechna systémová oprávnění a role PUBLIC, kdy se jedná o speciální roli, která je udělena všem uživatelům. Místní role se taktéž podobají místnímu uživateli a existují tedy pouze v konkrétní PDB databázi, což znamená, že jednotlivé PDB databáze mohou obsahovat stejně pojmenované místní role. [7]

Důležitým prvkem rolí je jejich udělování a pravidla udělování, které popisuje následující tabulka. Jako hlavní pravidlo platí, že společná role může být aplikována pouze na společného uživatele. Zatímco lokální role může být aplikována jak na místního uživatele, tak i na společného uživatele. Samozřejmě lokální role může postihovat společného uživatele pouze v té dané PDB databázi.

Tabulka 2: Společné a místní udělování oprávnění. (Zdroj: [7])

Objekt	Lze udělit lokální oprávnění	Může být uděleno lokálně	Lze udělit společné oprávnění	Může být uděleno společně
Společný uživatel	Ano	-	Ano	-
Místní uživatel	Ano	-	Ne	-
Společná role	-	Ano	-	Ano
Místní role	-	Ano	-	Ne

1.3 Správa dat

V dnešní době velmi rychlého růstu objemu dat určitě nikoho nepřekvapí, že velkou část IT rozpočtu společností tvoří datová uložiště. V některých případech se dokonce uvádí, že velké korporace každé dva až tři roky zdvojnásobují objem svých dat. [10]

Správu dat má na starosti oddělení Information Lifecycle Management (ILM), které se stará o ukládání dat na různá uložiště a určování úrovně jejich komprese podle současných potřeb společnosti při zachování výkonu. Obecně lze vyjádřit ILM jako nástroj sloužící k porozumění a monitorování všech dat ve společnosti s pomocí procesů, politik, software a hardware, jež mohou být použity v každé fázi životního cyklu dat. [11]

Datová uložiště obsahují totiž různě rychlá a kapacitně odlišná úložná zařízení, kdy rychlejší disky jsou drahé, a tím pádem je jejich kapacita obvykle omezena z důvodu velkých pořizovacích nákladů. Často používaná a důležitá data by měla být co nejrychleji dostupná, tudíž musí být na rychlém a výkonném disku. Naopak méně používaná nebo nepoužívaná data jsou buď postupně přesouvána a archivována na nižší úroveň s méně rychlostním diskem, ale s větší kapacitou, nebo smazána. V tomto případě záleží na daných politikách společnosti. Toto přesouvání dat z rychlejších disků na pomalejší se v anglickém jazyce nazývá Storage Tiering. Různé úrovně uložišť mohou být navíc podpořeny ještě úrovněmi komprese (Compression Tiering). Data tedy nejsou pouze přesunuta na pomalejší disk, ale jsou zároveň i komprimována. [11]

Proces vytvoření základní ILM strategie lze popsat třemi základními kroky: [11]

- Definice dat do tříd
    - Jedná se o pohled na veškerá data v organizaci, zjištění, která data jsou důležitá, kde jsou uložena, co se s nimi děje v průběhu životního cyklu, zdali je potřeba řešit jejich ochranu a zjistit právní či obchodní aspekty dat.

- Vytvoření úrovní uložišť
    - Jedná se o rozdělení datových uložišť do úrovní dle rychlosti a přiřazení definovaných tříd z předešlého kroku k jednotlivým úrovním úložišť dle jejich dostupnosti a zároveň s co nejmenšími náklady na úložná zařízení.
- Vytvoření přístupových a migračních politik
    - Tato část zajišťuje zabezpečení dat a přístup k nim, protože v průběhu jejich životního cyklu se můžou přístupová práva měnit. To samé se týká přesunu dat z jedné úrovně uložiště do jiné dle jejich využitelnosti.

Oracle přichází v této oblasti se dvěma novinkami - automatickou optimalizací dat (Automatic Data Optimalization - ADO) a tepelnou mapou (Heat map) sledující stav dat v uložištích. [5] Obě tyto novinky patří do oblasti ILM a pomáhají vytvořit ILM strategii ve společnosti, a tím pádem pomoci snižovat investiční a operační náklady na správu objemných dat a zoptimalizovat výkon z aplikačního i databázového hlediska.

1.3.1 Automatická optimalizace dat

Automatická optimalizace dat, jak již bylo zmíněno výše, je nástrojem ILM a slouží k určování politik pro kompresi a přesun dat. Automatická znamená, že se jedná o proces běžící na pozadí, v němž není vyžadována uživatelská interakce, ačkoliv možnost manuálního řízení ADO databázovým administrátorem je možná.

Automatická optimalizace dat pracuje s tzv. pojmem „teplota dat“, kdy pomocí nástroje Heat map sleduje přístup, poslední změny či datum vytvoření sledovaných dat. Avril a Hardie popisují situaci pomocí stupnice vyjadřující teplotu dat – horké, teplé, chladné a historické. V případě vložení či aktualizace dat jsou data horká a postupně chladnou. Když se dostávají do stavu teplá, znamená to, že je k datům často přistupováno. Následným stavem je stav chladný, který vyjadřuje příležitostný přístup k datům. Poté již může nastat pouze historický stav, ve kterém data nejsou téměř či vůbec využívána. [5] Změna stavu je vyvolána ADO operací v reakci na ADO politiky, které obsahují pravidla vyjadřující, jaká

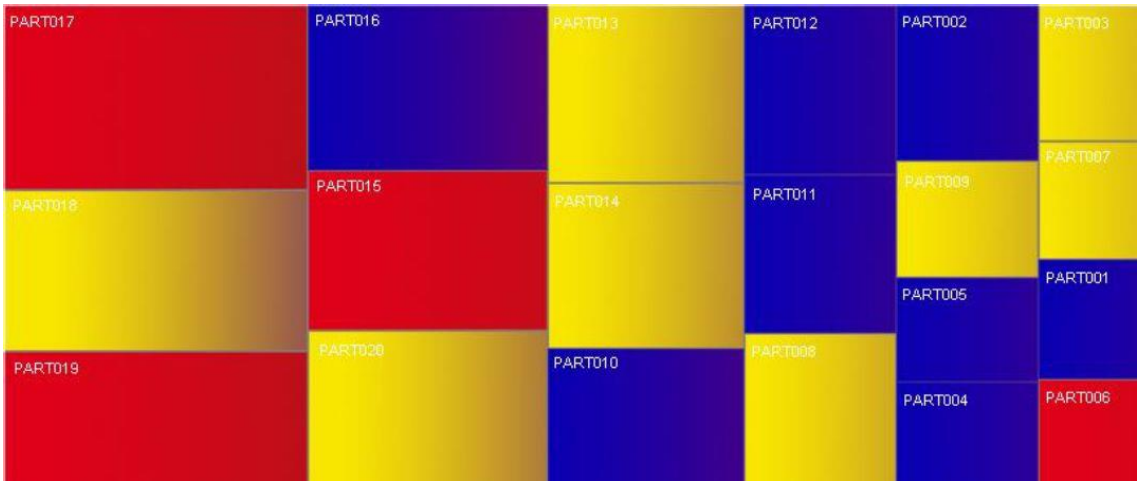
komprimace má být aplikovaná na sledovaná data a kam mají být tato data přesunuta. Obecně lze toto pravidlo napsat jako: Data, ke kterým nebylo přistoupeno / nebyla změněna po X dnech/měsících/letech zkomprimuj a přesuň na danou úroveň.

ADO je ve výsledku hlavně o komprimaci a přesunu dat na různé úrovně uložišť, kdy horká a často přístupná data jsou na rychlých discích a historická data jsou zkomprimována na velkokapacitních uložištích.

1.3.2 Tepelná mapa

Tepelná mapa je nástroj využívaný automatickou optimalizací dat, jehož účelem je automatické sledování informací o použití dat na úrovni řádků a segmentů[1], jinak řečeno tento nástroj sleduje životní cyklus dat a tyto informace dále poskytuje právě ADO, která rozhoduje o kompresi nebo přesunu dat na základě stanovených politik. Na úrovni řádků (datových bloků) jsou sledovány časové záznamy posledních změn, zatímco na úrovni segmentů jsou sledovány jak časové záznamy posledních změn, tak navíc i dotazů na tabulky nebo jejich oddíly. Tepelná mapa tedy udává detailní přehled o přístupu k datům a změn jejich stavu v průběhu času. Na obrázku 6 je znázorněn ukázkový příklad tepelné mapy, kde každý box reprezentuje jeden oddíl tabulky. Velikost boxu je relativní velikosti oddílu a barva znázorňuje frekvenci přístupů. [10]

[1] „Databázové řádky jsou uloženy v blocích, které jsou seskupeny v jistém rozsahu. Segment je množina rozsahů obsahující všechna data logické struktury uložiště v rámci tabulkového prostoru tj. tabulky nebo oddílu.“ (přeloženo autorem) [10]



Obrázek 6: Ukázkový příklad tepelné mapy (Zdroj: [10])

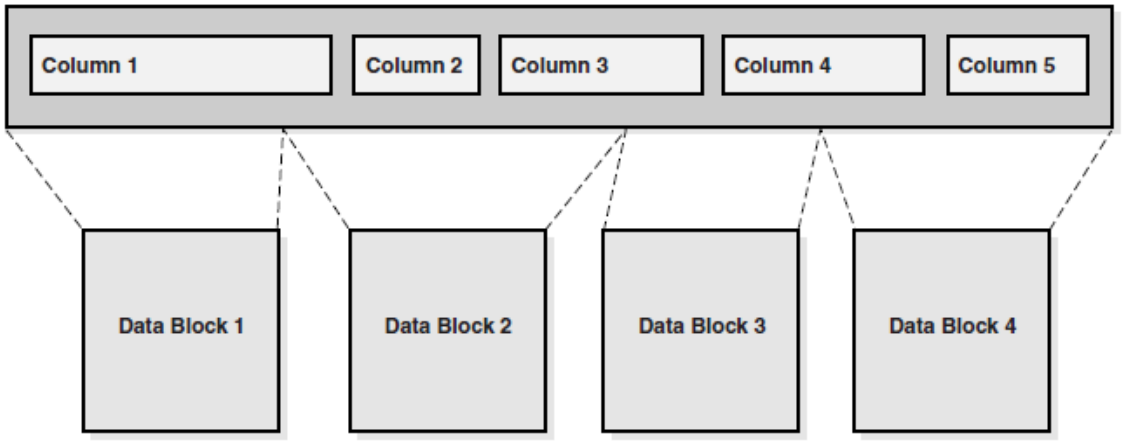
1.3.3 Komprese dat

Komprese neboli komprimace dat je nedílnou součástí optimalizace dat, kde se používá tabulková komprese dat v několika následujících verzích:

- Základní tabulková komprese
    - Jedná se o základní metodu komprese dat, která je založena na slovníkovém algoritmu, používanou pouze na tabulky, které jsou pouze ke čtení.
- Rozšířená řádková komprese
    - Tato metoda komprese dat je známa již od verze Oracle Database 11g pod názvem OLTP Table Compression a v nové verzi 12c došlo k přejmenování na Advanced Row Compression. Stejně jako základní tabulková komprese je tato metoda založena na slovníkovém algoritmu, v němž jsou duplikované hodnoty nahrazeny zástupným symbolem, resp. odkazem do tabulky symbolů obsahující informace o zástupných symbolech pro překlad do nekomprimovaného stavu. Výhodou rozšířené řádkové komprese není pouze ušetření místa na disku, kdy v závislosti na datech dochází k úspoře 2x-4x, ale Oracle dokáže číst i přímo komprimovaná data, aniž by byla nutná dekomprimace, což má za následek zvýšení výkonu. Na rozdíl od

základní tabulkové komprese není rozšířená řádková komprese omezena pouze na tabulky určené ke čtení, ale dokáže komprimovat i zapisovaná data pomocí DML příkazu, jako je INSERT a UPDATE. [12]

