Bezpečnosť v databázových systémoch

Diplomová práca

Radoslav Golian **2007**

Bezpečnosť v databázových systémoch

DIPLOMOVÁ PRÁCA

Radoslav Golian

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

Informatika

Školiteľ diplomovej práce

Ing. Miroslav Galbavý

BRATISLAVA, 2007

Čestne vyhlasujem, že som diplomovú prácu vypracoval samostatne, s použitím literatúry a zdrojov uvedených v záverepráce.

Bratislava, máj 2007

....

Radoslav Golian

Ďakujem môjmu diplomovému vedúcemu Ing. Miroslavovi Galbavému za odborné vedenie práce, cenné rady a pripomienky. Ďakujem mojej rodine a priateľke za podporu pri písaní tejto práce. Ďakujem aj firme Centaur s.r.o. za poskytnutie nástrojov, ktoré mi uľahčili modelovanie a programovanie.

Abstrakt

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

Autor: Radoslav Golian

Názov diplomovej práce: Bezpečnosť v databázových systémoch

Školiteľ diplomovej práce: Ing. Miroslav Galbavý

Rozsah práce: 84 strán (100 strán s prílohami)

Bratislava, máj 2007

Práca analyzuje problematiku zavedenia a udržiavania bezpečnosti v databázových systémoch, identifikuje najčastejšie bezpečnostné problémy a načrtáva riešenia týchto problémov, pričom sa sústredí na jemne štruktúrované riadenie prístupu (Fine-Grained Access Control, FGAC) a šifrovanie. V tejto práci sa nachádza porovnanie súčasného stavu implementácie bezpečnostných služieb a mechanizmov vo vybraných databázových systémoch. Posledná časť práce je zameraná na návrh databázových aplikácií s ohľadom na bezpečnosť, pričom pracujeme v databázovom systéme Oracle a zameriavame sa na využitie FGAC a transparentného šifrovania dát. Súčasťou práce sú zdrojové kódy k ukážkovej implementácii FGACa transparentného šifrovania.

Kľúčové slová: databáza, bezpečnosť, jemne štruktúrované riadenie prístupu, virtuálne privátne databázy, šifrovanie, transparentné šifrovanie dát, autorizované pohľady, Oracle

Obsah

Abstrakt....................1

Abecedný zoznam skratiek....................4

Zoznam tabuliek a obrázkov....................6

1 Úvod....................7

2 Bezpečnosť v DBMS....................8

2.1 Základné princípy bezpečnosti....................8

2.1.1 Bezpečnostné politiky....................12

2.1.2 Analýza rizík....................15

2.2 Riadenie prístupu v RDBMS....................17

2.3 Šifrovanie dát v DBMS....................27

2.4 Manažment kľúčov v DBMS....................28

2.5 Problémy a možnosti riešenia bezpečnosti v DBMS....................30

2.6 Prehľad bezpečnostných mechanizmov vo vybraných DBMS....................38

2.6.1 Hešovanie a šifrovanie....................38

2.6.2 Identifikácia a autentifikácia....................39

2.6.3 Riadenie prístupu....................39

2.6.4 Manažment kľúčov....................41

2.6.5 Deklaratívne TDE....................42

2.6.6 Používateľské profily....................43

3 Implementácia zabezpečenia na platforme Oracle....................44

3.1 Aplikačný kontext....................44

3.2 FGAC....................45

3.2.1 Implementácia autorizovaných pohľadov....................45

3.2.2 Implementácia pomocou VPD....................56

3.2.3 Porovnanie autorizovaných pohľadov a VPD....................62

3.3 Šifrovanie a hešovanie....................64

3.3.1 Balík DBMS_CRYPTO....................64

3.3.2 Problémy súvisiace s použitím balíka DBMS_CRYPTO....................66

3.4 Transparentné šifrovanie dát....................71

3.4.1 TDE pomocou pohľadov....................71

3.4.2 TDE deklaratívnym spôsobom....................74

3.4.3 Vyhodnotenie 76
4 Záver..........79
5 Zoznam bibliografických odkazov..........81
6 Prílohy..........85
Príloha 1: Porovnanie bezpečnostných prvkov DBMS..........85
Príloha 2: CD s ukážkovými príkladmi..........87
Príloha 3: Porovnanie výkonnosti aplikačného kontextu a funkcie balíka v dopyte.....88
Príloha 4: Popis dátového modelu vzorového príkladu..........89
Príloha 5: Príklad autorizovaného pohľadu..........92
Príloha 6: Príklad použitia autorizovaného pohľadu..........94
Príloha 7: Porovnanie výkonnosti implementácií TDE..........96
Príloha 8: Popis parametrov funkcie ADD_POLICY..........98
Príloha 9: Príklad dynamickej politiky..........99

Abecedný zoznam skratiek

3DES	*Triple Data Encryption Standard*
ACL	*Access Control List*
AES	*Advanced Encryption Standard*
CBC	*Cipher Block Chaining*
CFB	*Cipher Feedback*
DAC	*Discretional Access Control*
DBA	*Database Administrator*
DBMS	*Database Managment System*
DCE	*Distributed Computing Enviroment*
DES	*Data Encryption Standard*
DML	*Data Manipulation Language*
DoS	*Denial-of-Service*
DSD	*Dynamic Separation of Duty*
ECB	*Electronic Codebook*
HTTP	*Hypertext Transfer Protocol*
IDS	*Intrusion Detection System*
LDAP	*Lightweight Directory Access Protocol*
FGAC	*Fine-Grained Access Control*
MAC	*Mandatory Access Control*
MD2, MD4, MD5	*Message Digest, Version 2,4,5*
NAT	*Network Address Translation*
NTLM	*NT Lan Manager*
OFB	*Output Feedback*
OLS	*Oracle Label Security*

OS	*Operating System*
PAM	*Pluggable Authentication Module,*
PL/SQL	*Procedural Language/Structured Query Language*
PKI	*Public Key Infrastructure*
RADIUS	*Remote Authentication Dial in User Service*
RBAC	*Role-based access* control
RDBMS	*Relational Database Managment System*
SMTP	*Simple Mail Transfer Protocol*
SQL	*Structured Query Language*
SHA	*Secure Hash Algorithm*
SSD	*Static Separation of Duty*
SSL	*Secure Sockets Layer*
SSO	*Single Sign-On*
TDE	*Transparent Data Encryption*
VPD	V*irtual Private Database*
VPN	*Virtual Private Network*

Zoznam tabuliek a obrázkov

Tabuľka 2.2.1: Prehľad oprávnení pre manipuláciu s dátami v databázových tabuľkách podľa SQL 2003 20

Tabuľka 3.3.1.1: Prehľad možností šifrovania a hešovania balíka DBMS_CRYPTO.... 65

Obrázok 2.6.4.1: Manažment kľúčov v DBMS SQL Server 2005. …..........42

Obrázok 3.2.1.1: Graf závislosti doby vykonávania dopytu od počtu záznamov v tabuľke a použitej metódy.…..........47

Obrázok 3.2.1.2: Dátový model modelovej aplikácie. ….......... 49

Obrázok 3.2.1.3: Architektúra modelovej aplikácie a nastavovanie aplikačného kontextu.….......... 50

Obrázok 3.2.1.4: Autentifikácia zákazníka. Pri úspešnej autentifikácii sa nastaví aplikačný kontext.......... 51

Obrázok 3.4.1.1: Tabuľka CARD.…..........71

Obrázok 3.4.3.1: Porovnanie rýchlosti dopytov nad TDE..….......... 77

1 Úvod

Digitálna revolúcia nám umožnila komunikovať a bezprostredne pristupovať k informáciám kdekoľvek na svete. Zároveň však rozšírila spektrum rizík vzťahujúcich sa na citlivé a cenné informácie. Preto vzrastá potreba zabezpečiť obmedzený prístup k týmto informáciám.