- Hybridní sloupcová komprese
    - Hybridní sloupcová komprese neukládá data v řádkovém formátu jako předchozí metody, ale používá kombinaci řádkových a sloupcových metod. Výhodou sloupcové komprese je to, že data v daném sloupci mají stejný datový typ, a tím pádem společné vlastnosti. Pomocí této metody se velmi zvyšují možnosti úspory místa na úložném zařízení. K tomu používá hybridní sloupcová konstrukce speciální logickou konstrukci zvanou kompresní jednotka. Kompresní jednotka ukládá soubor řádků tabulky v sloupcovém formátu, kde všechny hodnoty daného sloupce jsou ukládány dohromady a následně je komprimován celý sloupec dle algoritmu v závislosti na datovém typu sloupce a dalších faktorech. [13]



Obrázek 7: Ukázka struktury kompresní jednotky obsahující tabulku o pěti sloupcích (Zdroj: [7])

- Hybridní sloupcová komprese může být dále rozdělena do čtyř úrovní: [4]
  - Data Warehouse Query Low a Query High
  - Archive Low a Archive High

  V závislosti na použití dané úrovně se úspora dat odhaduje 6x-15x. Nejvyšší úsporu úložného místa zaručuje úroveň Archive High sloužící, jak již název vypovídá, k archivaci starých neboli historických a nepoužívaných dat. [13]

1.4 Bezpečnost

V nové verzi Oracle Database 12c v oblasti zabezpečení je nejdůležitější novinkou funkce Oracle Data Redaction zajišťující maskování citlivých dat před neoprávněným přístupem uživatelů. K dalším novým funkcím patří například vylepšení kontroly komplexnosti hesel, přidání více rolí pro správu databáze nebo zobrazení časových údajů o posledním přihlášení uživatele. Informace o posledním přihlášení uživatele se ukládá do tabulky $USER a zobrazuje se např. při přihlášení uživatele pomocí databázového nástroje Oracle SQL*Plus. Tato novinka umožňuje bezpečnostním administrátorům mít přehled o poslední aktivitě uživatelů. [14]

1.4.1 Oracle Data Redaction

Oracle Data Redaction je nová funkce v oblasti zabezpečení patřící do modulu Oracle Advanced Security, který je součástí Oracle Database, tudíž není potřeba ho doinstalovat. Cílem Oracle Data Redaction je redigování neboli maskování celých či částí citlivých dat, jako jsou např. rodná čísla, čísla kreditních karet, bankovních účtů atd., která jsou dotazována neoprávněnými uživateli nebo aplikacemi. Samotné maskování dat probíhá v reálném čase poté, kdy uživatel vznese dotaz na konkrétní data. Data jsou redigována pouze na výstupu, tudíž nedochází k žádné úpravě uložených dat v databázi a databáze je s nimi schopna normálně pracovat.

Oracle Data Redaction je založeno na politikách, které vymezují podmínky a typ redigování pro maskování dat dané tabulky nebo pohledu. Vytváření politik pro

redigování dat umožňuje PL/SQL balík DBMS_REDACT, ke kterému administrátor potřebuje oprávnění EXECUTE pro balík DBMS_REDACT. [15]

Oracle Data Redaction podporuje několik základních typů redigování dat: [15]

- Full
    - Jedná se o defaultní typ redigování dat v případě, že administrátor v politice nespecifikuje konkrétní typ redigování. Oracle Database rediguje celý obsah specifického sloupce nebo tabulky tak, že např. místo rodného čísla, které bude považovat za znakový datový typ, se uživateli zobrazí pouze prázdné místo. Plné redigování je závislé na určitém datovém typu sloupce, kdy se místo znakových či textových datových typů zobrazí prázdné místo, na místo numerických datových typů bude na výstupu nula a v případě datového typu datum se zobrazí časový údaj 01-JAN-01.
- Partion
    - V případě částečného redigování dat databáze rediguje pouze část zobrazovaného výstupu v závislosti na bezpečnostní politice. Částečné redigování dat se nastavuje pomocí startovací pozice s indexováním od pozice jedna, počtu znaků k redigování a výběru samotného krycího znaku. V případě příkladu na rodném čísle 1122334444 se startovní pozicí 1, počtem krycích znaků 6 a krycím znakem * zobrazí Oracle databáze výsledek ve tvaru ******4444.
- Random
    - Náhodné redigování dat zobrazuje data ve výstupu pomocí náhodně vygenerovaných znaků či číslic v závislosti na datovém typu, jako tomu je u plného redigování dat. V případě rodného čísla 1122334444 by bylo nahrazeno náhodným číslem např. 8535857693. Pokud se uživatel bude dotazovat opakovaně na stejná data, pokaždé obdrží rozdílné náhodné hodnoty, což je založeno na principu redigování dat, protože k redigování dochází v reálném čase po uživatelově požadavku.

- Regular expressions
    - Regulární výrazy jsou speciální typ Oracle Data Redaction sloužící k vytváření vyhledávacích vzorů v řetězovitých datových typech pomocí standardizované syntaktické konvence.
- No redaction
    - Možnost „No redaction“ umožňuje testování Oracle Data Redaction politik před ostrým nasazením politik na tabulky.

1.4.2 Kontrola komplexnosti hesla

Kontrola komplexnosti hesla (Password Complexity Check) je funkcionalita zvyšující zabezpečení a ochranu databáze. Cílem této funkce je zapnutí již předdefinovaných politik, či vytvoření svojí politiky pro kontrolu uživatelských hesel při vytváření, upravování a resetování hesel, zda jsou nová hesla dostatečně silná a bezpečná. Tato funkce byla implementovaná již ve verzi Oracle Database 11g, ale v nové verzi Oracle Database 12c přibyly nové předdefinované verifikační funkce pro ověření složitosti hesla a možnost zapnutí kontroly komplexnosti hesla při vytváření nové databáze pomocí Oracle Database Configuration Assistant, jelikož ve výchozím stavu je tato funkcionalita vypnuta.

Oracle Database poskytuje tři předdefinované verifikační funkce pro kontrolu komplexnosti hesla. Tyto funkce jsou uloženy v PL/SQL skriptu utlpwdmg.sql umístěném v $ORACLE_HOME/rdbms/admin. Pro aktivování funkcionality pro kontrolu komplexnosti hesla je potřeba spustit již zmíněný skript, kdy dojde k vytvoření všech verifikačních funkcí ze skriptu ve schématu výchozího databázového administrátora SYS. Poté je potřeba přidat oprávnění EXECUTE na danou verifikační funkci všem uživatelům, u kterých je daná funkcionalita vyžadována, a v posledním kroku upravit defaultní uživatelský profil či konkrétní profily a nastavit danou verifikační funkci pro parametr `PASSWORD_VERIFY_FUNCTION`.

Jak již bylo zmíněno, kontrola komplexnosti hesla obsahuje tři předdefinované funkce - verify_function_11G, ora12c_verify_function a ora12c_strong_verify_function. Například heslo při zapnuté verifikační funkci

ora12c_strong_verify_function musí splňovat minimální délku o 9 znacích a musí obsahovat minimálně dva velké znaky, dva malé znaky, dvě číslice a dva speciální znaky a v případě změny hesla je nutné, aby se lišilo minimálně o čtyři znaky, než heslo předchozí. Popřípadě je možné funkce upravit dle svých parametrů ve skriptu utlpwdmg.sql a ten znovu spustit. [16]

1.5 Doplňující funkce

Tato podkapitola popisuje důležité novinky z oblasti vytváření či upravování struktury dat databáze.

1.5.1 Viditelnost sloupců

V Oracle Database 12c byla implementována nová funkcionalita umožňující zneviditelnění konkrétního sloupce tabulky. Sloupec se nezobrazí, pokud není explicitně napsán jeho název v SQL dotazu. Obdobně funguje i vkládání dat pomocí SQL příkazu `INSERT`, kde je taktéž nutno explicitně napsat název sloupce, jinak jsou hodnoty vkládány pouze do viditelných sloupců tabulky. Nezviditelňovat, resp. zviditelňovat sloupce tabulky lze kdykoliv za běhu databáze při vytváření či upravování tabulky. Principiálně je tato funkcionalita postavena na pořadí sloupců. Obvykle databáze ukládá a čísluje viditelné sloupce v pořadí, v jakém byly sloupce napsány v SQL příkazu `CREATE TABLE`. V případě přidání dalšího viditelného sloupce je sloupec zařazen na konec pořadí, zatímco neviditelné sloupce se vůbec nezařazují do pořadí sloupců, na kterém je založen výstup dotazu `SELECT * FROM` či `DESCRIBE`. Opětovné zviditelnění neviditelného sloupce znamená, že je taktéž zařazen na konec pořadí sloupců. Smyslem této funkcionality je skrýt požadované sloupce před koncovým uživatelem či samotnou aplikací při upravování tabulek a používání obecných SQL dotazů. [9]

1.5.2 Autoinkrementace

Uživatelé Oracle Database se s příchodem nové verze 12c konečně dočkali jednoduššího způsobu, jak automaticky inkrementovat identifikační hodnotu sloupce tabulky a nemuset používat kombinaci sekvencí a triggerů. Automatická inkrementace hodnot je implementována pomocí SQL klíčového slova `IDENTITY`

splňujícího standardizaci American National Standards Institute (ANSI), která umožňuje jednodušší DDL migraci z jiných databázových systémů.

Pro vytvoření autoinkrementační funkce na sloupci tabulky slouží klauzule `GENERATED AS IDENTITY`, která může být použita pouze na jednom sloupci tabulky a zároveň tento sloupec musí být numerického datového typu. Klauzule `GENERATED AS IDENTITY` ve své podstatě vytváří sekvenci a lze ji nastavit stejné parametry jako při práci se sekvencemi. Zároveň lze nastavit typ funkce, kdy se má vygenerovaná hodnota použít. Ve výchozím stavu je použit typ `ALWAYS`, kdy se generuje nová hodnota vždy a nelze ji explicitně v příkazu `INSERT` změnit, jelikož by došlo k vyvolání chyby. K dalším typům patří `BY DEFAULT` a `BY DEFAULT ON NULL`. V případě typu `BY DEFAULT` dojde k vytvoření vygenerované hodnoty vždy, když není explicitně napsána v příkazu INSERT a v případě `DEFAULT ON NULL` je hodnota vygenerována pouze, když je v příkazu INSERT explicitně napsána hodnota NULL. [16]

2 Praktická část

Praktická část práce se bude zabývat názornými příklady vztahujícími se k funkcionalitám popisovaným v teoretické části práce. Z oblasti multitenantní architektury budou znázorněny příklady pro vytvoření CDB databáze, různé techniky vytvoření PDB databáze a práce se společnými, resp. místními uživateli a rolemi. V dalších částech budou zobrazeny příklady pro správu dat, z oblasti zabezpečení budou ukázány příklady redigování dat a kontroly komplexnosti hesla a v závěru doplňující příklady pro viditelnost sloupců a auto inkrementaci. Praktické příklady budou ukázány pomocí dotazovacího jazyka SQL a procedurálního jazyka PL/SQL, kdy tyto jazyky lze využít v prostředí SQL*Plus nebo Oracle SQL Developer.

2.1 Nástroje pro správu Oracle Database

Pro práci s multitenantní architekturou můžeme použít stejné nástroje jako pro non-CDB databázi a to jsou: [7]

- SQL*Plus
    - Jedná se o interaktivní a dávkovací dotazovací nástroj podobný příkazovému řádku z prostředí Windows. Příkazový řádek pro Oracle databáze, který je obsažen v každé verzi Oracle Database, umožňuje řídit správu a vývoj databáze pomocí SQL dotazů, PL/SQL a vlastních interních příkazů.
- SQL Developer
    - SQL Developer je grafická verze SQL*Plus podporující SQL a PL/SQL pro správu databází. SQL Developer je dostupný při defaultní instalaci Oracle Database nebo je popřípadě zdarma ke stažení.
- Oracle Enterprise Manager Database Express 12c
    - Jedná se o webové rozhraní nevyžadující instalaci pro základní správu a monitorování Oracle Database.

- Oracle Database Configuration Assistant (DBCA)
    - DBCA je pomocný nástroj poskytující grafické prostředí pro vytváření a konfiguraci databází podporujících multitenantní architekturu.

2.2 Práce s multitenantní architekturou

K základním úlohám pro práci s novou multitenantní architekturou patří vytváření a správa CDB a PDB databází spolu se společnými i lokálními uživateli.

2.2.1 Vytvoření CDB

Pro vytvoření kontejnerové databáze je uváděn dotaz v jazyce SQL `CREATE DATABASE název ENABLE PLUGGABLE DATABASE`. Po spuštění dotazu bude vytvořena kořenová databáze (PDB$ROOT) a automaticky dojde k vytvoření semínka (PDB$SEED). V případě vynechání části za názvem `ENABLE PLUGGABLE DATABASE` dojde k vytvoření k non-CDB, z čehož vyplývá, jak již bylo zmíněno výše, že databáze nebude moci obsahovat PDB databáze. [7]

```
1 CREATE DATABASE nazevCDB ENABLE PLUGGABLE DATABASE
```

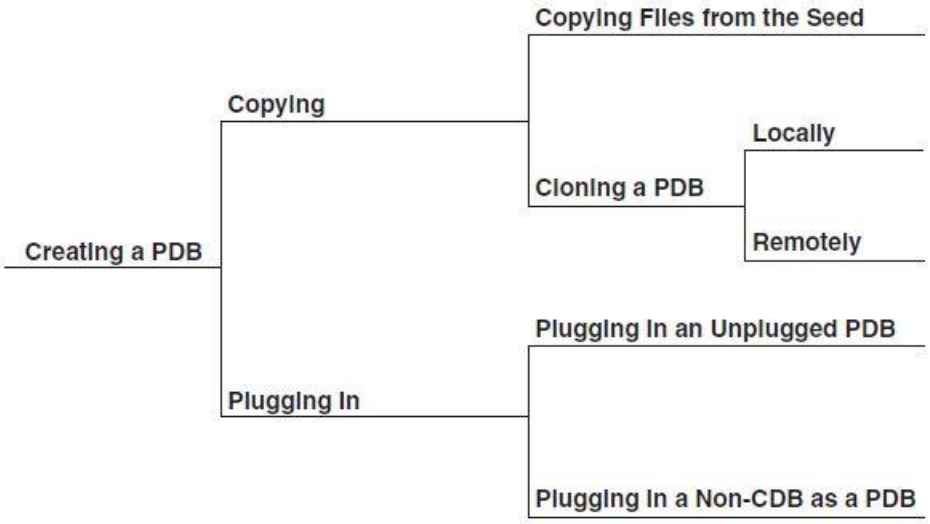
Příklad 1: Vytvoření CDB databáze

Jedná se pouze o teoretickou ukázku příkazu, kterou nelze takto jednoduše aplikovat. Pro vytvoření celé databáze s podporou multitenantní architektury je nutno rozšířit tento příkaz o mnoho parametrů. Proto pro vytvoření a konfiguraci databáze je mnohem efektivnější využití nástroje Oracle Database Configuration Assistant s možností vytvoření databáze s výchozí jednodušší konfigurací nebo v režimu rozšířené konfigurace umožňující například pouhé vygenerování skriptu s příkazem `CREATE DATABASE` do souboru.

2.2.2 Vytvoření PDB

O všech možnostech vytvoření PDB databází vypovídá následující obrázek 8. K základním metodám vytvoření PDB databáze v CDB databázi patří připojení odpojené PDB databáze např. z jiné CDB databáze, nebo připojení a vytvoření PDB databáze z non-CDB databáze pomocí více druhů technik. V případě vytváření PDB

databáze kopírováním odlišujeme pouze, zda se jedná o klon již existující PDB databáze, kterou lze klonovat lokálně i vzdáleně, nebo o vytváření prázdné PDB databáze kopírováním semínka PDB$SEED, které je součástí každého CDB kontejneru.



Obrázek 8: Možnosti vytvoření PDB databází (Zdroj: [7])

2.2.2.1 Podmínky pro vytvoření PDB

Samozřejmostí je již mít vytvořenou kontejnerovou databázi (CDB), která musí být v čtecím/zapisovacím módu. Z uživatelského hlediska přihlášený uživatel musí být z kategorie společných uživatelů (viz. kapitola 1.2.7.1) a musí mít systémové oprávnění pro vytvoření PDB databáze. Administrátor taktéž musí zvolit vhodný název databáze. Každý název musí začínat písmenem a musí být unikátní v rámci CDB kontejneru. [9]

2.2.2.2 Vytvoření PDB ze semínka

Prvním způsobem vytvoření PDB je použití dotazu `CREATE PLUGGABLE DATABASE`, kdy dojde k vytvořením PDB kopírováním souborů ze semínka PDB$SEED. Základní verze tohoto příkazu je ještě specifická tím, že je nutno spolu s vytvořením databáze vytvořit i jejího lokálního administrátora.

Základní postup vytvoření databáze ze semínka:

1) Přihlášení k databázi uživatelem s oprávněním pro vytvoření PDB databáze.

2) Je potřeba zjistit, zda se uživatel nalézá v kořenu CDB databáze, odkud je možné PDB databázi vytvořit. Jestliže uživatel pracuje v prostředí SQL*Plus, lze tento dotaz nahradit interním příkazem `SHOW CON_NAME`.

    a. V případě, že tomu tak není, je potřeba se přemístit do kořenu CDB databáze úpravou relace.

3) Samotný základní příkaz pro vytvoření PDB databáze ze semínka.

4) SQL dotaz pro získání informací o všech PDB databázích v CDB databázi z pohledu v$pdbs. Výstup zobrazuje identifikační číslo, název a informace o režimu databáze ze sloupce `OPEN_MODE`. Po vytvoření PDB databáze se databáze nachází ve stavu `MOUNTED`, který umožňuje přístup k databázi pouze administrátorům a neumožňuje čtení ani zapisování dat. K dalším stavům patří `READ ONLY` umožňující pouze čtení z databáze, `READ/WRITE` umožňující čtení i zápis a stav `UPGRADE` umožňující spouštění skriptů pro aktualizování databází. Poslední tři zmiňované stavy mohou mít navíc omezení `RESTRICTED` umožňující pouze přístup uživatelům, kteří mají přidělené oprávnění `RESTRICTED SESSION`. [9] Jestliže uživatel pracuje v prostředí SQL*Plus, lze tento dotaz nahradit interním příkazem `SHOW PDBS`.

5) Příkaz `OPEN` je totožný s `OPEN READ WRITE` a znamená tedy, že databáze přejde do stavu `READ/WRITE` umožňujícího čtení a zápis.

```
1  CONNECT sys /as sysdba
2  SELECT sys_context('USERENV', 'CON_NAME') FROM DUAL;
2a ALTER SESSION SET CONTAINER = CDB$ROOT;
3  CREATE PLUGGABLE DATABASE nazevDatabaze ADMIN USER
   nazevAdministratora IDENTIFIED BY hesloAdministratora;
4  SELECT name, con_id, open_mode, restricted FROM v$pdbs;
5  ALTER PLUGGABLE DATABASE nazevDatabaze OPEN;
```

Příklad 2: Postup vytvoření PDB databáze ze semínka

2.2.2.3 Vytvoření PDB klonováním

Přístup vytvoření PDB databáze klonováním vystihují autoři Ashdown a Kyte: „Technika kopírování souborů zdrojové PDB do nového umístění a asociování těchto souborů s novou PDB.“ (přeloženo autorem). [7] Jak již bylo prezentováno výše na obrázku 8, klonovat můžeme PDB databázi z aktuální otevřené lokální CDB nebo vzdálené CDB pomocí databázový odkazů.

Základní postup klonování lokální PDB databáze:

1) V prvním kroku je potřeba se přemístit do kořenu CDB databáze úpravou relace.
2) Druhý řádek znázorňuje přepnutí zdrojové PDB databáze pouze do režimu čtení. V případě, že se databáze nachází v `READ/WRITE` režimu, je potřeba ji zavřít a následně otevřít v `READ ONLY` režimu.
3) Třetí řádek příkladu znázorňuje samotný SQL příkaz pro kopírování lokální zdrojové PDB databáze *nazevDB1* do nové PDB databáze *NazevKopieDB*.
4) Kontrolní SQL dotaz pro získání informací o všech PDB databází v CDB databázi z pohledu v$pdbs.
5) Ve finálním kroku je potřeba už pouze zdrojovou i cílovou databázi uvést do `READ/WRITE` režimu.

```
1 ALTER SESSION SET CONTAINER = CDB$ROOT;
2 ALTER PLUGGABLE DATABASE nazevDB1 OPEN READ ONLY;
3 CREATE PLUGGABLE DATABASE NazevKopieDB FROM nazevDB1;
4 SELECT name, con_id, open_mode, restricted FROM
  v$pdbs;
5 ALTER PLUGGABLE DATABASE nazevDB1 CLOSE;
  ALTER PLUGGABLE DATABASE nazevDB1 OPEN;
  ALTER PLUGGABLE DATABASE nazevKopieDB OPEN;
```

Příklad 3: Klonování lokální PDB databáze

Dalším možným způsobem klonování PDB databáze je využití vzdáleného přístupu k odlišné CDB databázi pomocí databázového odkazu, kdy je potřeba pracovat v obou CDB databázích. Tento způsob klonování vzdálené PDB databáze funguje až od verze 12.1.0.2. Předchozí verze obsahuje chybu neumožňující klonování databáze pomocí databázového odkazu.

Základní postup klonování vzdálené PDB databáze:

1) Ve zdrojové databázi je potřeba nastavit kořenový kontejner a následně režim přístupu zdrojové PDB databáze, která bude klonována, a provést kontrolu stavu.

2) Zároveň je potřeba vytvořit společného nebo lokálního uživatele ve zdrojové PDB s oprávněním `CREATE PLUGGABLE DATABASE`. V příkladu je vytvořen uživatel s prefixem C## vyjadřujícím, že se jedná o společného uživatele. Následně je uživateli přidělena administrátorská role.

3) Ve třetím kroku dochází k nastavení kořenového kontejneru v cílové databázi.