DBMS sú v súčasnosti využívané v mnohých oblastiach týkajúcich sa bežného života väčšiny ľudí. Tieto systémy sa využívajú napríklad v bankách, zdravotných a sociálnych poisťovniach, štátnej správe, telekomunikáciách, rôznych finančných inštitúciách a iných oblastiach, pričom sa v nich často koncentrujú dôverné informácie, prípadne informácie, ktoré je potrebné chrániť pred modifikáciou nepovolanými osobami.

Predstavme si, že by sa útočníkovi úspešne podarilo preniknúť do informačného systému banky a následne aj do DBMS, pričom by získal možnosť manipulovať s našimi finančnými prostriedkami. Následky by mohli byť katastrofálne nie len pre nás, ale aj pre banku. My by sme mohli stratiť naše úspory a banka by stratila dôveru svojich klientov.

Požiadavka bezpečnosti DBMS teda vznikla z potreby chrániť dáta, jednak pred stratou a poškodením, a jednak pred neautorizovanými pokusmi o prístup alebo o zmenu týchto dát.

V tejto práci sa venujeme problematike bezpečnosti v databázových systémoch. V nasledujúcej kapitole uvádzame a vysvetľujeme základné pojmy, s ktorými sa čitateľ môže stretnúť pri skúmaní bezpečnosti DBMS. V tejto kapitole sa snažíme podať čitateľovi ucelený pohľad na túto problematiku, identifikovať bezpečnostné problémy, a prípadne ukázať možnosti riešenia týchto problémov. Zameriavame sa na problematiku šifrovania a riadenia prístupu a porovnávame aj bezpečnostné prvky niekoľkých súčasných DBMS.

V tretej kapitole sa venujeme implementácii zabezpečenia v DBMS Oracle a analyzujeme možnosti využitia FGAC a TDE. Navrhujeme spôsoby implementácie FGAC a TDE a demonštrujeme tieto spôsoby na príkladoch. V tejto kapitole ďalej popisujeme problémy, s ktorými sme sa pri implementácii stretli, a navrhujeme možnosti riešenia týchto problémov.

2 Bezpečnosť v DBMS

V tejto kapitole sme si dali za cieľ oboznámiť čitateľa so základnými princípmi bezpečnosti, a to hlavne v kontexte DBMS, pričom sa podrobnejšie zaoberáme možnosťami riadenia prístupu a šifrovania. V ďalších podkapitolách poukazujeme na bezpečnostné problémy a možné riešenia týchto problémov, a taktiež porovnávame bezpečnostné mechanizmy vo vybraných súčasných DBMS.

2.1 Základné princípy bezpečnosti

Pri vytváraní tejto kapitoly sme čerpali informácie a inšpiráciu z nasledujúcich zdrojov: [1], [2], [3], [4], [5], [6], [7], [8], [9] a [10].

Vo svete bezpečnosti sa používajú pojmy entita (*entity*) a identita (*identity*). Entita predstavuje človeka, aplikáciu, počítač, databázového používateľa a podobne. Identitu si môžeme predstaviť ako entitu v určitej bezpečnostnej doméne. Príkladom môže byť používateľ dvoch rôznych DBMS. Ide o tú istú entitu (človeka), ktorá predstavuje dve rozdielne identity v dvoch rozdielnych doménach. Bezpečnostnou doménou môže byť teda aj DBMS, ktorý používa stanovenú množinu pravidiel, ktoré definujú identitu, a podľa ktorých je táto identita vytvorená. Tieto pravidlá určujú, čo tieto identity majú povolené robiť (vytvoriť tabuľku, pohľad a pod.) resp. k akým informáciám majú prístup v rámci danej domény (vybranie záznamov z tabuľky a pod.).

Medzi základné požiadavky na bezpečnostné služby v DMBS patrí:

1. dôvernosť (utajenie) informácií (*confidentiality*) – Prístup k jednotlivým informáciám musí byť povolený jedine autorizovaným entitám. Splnenie tejto požiadavky je najčastejšie v DBMS realizované pomocou autentifikačných a autorizačných mechanizmov, pričom citlivé informácie musia byť šifrované a databázové súbory musia byť umiestnené na zabezpečenom serveri.

2. zaistenie integrity informácií (*integrity*) – Informácia z hľadiska bezpečnosti spĺňa vlastnosť integrity, ak nebola zmenená žiadnou entitou, ktorá na jej zmenu nemá právo. V kontexte DBMS a bezpečnosti zahŕňa táto požiadavka viacero aspektov:

   a) riadenie prístupu (*access control*) – Prístup k databázovým objektom musí byť riadený pomocou oprávnení, takže iba oprávnené entity môžu meniť dáta.

   b) ochrana DBMS – DBMS musí byť chránený proti škodlivému kódu, ktorý by

mohol poškodiť dáta (napr. vírusy).

c) ochrana sieťovej komunikácie – Sieťová komunikácia medzi DBMS a inou entitou musí byť chránená proti odstráneniu, poškodeniu a odpočúvaniu.

3. dostupnosť informácií (*availability*) – Informácie by mali byť autorizovanej entite dostupné kedykoľvek ich bude potrebovať. Dostupnosť informácií môžu ohroziť napríklad útoky typu DoS (*Denial-of-service*) [11].

4. sledovateľnosť (*auditing*) – Mala by existovať možnosť uchovávania záznamov o prístupoch (oprávnených ako aj neoprávnených) k informáciám a následná analýza týchto záznamov, ktorá môže prispieť k zvýšeniu bezpečnosti. V niektorých DBMS môžeme napríklad sledovať, kto a kedy databázové objekty vytváral, odstraňoval, editoval a podobne.

Bezpečnostnými požiadavkami ako napríklad ochrana dôvernosti, zaistenie integrity či nepopretie autorstva, ale aj aplikačnými oblasťami bezpečnosti (protokoly pre autentifikáciu, digitálne podpisy, atď.) sa zaoberá samostatná vedná oblasť – kryptológia. Kryptológia sa delí na dve časti – kryptografiu a kryptoanalýzu. Kryptografia sa zaoberá návrhom konštrukcií napĺňajúcich konkrétne bezpečnostné požiadavky. Úlohou kryptoanalýzy je skúmať možnosti útokov na tieto konštrukcie[3].