4) V následujícím kroku je potřeba vytvořit databázový odkaz pro připojení se ke zdrojové databázi. Samostatný databázový odkaz se skládá z názvu, z uživatele umožňujícího připojení ke vzdálené databázi a klauzule USING, která obsahuje připojovací řetězec obsahující informace o použitém protokolu, IP adrese nebo doménovém jméně, portu, na kterém naslouchá, a názvu služby[2] databáze, ke které je potřeba se připojit, v tomto případě k CDB databázi.

5) Příkaz pro vytvoření nové databáze *nazevKopie2* ze vzdálené databáze *nazevDB1*, ke které se přistupuje pomocí databázového odkazu *testLink*.

6) Finální krok provádí uvedení nové databáze do režimu `READ/WRITE`.

[2] Jméno služby (Service name) slouží k logické reprezentaci služby, přes které probíhá připojení ke konkrétní databázi. [4]

1z	ALTER SESSION SET CONTAINER = CDB$ROOT; ALTER PLUGGABLE DATABASE *nazevDB1* CLOSE; ALTER PLUGGABLE DATABASE *nazevDB1* OPEN READ ONLY; SELECT name, con_id, open_mode, restricted FROM v$pdbs;
2z	CREATE USER *C##LINK_USER* IDENTIFIED BY Password66; GRANT "DBA" TO "*C##LINK_USER*" CONTAINER=ALL;
3c	ALTER SESSION SET CONTAINER = CDB$ROOT;
4c	CREATE PUBLIC DATABASE LINK *testLink* CONNECT TO *C##LINK_USER* IDENTIFIED BY *Password66* USING '(DESCRIPTION= (ADDRESS=(PROTOCOL=TCP)(HOST=10.10.10.1)(PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=ORACDB)))';
5c	CREATE PLUGGABLE DATABASE *nazevKopie2* FROM *nazevDB1@testLink*;
6c	ALTER PLUGGABLE DATABASE *nazevKopie1* OPEN;

Příklad 4: Klonování vzdálené databáze, z - zdrojová databáze, c – cílová databáze

2.2.2.4 Smazání PDB databáze

Smazání PDB databáze je triviální příklad pro odstranění PDB databáze z CDB databáze v případě přesunutí nebo postradatelnosti. PDB databáze musí být před smazáním uzavřena nebo rovnou odpojena. Další podmínkou pro odstranění PDB databáze je uživatel s administrativním oprávněním SYSDBA nebo SYSOPER. Samozřejmostí je, že tato operace se opět musí odehrávat v kořenovém kontejneru CDB databáze. Samotné smazání je založeno na SQL příkazu `DROP PLUGGABLE DATABASE`, který může obsahovat ještě jeden ze dvou klauzulí – `INCLUDING DATAFILES` nebo `KEEP DATAFILES`. První zmiňovaný odstraní i všechny datové soubory PDB databáze, zatímco druhý zmiňovaný odstraní pouze dočasné soubory a záznam z CDB databáze, ale zanechá v datovém uložišti ostatní datové soubory, které je popřípadě nutno smazat manuálně v prostředí operačního systému. [9]

```
1  ALTER SESSION SET CONTAINER = CDB$ROOT;
2  ALTER PLUGGABLE DATABASE nazevPDB CLOSE;
3a DROP PLUGGABLE DATABASE nazevPDB INCLUDING DATAFILES;
3b DROP PLUGGABLE DATABASE nazevPDB KEEP DATAFILES;
```

Příklad 5: Smazání PDB databáze

2.2.2.5 Připojení a odpojení PDB

Připojení již existující odpojené PDB databáze či odpojení PDB databáze probíhá pomocí XML souboru, který obsahuje metadata popisující PDB databázi a její soubory potřebné k připojení k CDB databázi. K vytvoření XML souboru a zároveň odpojení databáze dochází při provedení následujícího SQL dotazu `ALTER PLUGGABLE DATABASE nazevPDB UNPLUG INTO '/cesta/k/souboru/nazevPDB.xml';` [9]

Postup práce pro odpojení PDB databáze s vysvětlením:

1) V prvním kroku je opět důležité, aby jako pracovní kontejner byl nastaven `CDB$ROOT`.

2) Uzavření přístupu k databázi.

3) Samotný příkaz k odpojení databáze s cílovou adresou pro uložení XML souboru obsahujícího metadata o databázi. V tomto případě je cílovou adresou složka s datovými soubory dané PDB databáze *pdbplug*.

4) Po zadání SQL dotazu pro získání informací o PDB databázích se záznam o PDB databázi *pdbplug* nachází stále v CDB kontejneru, ačkoliv došlo k jejímu odpojení a nelze s ní dále pracovat. Proto je v 5. kroku potřeba danou databázi ještě manuálně smazat. Dojde ovšem pouze ke smazání záznamu z CDB databáze, datové soubory odpojené PDB databáze zůstanou zachovány a lze je přemístit.

```
1  ALTER SESSION SET CONTAINER = CDB$ROOT;

2  ALTER PLUGGABLE DATABASE pdbplug CLOSE;

3  ALTER PLUGGABLE DATABASE pdbplug UNPLUG INTO
   'c:\app\ORA1\oradata\ORA1CDB\PDBPLUG\pdbplug.xml';

4  SELECT con_id, guid, name, open_mode FROM v$pdbs;

   CON_ID  GUID                             NAME      OPEN_MODE
   ------- -------------------------------- --------- ----------
         2 CD1E093738834E8980029E159B3B2E72 PDB$SEED  READ ONLY
         3 3C4C72BB0F044C17904D835426131D9D ORA1PDB   READ WRITE
         4 60D6A9299A2B41DDB476B715AC9A92B9 PDBPLUG   MOUNTED

5  DROP PLUGGABLE DATABASE pdbplug;
```

Příklad 6: Odpojení PDB databáze z CDB kontejneru

Jak již bylo poukázáno na obrázku 8 (viz. kapitola 2.2.2), Oracle Database bere připojení PDB databáze jako vytvoření nové PDB databáze s využitím SQL příkazu `CREATE PLUGGABLE DATABASE` s dodatečnou klauzulí `USING`, kde je specifikována cesta k XML souboru s informacemi o odpojené PDB databázi.

Následující příklad popisuje situaci opětovného připojení stejné odpojené PDB databáze. Příklad navazuje na předchozí příklad 6, kde došlo k odpojení a smazání databáze z CDB, ale XML soubor a datové soubory databáze zůstaly stále ve svých adresářích. Jelikož soubory nebyly přemístěny a stále jsou umístěny v lokaci, na kterou odkazují údaje z XML souboru, není třeba v příkazu `CREATE PLUGGABLE DATABASE` stanovovat změnu cesty, přejmenovávat soubory ani je znovu kopírovat. Pouze se připojí k příkazu klauzule `TEMPFILE REUSE`, která sděluje již dle názvu, že bude použit původní dočasný soubor (temp file).

```
1 ALTER SESSION SET CONTAINER = CDB$ROOT;

2 CREATE PLUGGABLE DATABASE pdbplug USING
  'c:\app\ORA1\oradata\ORA1CDB\PDBPLUG\pdbplug.xml'
  NOCOPY
  TEMPFILE REUSE;

3 ALTER PLUGGABLE DATABASE pdbplug OPEN;
```

Příklad 7: Připojení PDB databáze do stejné lokace, ze které byla odpojena

Další příklad už poukazuje na připojení PDB databáze do odlišné CDB databáze, než ze které byla odpojena. Tento příklad opět bude navazovat na příklad 6. S tím rozdílem, že XML soubor a datové soubory jsou přesunuty na cílový server, kde je nainstalován Oracle Database 12c. SQL příkaz `CREATE PLUGGABLE DATABASE` je rozšířen o klauzule `SOURCE_FILE_NAME_CONVERT` a `FILE_NAME_CONVERT`. První zmiňovaná klauzule `SOURCE_FILE_NAME_CONVERT` slouží ke změně aktuální lokace datových souborů PDB databáze, jelikož XML soubor v sobě obsahuje lokace z původní CDB databáze. Zatímco druhá klauzule `FILE_NAME_CONVERT` slouží k určení adresáře, kde jsou uloženy ostatní PDB databáze, což souvisí s klauzulí `COPY`, která určuje, že se datové soubory mají zkopírovat ze zdrojového adresáře `d:\pdbplug\` do `d:\app\AdminxDx\oradata\orclDB\pdbplug\`. V případě manuálního nakopírování zdrojových datových souborů do adresáře pro PDB databáze `d:\app\AdminxDx\oradata\orclDB\pdbplug\` lze vynechat klauzuli `FILE_NAME_CONVERT` a klauzuli `COPY` změnit na `NOCOPY`.

```
1 ALTER SESSION SET CONTAINER = CDB$ROOT;

2 CREATE PLUGGABLE DATABASE pdbplug USING
  'd:\PDBPLUG\pdbplug.xml'
  SOURCE_FILE_NAME_CONVERT = (
  'c:\app\ORA1\oradata\ORA1CDB\pdbplug\',
  'd:\pdbplug\')
  COPY
  TEMPFILE REUSE
  FILE_NAME_CONVERT = (
  'd:\pdbplug\',
  'd:\app\AdminxDx\oradata\orclDB\pdbplug\');

3 ALTER PLUGGABLE DATABASE pdbplug OPEN;
```

Příklad 8: Připojení PDB databáze do odlišné CDB databáze

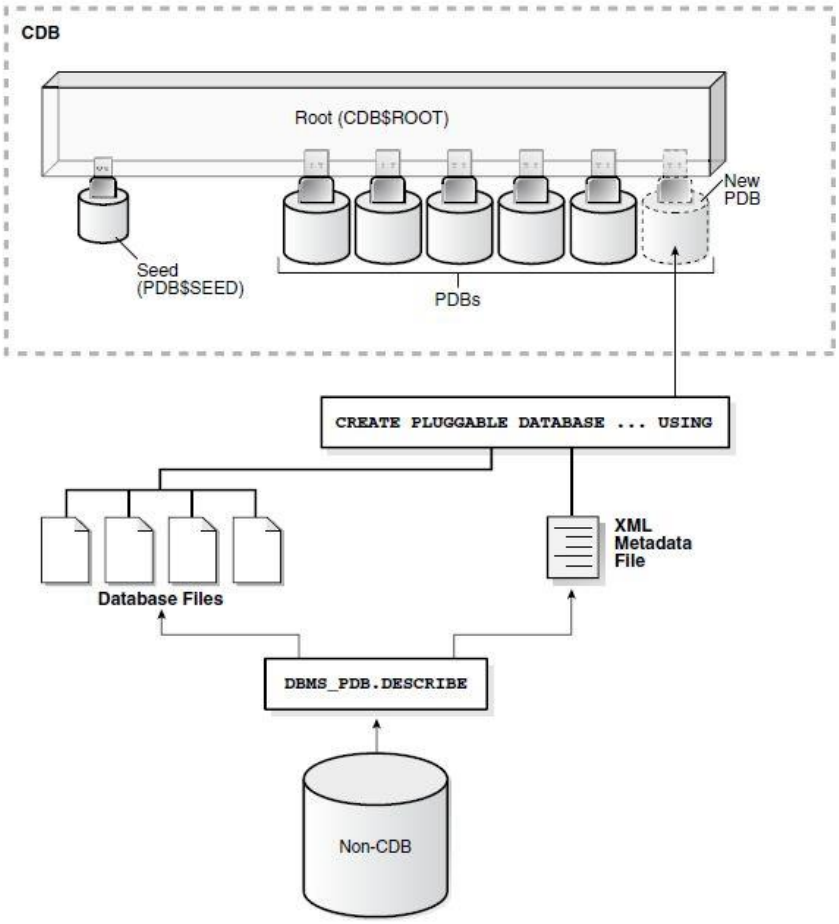
V případě, že PDB databáze je již připojena a uživatel by požadoval opětovné duplikátní připojení PDB databáze, lze předchozí příklad obměnit a přidat před klauzuli `USING` další klauzuli `AS CLONE`. Bez klausule `AS CLONE` by SQL příkaz `CREATE PLUGGABLE DATABASE` skončil chybovým hlášením o duplikátním globálním identifikátoru (GUID) a dalších unikátních identifikátorech PDB databáze.

```
1 CREATE PLUGGABLE DATABASE pdbplugClone AS CLONE USING
  'd:\PDBPLUG\pdbplug.xml'
  SOURCE_FILE_NAME_CONVERT = (
  'c:\app\ORA1\oradata\ORA1CDB\pdbplug\',
  'd:\pdbplug\')
  COPY
  TEMPFILE REUSE
  FILE_NAME_CONVERT = (
  'd:\pdbplug\',
  'd:\app\AdminxDx\oradata\orclDB\pdbplugClone\');
```

Příklad 9: Duplikátní připojení PDB databáze

2.2.2.6 Vytvoření PDB z non-CDB

V případě vytvoření PDB z non-CDB existuje více technik. Jednou z nich je opět práce s konfiguračním XML souborem a balíkem DBMS_PDB, který obsahuje procedury `DESCRIBE` a `CHECK_PLUG_COMPATIBILITY`.



Obrázek 9: Diagram znázorňující techniku s využitím procedury DBMS_PDB.DESCRIBE (Zdroj: [4])

Procedura `DESCRIBE` vytváří konfigurační XML soubor s informacemi o non-CDB databázi, stejně jako tomu konala klauzule z předchozích příkladů `UNPLUG INTO` v případě PDB databází. Procedura `CHECK_PLUG_COMPATIBILITY` se spouští na cílové CDB databázi pro zjištění, jak napovídá název procedury, kompatibility zdrojové non-CDB databáze s cílovou CDB databází. V případě kladného výstupu procedury `CHECK_PLUG_COMPATIBILITY` lze vytvořit PDB databázi z non-CDB databáze pomocí již zmiňovaného SQL příkazu `CREATE PLUGGABLE DATABASE ... USING ....` Po vytvoření PDB databáze je nutno ještě spustit SQL skript `noncdb_to_pdb`, který odstraňuje nadbytečná data z datového slovníku nové

PDB databáze, jelikož ten obsahuje nepotřebná data z původní non-CDB databáze. Tento postup je navíc podmíněn verzí non-CDB databáze, která musí být aktualizována pro Oracle Database 12c. [9]

Následující příklady 10a a 11b ukazují vytvoření PDB databáze z non-CDB databáze pomocí techniky využívající balík DBMS_PDB. Zdrojová non-CDB databáze i cílová databáze jsou verze Oracle Database 12c (12.1.0.2).

Příklad 10a značí postup ve zdrojové non-CDB databázi:

1) V prvním kroku je potřeba, aby se databáze před vytvářením XML souboru nacházela v konzistentním stavu a nedošlo k žádným změnám. Proto je potřeba non-CDB databázi zastavit a v následujícím kroku otevřít pouze ve čtecím režimu.

2) Zmiňovaný krok pro otevření databáze pouze v režimu pro čtení.

3) Na třetím řádku lze nalézt PL/SQL skript s procedurou `DESCRIBE` balíku DBMS_PDB a jejím jediným parametrem určujícím cílový adresář a pojmenování XML souboru. Po spuštění skriptu dojde k vygenerování XML souboru non-CDB databáze a posléze lze databázi fyzicky přesunout do cílové CDB databáze.

```
1 SHUTDOWN IMMEDIATE;
2 STARTUP OPEN READ ONLY;
3 BEGIN
  DBMS_PDB.DESCRIBE( pdb_descr_file =>
  'c:\app\ora1\oradata\ora1noncdb\ora1noncdb.xml');
  END;
```

Příklad 10a: Vytvoření XML souboru v non-CDB databázi

Příklad 11b značí postup v cílové CDB databázi:

1) Nastavení kořenového kontejneru v CDB databázi.

2) Druhý krok příkladu je volitelný, avšak je doporučeno ho použít. Jedná se o PL/SQL skript využívající balík DMBS_PDB s procedurou `CHECK_PLUG_COMPATIBILITY` s parametry pro určení adresáře, kde se nachází XML soubor non-CDB databáze a určení jejího budoucího názvu. V případě že non-CDB databáze je kompatibilní, vrátí tento PL/SQL skript hodnotu `TRUE` a lze pokračovat krokem č. 3. V opačném případě lze pokračovat pouze krokem 2a. Zdroj skriptu: [9]

    a. Jestliže non-CDB databáze není kompatibilní, krok 2a je konečným krokem. Z pohledu `PDB_PLUG_IN_VIOLATIONS` lze zjistit důvody nekompatibility.

3) Po odsouhlasení kompatibility lze vytvořit PDB databázi pomocí SQL příkazu `CREATE PLUGGABLE DATABASE` obdobně jako v předchozích příkladech.

4) Po vytvoření PDB databáze z non-CDB databáze s ní nelze okamžitě pracovat. Je potřeba ještě spustit SQL skript `noncdb_to_pdb` pro transformaci datového slovníku a odstranění přebytečných dat z něj. Pro spuštění je nutno pracovat v kontejneru PDB databáze.

5) Spuštění SQL skriptu `noncdb_to_pdb`.

6) Zobrazení informací o nové PDB databázi.

1
```
ALTER SESSION SET CONTAINER = CDB$ROOT;
```

2
```
SET SERVEROUTPUT ON
DECLARE
compatible CONSTANT VARCHAR2(3) :=
CASE DBMS_PDB.CHECK_PLUG_COMPATIBILITY(
pdb_descr_file =>
'c:\app\ORA1\oradata\ORA1CDB\ora1noncdb\ora1noncdb.xml',
pdb_name => 'ORANONPDB')
       WHEN TRUE THEN 'YES'
       ELSE 'NO'
END;
BEGIN
     DBMS_OUTPUT.PUT_LINE(compatible);
END;/
```

2a
```
SELECT name, cause, type, message, status FROM
PDB_PLUG_IN_VIOLATIONS WHERE name='ORANONPDB';
```

3
```
CREATE PLUGGABLE DATABASE oranonpdb USING
'c:\app\ORA1\oradata\ORA1CDB\ora1noncdb\ora1noncdb.xml'
SOURCE_FILE_NAME_CONVERT = (
'c:\app\ORA1\oradata\ORA1NONCDB\',
'c:\app\ORA1\oradata\ORA1CDB\'ora1noncdb\)
NOCOPY
TEMPFILE REUSE;
```

4
```
ALTER SESSION SET CONTAINER=oranonpdb;
```

5
```
@c:\app\ORA1\product\12.1.0\dbhome_1\rdbms\admin\noncdb_to_p
db.sql
```

6
```
SELECT con_id, name, open_mode FROM v$pdbs;

CON_ID  NAME          OPEN_MODE
------- ------------- ----------
      4 ORANONPDB     READ WRITE
```

Příklad 11b: Vytvoření PDB databáze z non-CDB databáze

Další možností pro vytvoření PDB z non-CDB databáze je využití nástroje Oracle Data Pump pro export a import dat. V non-CDB databázi provede Oracle Data Pump export dat, metadat a kontrolních informací do binárních souborů. Princip spočívá ve vytvoření prázdné PDB databáze a následném importování dat vyexportovaných souborů z původní non-CDB databáze. [7]

2.2.3 Uživatelé a role

Důležitou součástí multitenantní architektury je práce s uživateli a rolemi, kdy obě tyto skupiny jsou rozčleněny na společné a lokální uživatele, resp. role. Společného i lokálního uživatele lze vytvořit pomocí klasického SQL příkazu `CREATE USER`, u kterého, nepočítá-li se potřebné oprávnění, záleží pouze, v jakém kontejneru je daný příkaz spouštěn. Společný uživatel může být vytvořen pouze v kořenovém kontejneru (`CDB$ROOT`) CDB databáze a jméno uživatele musí začínat prefixem c prefixem c## nebo C##. U lokálního uživatele pouze záleží na nastaveném aktuálním kontejneru, resp. k jaké PDB databázi je uživatel s daným oprávněním připojen. [7]

Vytvoření společné nebo lokální role je principiálně stejné jako vytvoření uživatele, pouze dochází k obměně SQL příkazu `CREATE ROLE`.

Pro následující příklady je vytvořen modelový příklad obsahující tři PDB databáze vytvořené pomocí techniky kopírování semínka, při které dochází k vytvoření lokálního administrátora nové PDB databáze přidělením lokální role PDB_DBA. Dle výpisu má tato role pouze tři systémová oprávnění – `CREATE SESSION`, `SET CONTAINER` a `CREATE PLUGGABLE DATABASE`.

Tabulka 3: Zadání modelové situace pro příklady s uživateli a rolemi

CON_ID	NAME	ADMIN USER
2	PDB$SEED	-
3	ORA01PDB	A01PDB
4	ORA02PDB	A02PDB
5	ORA03PDB	A03PDB

Příklad 12 znázorňuje vytvoření společného uživatele *c##cdbAdmin*, kterému bude přidělena společná role DBA. Hlavní pointa tohoto příkladu je přidělení společné role s platností na všechny kontejnery, a tím pádem uživateli umožňuje práci ve všech databázích:

1) Jelikož je vytvářen společný uživatel, je potřeba pracovat v kořenovém kontejneru CDB databáze.

2) Vytvoření účtu společného uživatele *c##cdbAdmin* s heslem *Password12*, kterému je nastaven konec platnosti díky klauzuli `PASSWORD EXPIRE`. Uživatel bude po prvním přihlášení požádán ke změně hesla. Další klauzulí použitou v příkazu je nastavení výchozího tabulkového prostoru, kdy tento tabulkový prostor musí být vytvořen ve všech PDB databázích dané CDB databáze. Dodatečně příkaz obsahuje ještě klauzuli `ACCOUNT UNLOCK` vyjadřující, že účet je odemčen, a klauzuli `CONTAINER` stanovující vytvoření účtu ve všech kontejnerech.