Na zabezpečenie informácií existujú vo všeobecnosti nasledujúce základné bezpečnostné mechanizmy:

Kryptografické hešovanie (*cryptografic hashing*) – kryptografické hešovacie funkcie[1] zobrazujú elektronický dokument na správu alebo „odtlačok“, obvykle podstatne kratší ako pôvodný dokument. V našej práci budeme používať ďalej len pojem hešovacie funkcie, budeme mať však na mysli kryptografické hešovacie funkcie. Hešovacie funkcie sa používajú v schémach digitálnych podpisov, pri kontrole integrity, v kryptografických protokoloch a ďalších konštrukciách. V súvislosti s hešovacími funkciami sa môžeme stretnúť s pojmom soľ (*salt*). Soľ je reťazec náhodných bitov. V databázových aplikáciach a v DBMS sa hešovanie často používa napríklad na hešovanie hesiel. Soľ v tomto prípade môžeme použiť tak, že ňou rozšírime heslo predtým ako naň aplikujeme hešovaciu funkciu. Týmto krokom je možné výrazne sťažiť[2] slovníkové útoky[5]. Viac informácií o hešovacích funkciách a o vlastnostiach, ktoré by mali spĺňať, aby boli vhodné

1 Kryptografické hešovacie funkcie sú špeciálnou triedou hešovacích funkcii.
2 Pokiaľ má soľ t bitov, potom slovník by musel obsahovať 2^t krát viac hodnôt.

na kryptografické použitie, je možné nájsť v [3], [4] a [5].

Šifrovanie (*encryption*) – Šifrovanie si kladie za cieľ transformovať vstupné dáta do podoby, v ktorej sú pre potenciálneho útočníka nezrozumiteľné a nie je schopný rekonštruovať ich pôvodný tvar. Zároveň požadujeme, aby oprávnené entity mohli pôvodné dáta rekonštruovať. Vstupné dáta v ich pôvodnej podobe budeme nazývať otvorený text, prípadne správa. Proces ich transformácie sa nazýva šifrovanie a je realizované šifrovacím algoritmom (funkciou). Výsledok šifrovania je šifrový text (správa). Proces inverznej transformácie, keď zo šifrového textu dostaneme opäť pôvodný otvorený text, sa nazýva dešifrovanie (*decryption*) a je realizovaný dešifrovacím algoritmom (funkciou). Šifrovací algoritmus je parametrizovaný ďalším vstupom – kľúčom, ktorý nezávisí na otvorenom texte. Dešifrovací algoritmus je taktiež parametrizovaný kľúčom. Existujú tri typy šifrovania:

1. symetrické – Na šifrovanie a dešifrovanie sa použije ten istý utajený šifrovací kľúč. V DBMS sa najčastejšie využívajú blokové šifry. Blokové šifry sú triedou symetrických šifrovacích systémov, ktoré spracúvajú otvorený aj šifrový text po blokoch pevnej dĺžky. Pokiaľ dĺžka otvoreného textu nie je deliteľná veľkosťou bloku musíme použiť vypchávku (*padding*), pomocou ktorej doplníme otvorený text na požadovanú dĺžku. Vypchávka môže byť napríklad konštanta alebo náhodný reťazec. Požiadavky na vypchávku definuje štandard PKCS #5[12]. V blokových algoritmoch môže byť základná šifrovacia transformácia jedného bloku kombinovaná vo viacerých módoch (najznámejšie sú CBC, CFB, ECB, OFB). Viac informácii o blokových šifrách je možné nájsť napríklad v [3] a [13].

2. asymetrické – Na šifrovanie a dešifrovanie sa použijú dva rôzne kľúče – verejný a súkromný. Obidva kľúče sú skonštruované používateľom, pričom verejný kľúč používateľ zverejní a súkromný kľúč ponechá v tajnosti. Verejný kľúč slúži na šifrovanie a súkromný kľúč na dešifrovanie. Teda šifrovať môže každá entita, ktorá pozná daný verejný kľúč a dešifrovať môže iba vlastník súkromného kľúča. Pri asymetrickom šifrovaní, musí mať šifrovací systém nasledujúce vlastnosti:

   a) korektnosť – Dešifrovanie šifrovaného textu vedie k pôvodného textu.

   b) realizovateľnosť – Šifrovanie, dešifrovanie a vytváranie kľúčov je efektívne (obvykle polynomiálna časová zložitosť).

c) bezpečnosť – Zo znalosti verejného kľúča je „prakticky nemožné“ určiť súkromný kľúč. Význam slovného spojenia „prakticky nemožné“ záleží od našich požiadaviek.

3. hybridné – Symetrické šifrovanie je rýchlejšie v porovnaní s asymetrickým šifrovaním. Väčšinou sa asymetrické šifrovanie využíva na „dohodnutie“ kľúča symetrickej šifry (vygenerovaný kľúč pre symetrické šifrovanie je zašifrovaný verejným kľúčom adresáta) pri nadviazaní komunikácie medzi komunikujúcimi stranami. Takéto šifrovanie sa nazýva hybridné šifrovanie. Na zníženie možnosti prelomenia zabezpečenej komunikácie sa odporúča pri každej komunikácii využívať iný symetrický kľúč.

Viac informácií o šifrovaní je možné nájsť v [3], [4], [5] a [13].

Digitálny podpis (*digital signature*) – Digitálny podpis je kryptografická konštrukcia, ktorej cieľom je zabezpečiť autentickosť a integritu (prípadne aj nepopretie autorstva) správ a dokumentov v elektronickom prostredí. Viac informácií o digitálnom podpise je možné nájsť v [3] a [5].

Certifikát (*certificate*) – Certifikát je údajová štruktúra, ktorá obsahuje verejný kľúč žiadateľa, resp. držiteľa certifikátu, časový údaj, ktorý sa vzťahuje k dobe platnosti certifikátu a iné údaje vytvorené certifikačnou autoritou. Táto údajová štruktúra je podpísaná súkromným kľúčom certifikačnej autority, pričom akákoľvek entita môže verifikovať obsah certifikátu pomocou verejného kľúča certifikačnej autority. Certifikačná autorita (*certification authority*) je dôverná tretia strana, ktorá na základe žiadosti vydáva a aktualizuje certifikáty. Viac informácii je možné nájsť v [13].

Identifikácia (*identification*) a **Autentifikácia** (*authentication*) – proces na rozpoznanie a overenie entity, ktorá chce pristupovať k určitej informácii. Takáto entita sa nazýva autentifikovanou identitou, ak je dokázané napríklad podľa mena a hesla, že je to presne ona. Príkladom môže byť autentifikácia používateľa do DBMS pomocou používateľského mena a hesla.

1. Používateľ poskytne pre DBMS svoju identitu zadaním svojho používateľského mena.

2. Používateľ dokáže, že táto identita, ktorú poskytol je validná v danom DBMS tým, že zadá správne heslo.

3. Ak používateľ preukázal v druhom kroku validitu svojej identity, DBMS pridelí používateľovi jemu prislúchajúce oprávnenia.

Prvý krok v tejto sekvencii je identifikácia, druhý krok je autentifikácia. Svoju identitu môže používateľ poskytnúť napríklad aj pomocoucertifikátu alebo biometrie.

Autentifikácia sa delí na silnú (autentifikácia pomocou digitálnych certifikátov, tokenov, biometrie) a slabú (napríklad autentifikácia pomocou hesla). Viac informácií o autentifikácii je možné nájsť v [4] a [5].

Autorizácia (*authorization*) – proces identifikujúci, ktorá identita má právo pristupovať ku ktorej informácii a akým spôsobom. Príkladom je pridelenie práv databázovému používateľovi na vkladanie záznamov do nejakej databázovej tabuľky.