3) Přidělení společné role DBA s platností na všechny kontejnery.

```
1  ALTER SESSION SET CONTAINER = CDB$ROOT;

   CREATE USER c##cdbAdmin IDENTIFIED BY Password12
   DEFAULT TABLESPACE users
2  PASSWORD EXPIRE
   ACCOUNT UNLOCK
   CONTAINER=ALL;

3  GRANT DBA to c##cdbAdmin CONTAINER=ALL;
```

Příklad 12: Vytvoření společného uživatele s přiřazením společné role

Následující příklad 13 znázorňuje taktéž vytvoření společného uživatele s tím rozdílem, že bude omezen pouze na PDB databáze *ORA01PDB* a *ORA02PDB*. Toho lze docílit přiřazením pouze lokálních DBA rolí v těchto databázích.

```
1  ALTER SESSION SET CONTAINER = CDB$ROOT;
2  CREATE USER c##PdbAdmin0102 IDENTIFIED BY Password12
   DEFAULT TABLESPACE users
   PASSWORD EXPIRE
   ACCOUNT UNLOCK
   CONTAINER=ALL;
3  ALTER SESSION SET CONTAINER = ORA01PDB;
4  GRANT DBA TO c##PdbAdmin0102 CONTAINER=CURRENT;
5  ALTER SESSION SET CONTAINER = ORA02PDB;
6  GRANT DBA TO c##PdbAdmin0102 CONTAINER=CURRENT;
```

Příklad 13: Vytvoření společného uživatele s přiřazením lokální role

Další příklad slouží k vytvoření společné role s oprávněními pro připojení k databázi a vytváření tabulek. Tato společná role bude posléze v následujícím příkladu 15 lokálně přidělena lokálnímu uživateli v databázi *ORA03PDB.* V závěru teda databáze *ORA03PDB* získá nového lokálního uživatele *TableManager*, kterému je lokálně přiřazena společná role umožňující připojení k této PDB databázi a vytváření tabulek v jeho uživatelském schématu.

```
1  ALTER SESSION SET CONTAINER = CDB$ROOT;
2  CREATE ROLE c##TableManage CONTAINER=ALL;
3  GRANT CREATE SESSION, CREATE TABLE TO c##TableManage
   CONTAINER=ALL;
```

Příklad 14: Vytvoření společné role

```
1  ALTER SESSION SET CONTAINER = ORA03PDB;

2  CREATE USER TableManager IDENTIFIED BY Password12
   DEFAULT TABLESPACE users
   ACCOUNT UNLOCK
   CONTAINER=CURRENT;

3  GRANT c##TableManage TO TableManager
   CONTAINER=CURRENT;
```

Příklad 15: Vytvoření lokálního uživatele s přiřazením společné role

2.3 Správa dat

Z oblasti správy dat tato kapitola představí komplexní příklad pro zapnutí tepelné mapy a automatické optimalizace dat spolu s jejich nastavením a správou. Oracle Database 12c bohužel nepodporuje automatickou optimalizaci dat v multitenantní architektuře, proto je potřeba se přemístit do non-CDB databáze.

1) Funkcionalita tepelné mapy a automatické optimalizace dat spolu úzce souvisí a jejich zapnutí je shrnuto do jednoho společného systémového parametru `heat_map`. Ve výchozím stavu by tyto funkcionality měly být vypnuty. Příkaz `SHOW` zobrazuje hodnotu parametru `heat_map`.

2) Zapnutí tepelné mapy a automatické optimalizace dat. Informace o monitorování dat lze získat pomocí dotazů na následující pohledy - `v$heat_map_segment`, `all_heat_map_segment`, `all_heat_map_seg_histogram`, `dba_heatmap_top_tablespaces` a `dba_heatmap_top_objects`.

3) Pro nastavení automatické optimalizace dat je potřeba vytvořit politiky životního cyklu dat na daných tabulkách. V tomto příkladu je využita tabulka `EMPLOYEES` uživatele `HR`. `HR` schéma je testovací schéma dodávané společností Oracle. V tomto kroku je tedy nastavena první politika pro tabulku `EMPLOYEES`, která zkomprimuje data na úrovni řádků komprimační metodou Advanced row po deseti dnech, kdy nebudou nijak modifikovány.

4) Přidání další politiky na tutéž tabulky, která opět komprimuje data metodou Advanced row ale na úrovni segmentů po třiceti dnech, kdy nebudou nijak modifikována.

5) Na danou tabulku je přidána ještě poslední politika vyjadřující nejvyšší druh komprimace Archive High v případě, že k datům nebude vůbec přistoupeno po dobu 6 měsíců neboli 180 dní.

6) Dotaz pro zobrazení přehledu o daných politikách. Další informace lze získat z pohledů začínajících prefixem `dba_ilm` nebo `user_ilm`.

7) Dodatkové možnosti pro smazání nebo vypnutí všech ADO politik. Popřípadě lze vymazat či vypnout jednotlivé ADO politiky.

1
```
show parameter heat_map;
NAME         TYPE    VALUE
------------ ------- ------
heat_map     string  OFF
```

2
```
ALTER SYSTEM SET heat_map = on;
```

3
```
ALTER TABLE hr.employees ILM ADD POLICY
ROW STORE COMPRESS advanced row
after 10 days of no modification;
```

4
```
ALTER TABLE hr.employees ILM ADD POLICY
ROW STORE COMPRESS advanced segment
after 30 days of no modification;
```

5
```
ALTER TABLE hr.employees ILM ADD POLICY
column STORE COMPRESS for archive high segment
after 6 months of no modification;
```

6
```
SELECT policy_name, action_type, scope,
compression_level, condition_days FROM
dba_ilmdatamovementpolicies;

POLICY_NAME ACTION_TYPE SCOPE   COMPRESSION_LEVEL CONDITION_DAYS
----------- ----------- ------- ----------------- --------------
P61         COMPRESSION ROW     ADVANCED                      10
P63         COMPRESSION SEGMENT ARCHIVE HIGH                 180
P62         COMPRESSION SEGMENT ADVANCED                      30
```

7
```
ALTER TABLE hr.employees ILM delete_all;
ALTER TABLE hr.employees ILM disable_all;
```

Příklad 16: Automatická optimalizace dat a komprese

2.4 Bezpečnost

V části bezpečnost se tato práce bude zabývat ukázkami práce s Oracle Data Redaction a rozšířenou funkcionalitou pro komplexní kontrolu hesel.

2.4.1 Oracle Data Redaction

Jak bylo napsáno, Oracle Data Redaction je funkcionalita Oracle Database maskující výstup dat např. z důvodu ochrany citlivých údajů. Tato kapitola se bude skládat z několika na sebe navazujících příkladů rozdělených dle typu maskování. Pro začátek je však potřeba vytvořit modelovou situaci pro uplatnění Oracle Data Redaction. V následujících příkladech je vytvořena jednoduchá tabulka, která zastupuje entitu pacienta, jež má následující atributy – id, jméno, příjmení, rodné číslo a popis jeho stavu. Tabulka pacient je vytvořena v PDB databázi *ora03pdb* uživatelem *a03pdb*, který má pouze oprávnění týkající se připojení, tvorby tabulky a oprávnění EXECUTE pro balík DBMS_REDACTION.

```
1 CONNECT a03pdb/Password12@localhost:1521/ora03pdb

2 CREATE TABLE pacient(id NUMBER(10) PRIMARY KEY, jmeno
  VARCHAR(255), prijmeni VARCHAR(255), rodne_cislo
  NUMBER(10), popis VARCHAR(255));

3 INSERT INTO pacient VALUES (1, 'Radek','Soucek',
  1234567890,'Opakovane bolesti hlavy');
```

Příklad 17: Vytvoření modelové situace pro následující příklady

Následující příklad popisuje redigování dat typu FULL:

1) V kapitole 1.4.1 bylo psáno, že redigování dat je postaveno na politikách, proto je potřeba vytvořit PL/SQL skript s popisem politiky, což je vlastně pouze jediný potřebný krok pro funkční maskování dat. Při definování politiky je potřeba stanovit názvy politiky, schématu, tabulky a sloupce, kterého se redigování dat týká. Vlastnost `function_type` udává zvolený typ maskování a vlastnost `expression` obsahuje logický výraz pro určení platnosti redigování dat, resp. určení skupin uživatelů, na které se

redigování vztahuje. Výraz 1=1 vyjadřuje, že k redigování dat dochází vždy, vyjma uživatele SYS a uživatelů majících oprávnění `EXEMPT REDACTION POLICY`.

2) Zobrazení výsledku, kde rodné číslo je maskováno nulou. Výchozí znaky pro maskování může systémový administrátor zjistit z tabulky `EXEMPT REDACTION POLICY`.

3) Ukázka vymazání politiky pro redigování dat. Popřípadě politiku není nutno mazat, ale lze proceduru `DROP_POLICY` nahradit procedurou `DISABLE_POLICY` pro vypnutí dané politiky.

1
```
BEGIN
  DBMS_REDACT.ADD_POLICY(
    object_schema => 'a03pdb',
    object_name   => 'pacient',
    column_name   => 'rodne_cislo',
    policy_name   => 'politika_cisla_f',
    function_type => DBMS_REDACT.full,
    expression    => '1=1');
END;
```

2
```
SELECT * FROM pacient;

ID JMENO PRIJMENI RODNE_CISLO POPIS
-- ----- -------- ----------- --------------------------
 1 Radek S*****             0 Opakovane bolesti hlavy
```

3
```
BEGIN
  DBMS_REDACT.DROP_POLICY(
    object_schema => 'a03pdb',
    object_name   => 'pacient',
    policy_name   => 'politika_cisla_f');
END;
```

Příklad 18: Úplné redigování dat

Příklad 19 znázorňuje použití metody pouze pro částečné redigování dat, kdy v tomto příkladu dochází k redigování sloupce obsahujícího příjmení. Částečné redigování v tomto příkladu začíná znakem na druhé pozici a rediguje následných šest znaků krycím znakem *.

1
```
BEGIN
  DBMS_REDACT.ADD_POLICY(
    object_schema => 'a03pdb',
    object_name   => 'pacient',
    column_name   => 'prijmeni',
    policy_name   => 'politika_cisla_p',
    function_type => DBMS_REDACT.partial,
    function_parameters => 'VVVVVV,VVVVVV,*,2,6',
    expression    => '1=1');
END;
```

2
```
SELECT * FROM pacient;

ID JMENO PRIJMENI RODNE_CISLO POPIS
-- ----- -------- ----------- --------------------------
 1 Radek S*****    1234567890 Opakovane bolesti hlavy
```

Příklad 19: Částečné redigování dat

Poslední příklad z oblasti redigování dat navazuje na předchozí příklad, jelikož nedochází k vytvoření nové politiky, ale úpravě té předešlé, kdy je tato politika rozšířena ještě o náhodné redigování dat na sloupci popis. Pří úpravě politiky je důležitou vlastností vlastnost `action`, která určuje, o jakou úpravu jde.

1
```
BEGIN
  DBMS_REDACT.ALTER_POLICY(
    object_schema => 'a03pdb',
    object_name   => 'pacient',
    policy_name   => 'politika_cisla_p',
    action        => DBMS_REDACT.ADD_COLUMN,
    column_name   => 'popis',
    function_type => DBMS_REDACT.random);
END;
```

2
```
SELECT * FROM pacient;
ID JMENO PRIJMENI RODNE_CISLO POPIS
-- ----- -------- ----------- --------------------------
 1 Radek S*****    1234567890 H6[*J\_gP>c2RdgoXFk:Zlow`Ns
```

Příklad 20: Náhodné redigování dat s úpravou politiky

2.4.2 Kontrola komplexnosti hesla

Jak již bylo zmíněno, kontrola komplexnosti hesla je důležitá funkcionalita zvyšující zabezpečení a ochranu databáze. Bohužel ve výchozím stavu není tato funkcionalita implementována. Kontrola komplexnosti hesla je tvořena několika funkcemi a tyto funkce lze implementovat spuštěním SQL skriptu `utlpwdmg.sql` umístěném v $ORACLE_HOME/rdbms/admin/. Před spuštěním nebo i po spuštění lze tento skript libovolně modifikovat a přizpůsobit verifikační funkce požadavkům bezpečnostních politik, či vytvořit funkce úplně nové. Tudíž není nutné používat pouze verifikační funkce dodávané Oracle Database - verify_function_11G, ora12c_verify_function a ora12c_strong_verify_function. [17]

Následující příklad zobrazuje prvotní postup implementování funkcionality komplexnosti hesla a názorné příklady hesel pro implementované verifikační funkce.

1) V prvním kroku je důležitá volba kontejneru neboli databáze. V případě zvolení kořenového kontejneru CDB databáze bude funkcionalita dostupná ve všech PDB databázích.

2) Spuštění skriptu pro implementování funkcionality kontroly komplexnosti hesla.

3) Upravení výchozího profilu `DEFAULT` nastavením názvu určité vytvořené verifikační funkce pro parametr `PASSWORD_VERIFY_FUNCTION`. Kontrolu komplexnosti hesla lze případně vypnout zadáním hodnoty `NULL` místo názvu verifikační funkce. V tomto případě dochází k volbě nejsilnější verifikační funkce `ora12c_strong_verify_function`, kdy se heslo musí skládat nejméně z devíti znaků, z toho dvou číslic, dvou velkých znaků, dvou malých znaků a dvou speciálních znaků. [16]

4) Ukázka vytvoření uživatele s dostatečně silným heslem pro splnění verifikační funkce `ora12c_strong_verify_function`.

5) Opětovná úprava profilu `DEFAULT` a nastavení méně silnější funkce `ora12c_verify_function`, která požaduje heslo o minimální délce osmi znaků, z toho minimálně jedna číslice a jeden abecední znak. Na rozdíl od předchozí silnější funkce, tato verifikační funkce kontroluje i podobnost hesla s uživatelským jménem nebo jménem databáze. Zároveň implementace této verifikační funkce obsahuje stručný seznam s často používanými nebezpečnými hesly jako je např. „password“ a probíhá i kontrola podobnosti hesla s tímto seznamem. [16]

6) Ukázka vytvoření uživatele s dostatečně silným heslem pro splnění verifikační funkce `ora12c_verify_function`.

```
1 ALTER SESSION SET CONTAINER = CDB$ROOT;
2 @c:\app\ORA1\product\12.1.0\dbhome_1\rdbms\admin\noncdb_to_pdb.sql
3 ALTER PROFILE default limit PASSWORD_VERIFY_FUNCTION ora12c_strong_verify_function;
4 CREATE USER c##TestUser IDENTIFIED BY "TestPass12!@";
5 ALTER PROFILE DEFAULT limit PASSWORD_VERIFY_FUNCTION ora12c_verify_function;
6 CREATE USER c##TestUser2 IDENTIFIED BY "testpass1";
```

Příklad 21: Implementování funkcionality a názorné příklady

2.5 Doplňující příklady

Poslední kapitola praktické části vysvětluje příklady použití funkcionalit viditelnosti sloupců a nový přístup pro vytváření autoinkrementační funkce, aniž by bylo potřeba manuálně vytvářet sekvenci a trigger.

2.5.1 Viditelnost sloupců

Předposledním praktickým příkladem je funkcionalita řešící viditelnost sloupců tabulek a umožňující jednoduché skrytí konkrétních dat v aplikaci před uživateli.

1) Pro následující příklad je vytvořena jednoduchá tabulka zastupující entitu knihy, která má následující atributy – id, název, autor a počet stran.

2) Neviditelnost sloupce lze specifikovat již při tvorbě tabulky. V tomto případě je použit příkaz `ALTER TABLE` k úpravě tabulky a zneviditelnění sloupce autor.

3) Ve výstupu příkazu `SELECT` lze vidět, že sloupec je opravdu skryt a nezobrazuje se.

4) Ovšem jinak tomu je v případě explicitního vyjádření sloupců v příkazu `SELECT`.

5) Sloupec autor je skryt i pro příkaz `INSERT` pro vkládání dat do tabulky.

6) Pro vložení hodnoty autor je nutné daný sloupec explicitně vyjádřit, jako tomu bylo u příkazu `SELECT` v kroku č. 4.

7) Opětovné zviditelnění sloupce autor.

8) Finální dotaz nad tabulkou kniha, kde si lze všimnout, že sloupec autor se zobrazuje až jako poslední sloupec, ačkoliv v příkazu `CREATE TABLE` byl na třetím místě (viz. kapitola 1.5.1).

```
1  CREATE TABLE kniha (id number(10) PRIMARY KEY, nazev
   varchar(255), autor varchar(255), pocet_stran number);

2  ALTER TABLE kniha MODIFY autor INVISIBLE;

3  SELECT * FROM kniha;
   ID NAZEV                POCET_STRAN
   --- -------------------- -----------
     1 Harry Potter 1              286
     2 Harry Potter 2              288
     3 Harry Potter 3              386

4  SELECT id, nazev, autor, pocet_stran FROM kniha;

    ID NAZEV                AUTOR      POCET_STRAN
   --- -------------------- ---------- -----------
     1 Harry Potter 1       Rowling            286
     2 Harry Potter 2       Rowling            288
     3 Harry Potter 3       Rowling            386

5  INSERT INTO kniha VALUES (4, 'Pán prstenů', 474);

6  INSERT INTO kniha (id, nazev, autor, pocet_stran)
   VALUES (5, 'Hobit', 'Tolkien', 384);

7  ALTER TABLE kniha MODIFY autor VISIBLE;

8  SELECT * FROM kniha;

    ID NAZEV                POCET_STRAN AUTOR
   --- -------------------- ----------- ---------
     1 Harry Potter 1              286 Rowling
     2 Harry Potter 2              288 Rowling
     3 Harry Potter 3              386 Rowling
     4 Pán prstenů                 474
     5 Hobit                       384 Tolkien
```

Příklad 22: Komplexní příklad pro obeznámení s viditelností sloupců

2.5.2 Autoinkrement

Autoinkrement označuje novou funkcionalitu „Identity column“ sloužící pro automatické vkládání hodnoty atributu identifikujícího daný záznam tabulky. Každá tabulka může obsahovat pouze jeden sloupec identity, který musí být specifikován již při vytváření tabulky. Pozdější úprava tabulky pro přidání sloupce identity není možná.

Poslední praktický příklad této práce se zaměřuje na vytvoření a porozumění této nové autoinkrementační funkce:

1) Prvním krokem je vytvoření ukázkové tabulky, jako tomu bylo v minulém příkladu. Opět je vytvořena jednoduchá tabulka zastupující entitu knihy, která má následující atributy – id, název, autor a počet stran. Nejprve je pro otestování funkcionality sloupce identity nastaven typ `ALWAYS`, který je aplikován na atribut ID, který je unikátním identifikátorem záznamů v tabulce. Ve výchozím stavu je auto inkrementace nastavena na hodnotu jedna s inkrementací jedna. Typ `ALWAYS` navíc udává, že hodnota je automaticky generována do každého vloženého záznamu a nelze ji vložit v případě explicitního vyjádření.

2) Ukázka podoby pro vložení záznamu.

3) Úprava sloupce id pro změnu typu sloupce identity z typu `ALWAYS` na `BY DEFAULT`. Inkrementovaná hodnota je vkládána v případě, že není explicitně vyjádřena v příkazu pro vložení záznamu.

4) Ukázky podoby příkazů pro vložení záznamů. Druhý příkaz pro vložení vyjadřuje explicitní přidání hodnoty ID.

5) Úprava sloupce ID pro rozšíření typu sloupce identity `BY DEFAULT` o klauzuli `ON NULL`. Inkrementovaná hodnota je vkládána v případě, že není explicitně vyjádřena v příkazu pro vložení záznamu nebo je zastoupena hodnotou NULL. Navíc pomocí klauzule `START WITH LIMIT VALUE` dochází k obnově inkrementační funkce, kdy hodnota je nastavena na největší hodnotu v sloupci ID a zvětšena o jedna. Obnova nejvyšší hodnoty je víceméně nutná, jelikož v předchozím kroku byla manuálně vložena

hodnota 10 a až by sekvence došla k této hodnotě, skončila by na chybě z důvodu unikátnosti primárního klíče. Nadto je ještě nastavena změna inkrementační hodnoty, kdy bude docházet ke zvyšování hodnoty o deset.

6) Ukázky podoby pro vložení záznamů.

7) Finální dotaz nad tabulkou kniha zobrazující změny atributu ID.

```
1  CREATE TABLE kniha (id NUMBER(10) generated always as
   identity PRIMARY KEY, nazev VARCHAR(255), autor
   VARCHAR(255), pocet_stran NUMBER(10));

2  INSERT INTO kniha (nazev, autor, pocet_stran) VALUES
   ('Harry Potter 1', 'Rowling', 286);

3  ALTER TABLE kniha MODIFY id NUMBER(10) generated by default
   as identity;

4  INSERT INTO kniha (nazev, autor, pocet_stran) VALUES
   ('Harry Potter 2', 'Rowling', 288);
   INSERT INTO kniha (id, nazev, autor, pocet_stran)
   VALUES (10,'Harry Potter 3', 'Rowling', 386);

5  ALTER TABLE kniha MODIFY id NUMBER(10) generated by
   default on null as identity (START WITH LIMIT VALUE
   INCREMENT BY 10);

6  INSERT INTO kniha (nazev, autor, pocet_stran) VALUES
   ('Pan prstenu', 'Tolkien', 474);
   INSERT INTO kniha (id, nazev, autor, pocet_stran)
   VALUES (null,'Hobit', 'Tolkien', 384);

7  SELECT * FROM kniha;
    ID NAZEV                AUTOR     POCET_STRAN
   --- -------------------- --------- -----------
     1 Harry Potter 1       Rowling           286
     2 Harry Potter 2       Rowling           288
    10 Harry Potter 3       Rowling           386
    11 Pan prstenu          Tolkien           474
    21 Hobit                Tolkien           384
```

Příklad 23: Komplexní příklad s novou technikou autoinkrementace

Shrnutí výsledků

Ačkoliv hlavním symbolem nové majoritní verze je cloudové řešení databází, samotného cloudu se tato práce dotýká pouze okrajově. Primárně se zabývá nejdůležitější pokrokovou novinkou nazvanou multitenantní architektura, která umožňuje plynulý přechod ke cloudovému řešení. Není nezbytné multitenantní architekturu využívat pouze pro cloudová řešení, ale i pro necloudové řešení pro uživatele či společnosti, které spravují a provozují velké množství databází a aplikací.

Multitenantní architektura poskytuje databázový kontejner (CDB), který může obsahovat až 252 zásuvných databází (PDB), přičemž tyto databáze sdílejí veškeré technické prostředky serveru, na kterém je databázový systém nainstalován. Tím pádem dochází k velkému zefektivnění využití technických prostředků a může dojít k obsáhlé redukci databázových serverů, která se nazývá konsolidací databází. Konsolidace databáze redukuje tedy počet technického vybavení, a tím pádem dochází k úsporám na pořizovacích nákladech, nákladech na prostor, nákladech za energie a nákladech za licencování databázových systémů. Zároveň multitenantní architektura umožňuje spravovat veškeré zásuvné databáze jako jednu jedinou. Dochází tedy k zefektivnění práce a naopak snížení časové náročnosti databázových administrátorů. Ve výsledku je možno zredukovat tým databázových administrátorů, jelikož už není např. potřeba aktualizovat nebo zálohovat zvlášť stovky databází, ale lze to udělat jedním krokem pomocí CDB databáze. Nejdůležitějším prvkem multitenantní architektury je oddělení dat a uživatelů jednotlivých zásuvných databází z důvodů ochrany dat. Multitenantní architektura využívá virtuální rozdělený datový slovník, kdy část datového slovníku je v kontejnerové databázi obsahující systémové informace společné pro všechny zásuvné databáze a druhá část datového slovníku s uživatelskými daty se nachází v konkrétní zásuvné databázi. Praktická část multitenantní architektury se věnovala základní správě architektury a byly vytvořeny a vyzkoušeny různé varianty pro vytváření zásuvných databází, připojování, odpojování a mazání PDB databází a následně se práce také věnovala správě společných a místních uživatelů, resp. rolí v multitenantní architektuře.