2.1.1 Bezpečnostné politiky

Implementácia zabezpečenia nezačína však hešovaním, šifrovacími algoritmami a podobne, ale defíníciou bezpečnostných politík (ďalej len politika). Informácie pre vytvorenie tejto kapitoly sme čerpali z [6], [10], [14], [15] a [16].

Politika je množina pravidiel, ktorými by sa mali riadiť všetci používatelia, a ktoré predstavujú smernicu potrebnú pre implementáciu zabezpečenia. Politiky rozdeľujeme na globálne, ktoré definujú množinu všeobecných pravidiel, ktoré platia pre celú organizáciu a obsahujú všeobecnejšie a abstraktnejšie postupy, čím nastavujú úroveň zabezpečenia, a na lokálne, ktoré dopĺňajú globálne politiky tým, že definujú technické detaily. Lokálne politiky sú konkrétnejšie ako globálne politiky. Lokálne politiky môžu stanoviť podmienky, za ktorých sú možné výnimky z globálnej politiky.

Globálne politiky môžu napríklad vynucovať sledovateľnosť operácií na inštanciách DBMS. Úroveň sledovateľnosti sa však môže na jednotlivých inštanciách DBMS líšiť (napríklad v závislosti od úrovne dôvernosti informácií v nich obsiahnutých), a preto by mala byť určená v rámci lokálnych politík pre jednotlivé inštancie, a nie v rámci globálnej politiky.

Z predchádzajúceho príkladu je zrejmé, že jednotlivé politiky sa môžu líšiť v úrovni vynucovaného zabezpečenia. Pre dôverné dáta musíme definovať prísne politiky pokrývajúce ochranu týchto dát na súborovom systéme, ochranu médií obsahujúcich zálohy dôverných dát, a podobne. Pre dáta, ktoré nie sú dôverné definujeme zväčša menej

obmedzujúce a menej obsažné politiky.

Na to, aby sme mohli vytvoriť správne politiky potrebujeme porozumieť bezpečnostným rizikám. Toto zahŕňa aj identifikáciu potenciálnych útočníkov, ich cieľov a motivácie, a taktiež spôsobov, ktorými môžu tieto ciele dosiahnuť. Predstavme si napríklad nasledujúcu situáciu: Existujú politiky, ktoré vynucujú dôvernosť pomocou šifrovania citlivých informácií v DBMS, pričom tajné kľúče sú manažované pomocou DBMS a DBA nebol identifikovaný ako potenciálny útočník. Keďže sú kľúče manažované pomocou DBMS, DBA má k týmto kľúčom prístup a môže získať informácie, dôvernosť ktorých sme chceli zaručiť. Takéto politiky teda nie sú dostatočné, keďže nechránia dáta pred DBA, ako potenciálnym útočníkom. Jedným z možných riešení je definovať dodatočnú politiku, vynucujúcu sledovanie čítania citlivých informácii, pričom záznamy zo sledovania by sa ukladali v OS (predpokladáme, že DBA nie je administrátor OS).

Pri vytváraní politík by sme sa mali snažiť znovu využívať už existujúce politiky. Jedným z možných prístupov je vytvorenie abstraktných globálnych politík – šablón, pričom tieto šablóny kategorizujeme podľa toho, na aké informácie budú mať dosah (dôverné, dostupné pre vybraných partnerov, verejne dostupné, a podobne). Lokálne politiky budú potom vytvárané na základe týchto šablón. Vytváraním bezpečnostných politík sa zaoberá aj niekoľko nezávislých organizácii, ako napríklad:

- National Institutes of Standards and Technologies (NIST) – v edícii dokumentov Special Publications 800 Series [14].
- International Standards Organization (ISO) – v štandarde ISO/IEC 17799:2005 [15].
- Health Insurance Portability and Accountability Act (HIPAA) [16].

Dokumenty od týchto organizácii poskytujú veľmi dobrý a vyčerpávajúci zdroj informácií, na základe ktorých je možné vytvoriť bezpečnostné politiky.

Ak vytvárame robustnú a komplexnú politiku, je potrebné brať ohľad na možné vedľajšie efekty. Vedľajším efektom príliš prísnej politiky často býva znížená použiteľnosť a výkonnosť systému. Ideálnym ilustračným príkladom zníženej použiteľnosti je prísna politika pre používateľské heslá[3] vytvorená za účelom zvýšenia bezpečnosti. Vedľajším efektom takejto prísnej politiky môže byť to, že bežný používateľ nie je schopný v takej

3 Napríklad: dĺžka aspoň dvanásť znakov, aspoň jedno číslo, aspoň jeden špeciálny znak, odlišnosť od predchádzajúceho hesla aspoň v troch znakoch, exspirácia každý mesiac, možnosť opätovného použitia po troch rokoch

krátkej perióde generovať a zapamätať si také zložité heslá, čo následne môže viesť k tomu, že si tento používateľ vytvorí zoznam hesiel, ktorý nechá voľne položený na svojom pracovnom mieste. Vyžadovanie takejto prísnej politiky potom nevedie k zvýšeniu bezpečnosti, ale má presne opačný efekt. K zníženiu výkonnosti systému v dôsledku neprimerane prísnej politiky dochádza napríklad pri nadmernom audite.

V praxi je návrh politík vždy kompromisom medzi zabezpečením a použiteľnosťou. Nie je možné navrhnúť politiky pre súčasné DBMS tak, aby sme dosiahli stopercentnú bezpečnosť. Ak sa o to pokúsime, výsledkom sú zväčša nepraktické, nepoužiteľné a finančne náročné politiky s negatívnym vplyvom na výkonnosť a administráciu DBMS, pričom ako sme ukázali na príklade prísnej politiky pre používateľské heslá, tieto politiky môžu viesť v konečnom dôsledku k zníženiu bezpečnosti.

V rámci analýzy sme identifikovali niekoľko všeobecných proaktívnych princípov bezpečnosti aplikovateľných na databázové aplikácie.

Návrh aplikácie s ohľadom na bezpečnosť – O bezpečnosti aplikácie je potrebné uvažovať už pri jej návrhu a nie až po implementácii. Je potrebné navrhnúť vhodnú architektúru aplikácie, rozhodnúť sa akým spôsobom budeme implementovať napríklad riadenie prístupu – použijeme DAC, RBAC, alebo budeme implementovať autorizované pohľady?

Viacvrstvová bezpečnostná architektúra – Vytvoríme viacero bezpečnostných vrstiev, čím zabránime existencii jednotného bodu zlyhania (*single point of failure*). Útočník by musel prekonať všetky vrstvy. Vrstvami môže byť napríklad firewall, autentifikácia do DBMS alebo OS, šifrovanie dát. Ak chce útočník získať dáta, musí prekonať všetky vrstvy. Viac informácii o tomto princípe je možné nájsť v [17].

Princíp najmenšej množiny oprávnení – Tento princíp hovorí o tom, že používateľ dostane jedine tie oprávnenia, ktoré nevyhnutne potrebuje pre vykonanie svojich úloh. Porušenie tohto princípu môže mať vážne dôsledky, ak sa útočníkovi podarí získať identitu používateľa s nadbytočnými oprávneniami. Extrémnym, v praxi nie zriedkavým príkladom je pridelenie roly databázového administrátora používateľovi, ktorý potrebuje napríklad vytvoriť len niekoľko tabuliek, pohľadov a úložných procedúr. Takýto prístup uľahčí prácu databázovému administrátorovi (nepotrebuje presne skúmať, aké oprávnenia používateľ potrebuje) a aj používateľovi (nestane sa, že nebude môcť vytvoriť nejaký typ objektu), pre DBMS však predstavuje veľké bezpečnostné riziko, keďže útočník, ktorému

sa podarí získať identitu takéhoto používateľa, získa kontrolu nad celým DBMS.