Multitenantní architektuře jako pokrokové a nejdůležitější nové technologii se práce věnovala více než z poloviny a v dalších částech práce přešla do oblasti správy dat, bezpečnosti a doplňující části z tvorby struktury databáze. Oblast správy dat slouží k efektivnímu využití datových uložišť pomocí komprimace dat a přesunu dat mezi různě rychlými datovými uložišti. Oracle Database přichází s funkcionalitou automatické optimalizace dat, která řídí životní cyklus dat pomocí nadefinovaných politik a dle nich poté dochází ke komprimaci dat pomocí různých technik, nebo k přesunu dat mezi různě rychlými datovými uložišti. K tomu, aby mohlo docházet k efektivní komprimaci nebo přesunu dat, je ale zapotřebí monitorovat přístupy nebo poslední úpravy nad daty v tabulkách, k čemuž je využit nástroj v překladu nazývaný tepelná mapa.

V oblasti bezpečnosti přišel Oracle Database s novou funkcionalitou pro redigování neboli maskování citlivých dat před neoprávněnými uživateli či z důvodů ochrany osobních údajů. Tato funkcionalita se nazývá Oracle Data Redaction a umožňuje využít několik druhů redigování dat jako např. úplné či pouze částečné maskování dat. K samotnému redigování dat dochází v reálném času až při výstupu dat po SQL dotazu, a tím pádem není nutná žádná úprava uložených dat. V oblasti zabezpečení v nové verzi Oracle Database 12c prošla vylepšením kontrola komplexnosti hesla, kde byly přidány nové předdefinované funkce zaručující silnější a bezpečnější politiky pro vytváření uživatelských hesel. V praktické části byly vytvořeny a otestovány příklady znázorňující tři druhy redigování dat a ukázkový příklad s nastavením kontrolní komplexnosti hesel.

V poslední části se práce zabývala novými možnostmi při vytváření či úpravě struktury databáze. Uživatelé Oracle Database se konečně dočkali nové auto inkrementační techniky hodnoty identifikačního sloupce tabulky. Již není potřeba složitě manuálně vytvářet sekvenci a trigger, ale stačí při definování struktury tabulky využít klauzule GENERATED AS IDENTITY za daný sloupec s danou možností, kdy má dojít k automatickému vložení hodnoty. Poslední funkcionalitou této práce bylo nastavení viditelnosti sloupců tabulky, která umožňuje zneviditelnit konkrétní sloupce tabulky, pokud není explicitně napsán v SQL dotazu.

Závěry a doporučení

Tato práce nemohla vystihnout veškeré nové technologie a přístupy v nové majoritní verzi Oracle Database 12c, jelikož s příchodem nové verze bylo zpřístupněno přes tisíc nových přístupů a funkcí, ale zaměřila se na technologie a přístupy, které byly shledány důležitými autorem práce.

Bakalářská práce by mohla být přínosem pro databázové administrátory i studenty pro seznámení s novými technologiemi a přístupy této nové databázové verze a poukázat na důvody, proč je nová majoritní verze databázového systému primárně s podporou multitenantní architektury tak výhodná ať z ekonomického či pracovního hlediska, ale i poskytnout základní pracovní postupy díky ukázkovým praktickým příkladům, navíc v přívětivé podobě českého jazyka, jelikož všechny zdroje pochází z anglického jazyka.

Do budoucna se lze zabývat zpracováním dalších důležitých funkcionalit, které se do této práce nevešly. Dále je možné zabývat se detailním zpracováním cloudového řešení a všech produktů v Oracle Database 12c.

Seznam použité literatury

[1] MELL, Peter a Timothy GRANCE. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *The NIST Definition of Cloud Computing* [online]. 2011 [cit. 2015-06-30]. Special Publication 800-145. Dostupné z: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[2] GREENWALD, Rick, Robert STACKOWIAK a Jonathan STERN. *Oracle Essentials*, *Fifth edition*. Sebastopol: O'Reilly Media, 2013, 405 s. ISBN 1449343031.

[3] LLEWELLYN, Bryn. *Consolidation Best Practices*. In: Oracle [online]. Redwood Shores: Oracle Corporation, November 2014 [cit. 2015-07-11]. Dostupné z: http://www.oracle.com/us/products/database/database-private-cloud-wp-360048.pdf

[4] *Accelerate the Journey to Enterprise Cloud with Oracle Database 12c*. In: Oracle [online]. Redwood Shores: Oracle Corporation, April 2015 [cit. 2015-07-11]. Dostupné z: http://www.oracle.com/technetwork/database/database-cloud/journey-to-enterprise-cloud-wp-1959164.pdf

[5] AVRIL, Penny a Willie HARDIE. *Plug into the Cloud with Oracle Database 12c*. In: Oracle [online]. Redwood Shores: Oracle Corporation, 2013 [cit. 2014-06-22]. Dostupné z: http://www.oracle.com/technetwork/database/plug-into-cloud-wp-12c-1896100.pdf?ssSourceSiteId=ocomen

[6] LLEWELLYN, Bryn. *Oracle Multitenant*. In: Oracle [online]. Redwood Shores: Oracle Corporation, June 2013 [cit. 2014-06-22]. Dostupné z: http://www.oracle.com/technetwork/database/multitenant-wp-12c-1949736.pdf?ssSourceSiteId=ocomen

[7] ASHDOWN, Lance a Tom KYTE. *Oracle Database Concepts, 12c Release 1 (12.1)*. In: Oracle [online]. 2014 [cit. 2014-06-23]. Dostupné z: http://docs.oracle.com/database/121/CNCPT/E41396-12.pdf

[8] SOORMA, Gavin. *12c New Feature - Container and Pluggable Databases (Multitenant Option)*. In: Oracle DBA – Tips and Techniques [online]. 2013 [cit.

2014-06-23]. Dostupné z: http://gavinsoorma.com/2013/07/12c-new-feature-container-and-pluggable-databases-multitenant-option/

[9] URBANO, Randy. *Oracle Database Administrator's Guide, 12c Release 1 (12.1)*. In: Oracle [online]. 2014 [cit. 2014-06-29]. Dostupné z: http://docs.oracle.com/database/121/ADMIN/E41484-10.pdf

[10] JERNIGAN, Kevin. *Automatic Data Optimization with Oracle Database 12c*. In: Oracle [online]. 2013 [cit. 2014-06-30]. Dostupné z: http://www.oracle.com/technetwork/database/automatic-data-optimization-wp-12c-1896120.pdf

[11] *Oracle Database VLDB and Partitioning Guide, 12c Release 1 (12.1)*. In: Oracle [online]. Redwood Shores: Oracle Corporation, 2015 [cit. 2015-07-08]. Dostupné z: https://docs.oracle.com/cloud/latest/db121/VLDBG.pdf

[12] *Oracle Advanced Compression with Oracle Database 12c*. In: Oracle [online]. Redwood Shores: Oracle Corporation, January 2015 [cit. 2015-07-11]. Dostupné z: http://www.oracle.com/technetwork/database/options/compression/advanced-compression-wp-12c-1896128.pdf

[13] LEDBETTER, Scott. *Implementing Hybrid Columnar Compression on the Oracle ZFS Storage Appliance*. In: Oracle [online]. Redwood Shores: Oracle Corporation, May 2014 [cit. 2015-07-11]. Dostupné z: http://www.oracle.com/technetwork/server-storage/sun-unified-storage/documentation/problemsolver-hcc-52014-2202692.pdf

[14] *Oracle Database New Features Guide, 12c Release 1 (12.1)*. In: Oracle [online]. 2015 [cit. 2015-06-30]. Dostupné z: https://docs.oracle.com/database/121/NEWFT/E49322-10.pdf

[15] HUEY, Patricia. *Oracle Database Advanced Security Guide 12.1*. In: Oracle [online]. May 2015 [cit. 2015-07-16]. Dostupné z: http://docs.oracle.com/database/121/ASOAG/E50333-09.pdf

[16] HUEY, Patricia. *Oracle Database Security Guide, 12c Release 1 (12.1)*. In: Oracle [online]. November 2014 [cit. 2015-07-16]. Dostupné z: http://docs.oracle.com/database/121/DBSEG/E48135-11.pdf

[17] ROESER, Mary Beth. *Oracle Database SQL Language Reference, 12c Release 1 (12.1)*. In: Oracle [online]. May 2015 [cit. 2015-07-20]. Dostupné z: http://docs.oracle.com/database/121/SQLRF/E41329-17.pdf

Seznam použitých zkratek

ADO	Automatic Data Optimalization, Automatická optimalizace dat je nástroj ILM pro tvorbu politik a dle nich automatickou komprimaci a přesun dat mezi datovými uložišti
CDB	Container Database, databázový kontejner umožňující mít připojených až 252 PDB databází
DaaS	Database as a Service, cloudová služba poskytující databázovou instanci uživateli
DDL	Data Definition Language, příkazy pro tvorbu nebo změnu struktury dat
DML	Data Manipulation Language, příkazy pro výběr, vložení, upravení a smazání dat
IaaS	Infrastructure as a Service, cloudová služba poskytující výpočetní zařízení
ILM	Information Lifecycle Management, oddělení pro monitorování, vytváření a řízení strategií pro ukládání dat
Non-CDB	Non Container Database, klasická databáze nepodporující multitenantní architekturu
ODCA	Oracle Database Configuration Assistent, grafický nástroj pro vytváření a konfiguraci databází
PaaS	Platform as a Service, cloudová služba poskytující kompletní platformu pro vývoj a běh aplikací
PDB	Pluggable Database, zásuvná databáze multitenantní architektury
PL/SQL	Procedural Language/Structured Query Language, procedurální rozšíření jazyka SQL
SaaS	Software as a Service, cloudová služba poskytující konkrétní specializovanou aplikaci
SQL	Structured Query Language, strukturovaný dotazovací jazyk pro relační databáze

Seznam obrázků

Seznam příkladů

Seznam tabulek

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Radek Souček**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Barbora Tesařová**

Název práce:
Nové technologie a přístupy v Oracle Database 12c

Název práce v AJ:
New technologies and approaches in Oracle Database 12c

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je představit nové technologie a přístupy databázové platformy Oracle Database 12c. V práci bude na praktických příkladech představena multitenantní architektura, konsolidace dat a správa databáze.

Osnova práce:
- Úvod
- Novinky v Oracle Database 12c
- Multitenantní architektura
  - CDB
  - PDB
- Konsolidace dat
- Praktické příklady na využití nových technologií
- Závěr

Projednáno dne: 15. 10. 2014

Podpis studenta

Podpis vedoucího práce