2.1.2 Analýza rizík

Každá bezpečnostná politika musí začať analýzou a manažmentom zistených rizík. Analýza rizík je jednou z kľúčových aktivít zvyšovania bezpečnosti. Cieľom je zistiť, čo ohrozuje DBMS a údaje v ňom, ako je DBMS proti takýmto hrozbám chránený, a kde sú zraniteľné miesta v bezpečnosti DBMS. Analýza rizík pozostáva z nasledujúcich štyroch krokov:

1. Identifikácia hrozieb
    - Externé hrozby – útočníci.
    - Interné hrozby – zamestnanci s prístupom k databázovej aplikácií, OS alebo DBMS.
    - Prírodné katastrofy – povodeň,zemetrasenie, atď.
    - Technické poruchy – napríklad výpadok elektrickej energie.
2. Identifikácia zraniteľných miest prekaždú aplikáciu (vrátane DBMS a OS).
3. Určenie pravdepodobnosti výskytu hrozby, a na základe toho určenie priority implementácie protiopatrení.
4. Identifikácia následkov, ktoré nastanú v dôsledku využitia zraniteľného miesta útočníkom a identifikácie možností zamedzenia využitia zraniteľného miesta.

Výsledky analýzy by mali byť zdokumentované, pričom táto dokumentácia by mala obsahovať rozhodnutia, ktoré boli vybrané a dôvody, pre ktoré boli vybrané. Mala by obsahovať aj zoznam identifikovaných hrozieb a zraniteľných miest. Výsledky môžu byť použité ako referencia pri audite a môžu slúžiť aj ako šablóna pre ďalšie systémy.

Keď už sme identifikovali relevantné riziká, ďalším krokom je vytvoriť procedúry, ktoré sa budú uplatňovať v prípade výskytu ohrozenia, obsahujúce postupnosť krokov, ktoré je potrebné vykonať. Existuje niekoľko možných všeobecne akceptovaných postupov.

Redukcie hrozieb (*Threat reduction*) – Napríklad môžeme odstrániť hrozbu uhádnutia hesla použitím iného typu autentifikácie (biometria, certifikáty).

Redukcia zraniteľných miest (*Vulnerability reduction*) – Zabezpečíme zraniteľné miesta. Napríklad identifikujeme zraniteľný kód v databázovej aplikácii, ktorý nie je

odolný voči injektovaniu do SQL dopytov.

Zmiernenie dopadu a následkov (*Impact reduction*) – Príkladom môže byť použitie vhodnej architektúry pre databázové aplikácie, čo môže zahŕňať segregáciu databázových objektov do viacerých schém, následné uzamknutie týchto schém a vytvorenie databázovej schémy, ktorá bude mať len potrebné oprávnenia nad týmito objektami. Ak by aplikácia v tomto prípade nebola odolná voči SQL injektovaniu, útočník by síce možno vedel prečítať údaje, pre ktoré nie je autorizovaný, ale nevedel by odstrániť databázové objekty, pokiaľ by na to databázový používateľ pod ktorým sa dopyt spúšťa nemal právo. V prípade, že by bol dopyt spúšťaný pod vlastníkom databázových objektov, útočník môže odstrániť ľubovoľný objekt, ktorý tento databázový používateľ vlastní.

Detekcia (*Detection*) – Môžeme použiť systémy umožňujúce detekciu prienikov. V DBMS je možné použiť systémy na detekciu injektovania do SQL dopytov.

Zotavovanie (*Recovery*) – V prípade straty dát, či už z dôvodu technickej poruchy, prírodnej katastrofy, alebo deštrukčnej činnosti útočníka, by sme mali byť schopní tieto dáta obnoviť a z chyby sa zotaviť.

Riziko je možné aj preniesť na inú organizáciu (poistenie, outsourcing), prípadne akceptovať a neaplikovať žiadne protiopatrenia, obyčajne z dôvodu vysokej ceny (minieme 3000Sk aby sme zabezpečili údaje, ktoré majú pre nás hodnotu 1000Sk?).

DBMS interaguje s počítačovou sieťou, OS, klientskými pracovnými stanicami, a prípadne aj s inými servermi, preto je dôležité zabezpečiť celú infraštruktúru. Cieľom tejto práce je venovať sa bezpečnosti v DBMS, preto uvedieme len stručný a neúplný zoznam problémov, ktoré je potrebné uvažovať v rámci zabezpečenia infraštruktúry:

- Fyzické zabezpečenie serverov
- Nepotrebné služby (telnet, ftp, zdieľanie a podobne)
- Nastavenie práv pre súbory a adresáre a prípadne sledovanie zmien práv
- Nepoužívané používateľské účty
- Nepotrebné a nepoužívané aplikácie
- Uzamykanie pracovných staníc
- Detekcia prienikov a využitie detekčných systémov (*IDS, Intrusion Detection System*)

- Ochrana pre počítačovými infiltráciami (vírusy, trójske kone a podobne)
- Aplikovanie záplat (*patching*)
- Monitorovanie logovacích súborov a zabezpečenie týchto súborov pred modifikáciou.
- Obmedzenie počtu administrátorov
- Návrh topológie a architektúry počítačovej siete
- Zabezpečenie siete a ochrana komunikácie(Firewall, VPN [18], NAT [19], IPSec [20], SSL, atď.) [3].

Problematike zabezpečenia OS a počítačovej siete sa venuje veľké množstvo článkov, webových stránok[4] a aj knižných publikácií[5].

2.2 Riadenie prístupu v RDBMS

Jedným z cieľov našej práce je analyzovať možnosti VPD v kontexte zavedenia a udržiavania bezpečnosti. VPD je jedným zo spôsobov riadenia prístupu, preto sme sa tejto problematike rozhodli venovať samostatnú kapitolu. V súčasnosti existuje niekoľko filozoficky odlišných spôsobov riadenia prístupu, pričom medzi hlavné patria:

MAC (*Mandatory access control*) – povinné riadenie prístupu. Tento spôsob je založený na určení triedy dôvernosti informácie a autorizovaní používateľa pre vybrané triedy. Používateľ nemá právo zmeniť triedu dôvernosti, táto je určená organizáciou (administrátorom). Triedy dôvernosti môžu mať hierarchickú štruktúru (strom, les). V takomto prípade má používateľ prístup ku všetkým informáciám, ktorých trieda dôvernosti patrí do niektorého z podstromov, ktorých koreňmi sú triedy, pre ktoré je tento používateľ autorizovaný. Príkladom takejto hierarchie môže byť jednoduchá hierarchia „prísne tajné", „tajné", „dôverné", „vyhradené", kde trieda „vyhradené" je synom triedy „dôverné". Trieda „dôverné" synom triedy „tajné" a trieda „tajné" je synom triedy „prísne tajné". Používateľ, ktorý je autorizovaný pre triedu „tajné" má prístup aj k informáciám, ktoré majú triedu „dôverné" alebo „vyhradené". Používateľ, ktorý je autorizovaný pre triedu „prísne tajné" má prístup ku všetkým informáciám. Informácie môžu byť zaradené do viacerých tried (kde jednotlivé triedy môžu mať hierarchickú štruktúru). V tomto prípade má používateľ prístup k informácii len vtedy, ak je autorizovaný pre každú triedu do ktorej informácia patrí. Viac informácií o MAC je možné nájsť

4Dobrým zdrojom je napríklad [23].
5Napríklad [21] a [22].

napríklad v [24] a [25].

DAC (*Discretionaly access control*) – voliteľné riadenie prístupu. Ide o veľmi flexibilný spôsob riadenia prístupu, kde používateľ má možnosť prideľovať a odoberať oprávnenia objektov, ktorých je vlastníkom. Existuje aj možnosť delegácie prideľovania oprávnení vzťahujúcich sa na vlastné objekty na iných používateľov. Títo používatelia môžu potom prideľovať oprávnenia aj na objekty, ktorých nie sú vlastníkmi. Ďalšie informácie je možné nájsť v [26].

RBAC[6] (*Role based access control*) – riadenie prístupu založené na rolách. Používateľom nie sú prideľované jednotlivé oprávnenia, ale roly, ktoré sú pomenované množiny oprávnení, používateľov a rôl (je teda možné vytvárať hierarchie). Roly často mapujú skutočné funkcie používateľov v organizácii. Napríklad môžu existovať roly ako MANAGER, TELLER (pokladník v banke), CLIENT a podobne. RBAC umožňuje rýchlejšiu a prehľadnejšiu administráciu. Napríklad, je veľká pravdepodobnosť, že pri prideľovaní roly používateľovi, už táto rola existuje. Prideľovanie a odoberanie roly je rýchlejšie ako prideľovanie a odoberanie jednotlivých oprávnení, teda je možné aj jednoducho zmeniť používateľovi funkciu (napríklad z TELLER na MANAGER). Roly môžu mať prirodzené mená, čím sa sprehľadňuje administrácia a je možné vytvoriť dobre zrozumiteľnú hierarchiu rôl. Dôležitou vlastnosťou RBAC je možnosť statického vylúčenia zodpovednosti (SSD, *Static Separation of Duty*) a dynamického vylúčenia zodpovednosti (DSD, *Dynamic Separation of Duty*).

- Dve roly sa staticky vylučujú, ak ich používateľ nemôže mať priradené obe súčasne. Ilustračným príkladom môžu byť roly TELLER (pokladník v banke) a CLIENT. TELLER by nemal mať možnosť sám sebe vyplatiť peniaze. SSD nám dáva možnosť zakázať priradenie roly CLIENT používateľovi, ak už zastáva rolu TELLER, a naopak zakázať priradenie roly TELLER, ak už zastáva rolu CLIENT.
- Dve roly sa dynamicky vylučujú, ak ich používateľ nemôže mať aktívne obe súčasne. V predchádzajúcom príklade by TELLER nemohol byť klientom banky, v ktorej pracuje, čo je príliš obmedzujúca podmienka. DSD dáva možnosť povoliť, aby mohol byť používateľ aj pokladníkom (TELLER) aj klientom (CLIENT), nie však súčasne.

Podrobnejšie informácie o RBAC je možné nájsť napríklad v [28] a [27].

6 Uvažujeme ANSI štandard vytvorený organizáciou NIST[27].

V RDBMS sa na riadenie prístupu k databázovým objektom najčastejšie využíva DAC, ktorý je implementovaný pomocou jazyka SQL. Štandard SQL 2003 ([29], [30]) definuje príkazy `GRANT` a `REVOKE`, ktoré umožňujú DAC.

Príkaz `GRANT` umožňuje vlastníkovi objektu prideliť oprávnenie na databázový objekt inému používateľovi, role, alebo skupine `PUBLIC`, ktorá reprezentuje všetkých používateľov. Príkaz `GRANT` môže obsahovať klauzulu `WITH GRANT OPTION`. V tomto prípade môže používateľ, ktorému prideľujeme oprávnenie, prideliť toto oprávnenie iným používateľom.

Príkaz `REVOKE` je sprava inverzným príkazom k príkazu `GRANT`. To znamená, že umožňuje odobrať pridelené oprávnenie. Neumožňuje však explicitne oprávnenie zakázať.

Štandard SQL 2003 ďalej čiastočne umožňuje RBAC, keďže v [30] definuje roly. Rola, podľa SQL 2003, je množina oprávnení a rôl (je možná hierarchia). Rola obsahujúca viacero rôl obsahuje všetky oprávnenia, ktoré sú jej reflexívno-tranzitívnym uzáverom. Štandard SQL 2003 sa však nezmieňuje o SSD, či DSD[7]. Pridelenie roly používateľovi sa realizuje pomocou príkazu `GRANT`, pričom je možné uviesť klauzulu `WITH ADMIN OPTION`. Používateľ, ktorému bola pridelená rola s touto klauzulou, má právo rolu meniť, odstrániť a prideliť ju inému používateľovi.

V RDBMS, ktorý implementuje DAC a RBAC podľa SQL 2003, je možné prideliť jedno oprávnenie dvomi spôsobmi. Priamo pomocou príkazu `GRANT` alebo ako súčasť roly. Pokiaľ je teda oprávnenie pridelené používateľovi obidvomi spôsobmi a odoberieme iba oprávnenie pridelené priamo, potom má tento používateľ oprávnenie, ktoré sme mu chceli odobrať, prístupné cez pridelenú rolu. Túto skutočnosť ilustrujeme na príklade:

```
GRANT SELECT ON table1 TO user1;
GRANT SELECT ON table1 TO role1;
GRANT role1 TO user1;
REVOKE SELECT ON table1 FROM user1;
```

Príklad 2.2.1: Po vykonaní tejto sekvencie príkazov, bude môcť používateľ `USER1` prehliadať záznamy v tabuľke `TABLE1`. Kompletná implementácia je dostupná v prílohe.

Možným riešením tohto problému je vytvorenie skriptu, ktorý odoberie priame duplicitné oprávnenia. Roly ostanú nezmenené.

7 Mnohé RDBMS preto SSD ani DSD neimplementujú[28]

Pozrime sa teraz bližšie na oprávnenia pre manipuláciu s dátami v databázových tabuľkách.

Oprávnenie	***Popis***
`SELECT[(col_1,...,col_n)]`	Oprávnenie umožňujúce záznamy čítať z tabuľky. V prípade, že boli špecifikované názvy stĺpcov, používateľ má prístup iba k projekcii záznamov definovanej množinou názvov stĺpcov
`UPDATE[(col_1,...,col_n)]`	Oprávnenie umožňujúce modifikovať záznamy v tabuľke. V prípade, že boli špecifikované názvy stĺpcov, používateľ môže modifikovať len tieto špecifikované stĺpce.
`INSERT[(col_1,...,col_n)]`	Oprávnenie vkladať záznamy do tabuľky. V prípade, že boli špecifikované názvy stĺpcov, používateľ môže definovať hodnoty len pre uvedené stĺpce. Ostatné stĺpce buď nemôžu mať obmedzenie `NOT NULL` alebo musia mať definovanú hodnotu pomocou klauzuly `DEFAULT`.
`DELETE`	Oprávnenie odstrániť záznamy z tabuľky.
`REFERENCES[(col_1,...,col_n)]`	Oprávnenie vytvoriť cudzí kľúč do tabuľky. V prípade, že boli špecifikované názvy stĺpcov, používateľ môže vytvoriť cudzí kľúč, odkazujúci sa iba na tieto stĺpce.
`ALL [PRIVILEGES]`	Všetky oprávnenia (bez obmedzenia na stĺpce).

Tabuľka 2.2.1: Prehľad oprávnení pre manipuláciu s dátami v databázových tabuľkách podľa SQL 2003.

Z vyššie uvedenej tabuľky je zrejmé, že súčasný autorizačný mechanizmus v jazyku SQL podľa štandardu SQL 2003 ponúka možnosť obmedziť oprávnenia pre manipuláciu s dátami na úrovni tabuliek, prípadne stĺpcov tabuliek[8].

Uvažujme teraz nasledujúci scenár:

8 Kým oprávnenia na úrovni tabuliek sú implementované vo veľkej väčšine RDBMS, oprávnenia na úrovni stĺpcov tabuliek sú implementované menším počtom výrobcov, pričom často nie sú podporované všetky uvedené príkazy (`SELECT`, `INSERT`, `UPDATE`, `REFERENCES`).

Máme informačný systém banky, pričom chceme, aby každý klient videl iba svoje transakcie, a nie transakcie iných klientov. Ďalšou požiadavkou je, že pokladník v banke môže získať prehľad transakcií klienta iba na základe identifikátora zákazníka.

V tomto scenári je potrebná autorizácia na časť tabuľky, konkrétne autorizácia na úrovni jednotlivých riadkov tabuľky. Prístup na úrovni riadkov a stĺpcov tabuľky sa nazýva jemne štruktúrované riadenie prístupu (*FGAC, Fine-Grained Access Control*). Skúmali sme možnosti implementácie FGAC pomocou konštrukcií definovaných v štandarde SQL 2003. Výsledkom nášho skúmania bolo zistenie, že na implementáciu FGAC je možné využiť pohľady. SQL 2003 navyše umožňuje vytvoriť pohľad s klauzulou `WITH CHECK OPTION`, pričom ďalej špecifikuje, že ak uvedieme túto klauzulu, potom by mal byť pohľad modifikovateľný[9]. Používateľ môže vložiť alebo modifikovať údaje v tomto pohľade iba v tom prípade, že pre modifikovaný, respektíve nový riadok bude splnená podmienka definovaná vo `WHERE` klauzule. Modifikovateľný pohľad je pohľad, ktorý umožňuje zmeniť údaje v zdrojovej tabuľke.

```
CREATE VIEW my_transaction AS
  SELECT transaction_id, client_id, note
  FROM transaction
  WHERE client_id = 10
WITH CHECK OPTION
```

Príklad 2.2.2: Príklad modifikovateľného pohľadu. Kompletná implementácia s ilustráciou vkladania, modifikovania a odstránenia riadkov je dostupná v prílohe.

V rámci skúmania možností implementácie FGAC sme našli nasledujúce spôsoby, pomocou ktorých je možné implementovať FGAC:

1. Implementácia v aplikačnej logike. Tento prístup má viacero nevýhod:
    - Riadenie prístupu je možné obísť pomocou inej klientskej aplikácie.
    - Riadenie prístupu je potrebné zakaždým implementovať pre každé nové používateľské rozhranie.
    - Viac aplikačného kódu.
    - Každá zmena v logike riadenia prístupu znamená zmenu vo všetkých klientských aplikáciách.
2. Vytvorenie zoznamu autorizovaných používateľov (ACL, *access control list*)

9 Pohľad môže byť modifikovateľný, aj keď neuvedieme túto klauzulu

pre každý riadok tabuľky. Nevýhody:

- Komplikované vytváranie zoznamu.
- Pre väčšie množstvo používateľov je tento prístup nepraktický.
- Slabšia výkonnosť.

ACL je špeciálnym prípadom Maticového riadenia prístupu (*Access Matrix Model*)[31].

3. Implementácia MAC na úrovni tabuľky. Nevýhody:
   - Nevhodné pre popísaný scenár (FGAC pre veľký počet používateľov), museli by sme vytvoriť triedu dôveryhodnosti pre každého používateľa.
4. Vytvorenie pohľadu pre každého používateľa a operáciu.
   - Veľké množstvo pohľadov. Pri veľkom počte používateľov je možné počet pohľadov výrazne znížiť parametrizovaním pohľadov, pokiaľ to daný RDBMS umožňuje. Ďalšie zníženie počtu pohľadov je možné zlúčením pohľadov pre jednotlivé operácie do jedného pohľadu, pokiaľ je to možné (musia mať rovnakú podmienku vo `WHERE` klauzule).
   - Nie každý RDBMS umožňuje vytvárať modifikovateľné pohľady.

Skúmali sme niekoľko rôznych teoretických modelov umožňujúcich FGAC ako Maticové riadenie prístupu (*Access Matrix Model*) [31], Flexibilný autorizačný systém (*Flexibile Authorization Framework*)[32] a Viacúrovňové relácie (M*ultilevel Relations*) [33]. Najviac nás však zaujali dva modely, ktoré definoval Rizvi v [34], pretože využívajú SQL Framework[29] a z toho dôvodu sa aj dobre mapujú do RDBMS.

Rizvi vo svojich modeloch[34] využíva parametrizovaný pohľad, ktorý nazýva autorizovaný pohľad (*authorized view*). Autorizovaný pohľad je ako iný pohľad v databáze, a používateľovi môže byť teda pridelené právo na tento pohľad. Používateľ má pridelenú množinu usporiadaných dvojíc (parameter, hodnota), pomocou ktorej sa inštancujú parametrizované pohľady. Používateľ nemôže meniť hodnoty v tejto množine.

Nasledujúci príklad definuje autorizovaný pohľad, pomocou ktorého môže každý klient zistiť svoje transakcie.

```
CREATE AUTHORIZATION VIEW my_transaction AS
```

```
  SELECT *
  FROM transaction
  WHERE client_id = $client_id
```
Príklad 2.2.3: Autorizovaný pohľad, $client_id je parameter.

Ďalším typom parametrizovaného pohľadu je vzorový pohľad (*access pattern view*), ktorý sa od autorizovaného pohľadu líši tým, že používateľ musí určiť hodnotu parametra. Typický príklad využitia vzorových pohľadov je webový formulár s povinným vstupom.

Nasledujúci príklad definuje pohľad, pomocou ktorého je možné zistiť transakcie ktoréhokoľvek klienta. pokiaľ vieme jeho identifikačné číslo. Tento pohľad môže využiť pokladník banky, aby získal informácie o transakciách daného klienta.

```
CREATE AUTHORIZATION VIEW client_transaction AS
  SELECT *
  FROM transaction
  WHERE client_id = $$client_id
```
Príklad 2.2.4: Vzorový pohľad, $$client_id je ľubovoľné id klienta.

Aby sme mohli údaje aj vkladať, modifikovať a odstrániť, sú definované ďalšie tri príkazy: `AUTHORIZE INSERT`, `AUTHORIZE UPDATE` a `AUTHORIZE DELETE`.

V nasledujúcom príklade si môže klient vytvoriť preddefinovanú platbu, v tomto zázname môže zmeniť položku account_no (číslo účtu) a položku variable_symbol (variabilný symbol) a amount (suma) . Prípadne môže svoju preddefinovanú platbu odstrániť.

```
AUTHORIZE INSERT ON predefined_payment
  WHERE client_id=$client_id;
AUTHORIZE UPDATE ON predefined_payment(account_no, variable_symbol,
amount)
  WHERE client_id=$client_id;
AUTHORIZE DELETE ON predefined_payment
  WHERE client_id=$client_id;
```
Príklad 2.2.5: Použitie príkazov `AUTHORIZE INSERT`, `AUTHORIZE UPDATE` a `AUTHORIZE DELETE`

Zrejmá možnosť ako vynútiť používanie autorizovaných pohľadov je, že nepridelíme práva pre pôvodné relácie, ale iba pre autorizované pohľady. Rôzni používatelia (prípadne skupiny používateľov) môžu mať rôznu množinu rôznych pohľadov.

Ukážeme, ako je tento prístup možné implementovať pomocou konštrukcií definovaných

v SQL 2003.

```
CREATE VIEW predefined_payment_view AS
  SELECT *
  FROM predefined_payment
  WHERE client_id=immutable_parameter('client_id');
WITH CHECK OPTION;

GRANT SELECT ON predefined_payment_view TO client
GRANT INSERT ON predefined_payment_view TO client;
GRANT UPDATE (account_no, variable_symbol, amount) ON
predefined_payment_view TO client;
GRANT DELETE ON predefined_payment_view TO client;
```

Príklad 2.2.6: Implementácia FGAC pomocou SQL 2003. Autorizované pohľady.

Vo vyššie uvedenom príklade sme používateľovi „client" neumožnili pristupovať do tabuľky `PREDEFINED_PAYMENT`, ale vytvorili sme modifikovateľný pohľad nad touto tabuľkou. V pohľade je použitá funkcia `immutable_parameter`. Predpokladáme, že táto funkcia vracia pre dané meno parametra hodnotu parametra, prislúchajúcu používateľovi, ktorý dopyt spúšťa. Implementácia tejto funkcie závisí od konkrétneho RDBMS. Pohľad limituje množinu záznamov pre daného používateľa iba na jeho vlastné záznamy. Používateľ môže teda vložiť, modifikovať, odstrániť a prezerať iba taký záznam, ktorý obsahuje jeho identifikátor, pričom modifikovať sme mu umožnili iba stĺpce `ACCOUNT_NO`, `VARIALBE_SYMBOL` a `AMOUNT`.

Podobným spôsobom je možné implementovať vzorové pohľady. Jediný rozdiel je v tom, že umožníme používateľovi definovať návratovú hodnotu, pre príslušnú funkciu.

```
CREATE VIEW client_transaction AS
  SELECT *
  FROM transaction
  WHERE client_id=mutable_parameter('client_id');
WITH CHECK OPTION;

GRANT SELECT ON client_transaction TO teller;
...
EXECUTE set_mutable_parameter('client_id', 10);
SELECT *
FROM client_transaction;
```

Príklad 2.2.7: Implementácia FGAC pomocou SQL 2003. Vzorové pohľady.

Vyššie uvedený príklad je implementáciou vzorového pohľadu pomocou SQL 2003. V pohľade sa používa funkcia `mutable_parameter`. Predpokladáme že táto funkcia

vracia pre dané meno parametra hodnotu parametra, prislúchajúcu používateľovi, ktorý dopyt spúšťa. Implementácia tejto funkcie závisí od konkrétneho RDBMS. Hodnotu parametra môže používateľ nastaviť pomocou funkcie `set_mutable_parameter`. Tento pohľad teda umožňuje zobraziť informácie o transakciách ktoréhokoľvek klienta, pričom je však potrebné zadať identifikačné číslo klienta.

Alternatívnym prístupom vynucujúcim použitie autorizovaných pohľadov je umožniť dopytovať pôvodné relácie, pričom buď nahradíme v dopyte pôvodnú reláciu autorizovaným pohľadom (Trumanovský model), alebo evalvujeme oprávnenosť vykonať dopyt (Netrumanovský model).

Implementácia FGAC v Oracle prostredníctvom VPD, ktorá modifikuje dopyty pridaním dodatočných predikátov, spadá teda pod Trumanovský model. Tento prístup má však niekoľko vedľajších efektov. Problematike FGAC v Oracle je venovaná samostatná kapitola.

Trumanovský model (*Truman model*) - tento model zovšeobecňuje prístup využívajúci modifikáciu dopytov, pričom využíva parametrizované pohľady. Ideou tohto modelu je poskytnúť každému používateľovi vlastný pohľad na celú databázu. Dopyty používateľov sú transparentne modifikované, aby im bolo umožnené vidieť iba dáta z ich vlastného pohľadu na databázu. V Trumanovskom modeli sa teda zadefinuje množina parametrizovaných pohľadov pre každú reláciu v databáze. Dopyt používateľa je potom modifikovaný tak, že sa nahradí každá relácia v tomto dopyte prislúchajúcim parametrizovaným pohľadom. Parametrizované pohľady sú inštancované pred spustením dopytu.

Obmedzenia tohto modelu pramenia z transparentnej modifikácie dopytu. Vychádzajme z predchádzajúcich príkladov a predstavme si, že máme v našom bankovom informačnom systéme tabuľku `FOND(CLIENT_ID, FOND_ID, AMOUNT)`, ktorá obsahuje investície klientov do podielových fondov. Ďalej si predstavme, že chceme umožniť klientovi pozrieť si, aké množstvo peňažných prostriedkov je v jednotlivých fondoch

```
SELECT fond_id, sum(amount)
FROM fond
GROUP BY fond_id
```

k tabuľke fond prislúcha nasledujúci autorizovaný pohľad.

```
CREATE AUTHORIZATION VIEW my_fond AS
```

```sql
SELECT *
FROM fond
WHERE client_id=$client_id
```

Z tohto dôvodu je pôvodný dopyt prepísaný na

```sql
SELECT fond_id, sum(amount)
FROM my_fond
GROUP BY fond_id
```

čím dostaneme množstvo peňažných prostriedkov klienta, ktorý zadával dopyt v jednotlivých fondoch, čo sme pôvodne nechceli.

Ďalšou nevýhodou je, že ak je v autorizovaných pohľadoch implementovaná zložitejšia bezpečnostná politika, exekučné vlastnosti pôvodného a modifikovaného dopytu môžu byť veľmi odlišné, pričom sa môžu vyskytnúť redundantné predikáty (napríklad ak bezpečnostná politika a pôvodný dopyt obsahujú rovnaký predikát).

Skúmali sme možnosť implementácie tohto modelu pomocou SQL 2003. Zistili sme však, že tento štandard neposkytuje mechanizmus na prepisovanie dopytov. SQL 2003 síce definuje spúšťač (*trigger*), čo je databázový objekt, obsahujúci algoritmus, ktorý sa vykoná pred alebo po určitej udalosti, neumožňuje však vykonať algoritmus namiesto tejto udalosti[10]. Udalosť je `INSERT`, `UPDATE` alebo `DELETE`.

Netrumanovský model (*Non-Truman model*)

Používateľ vlastní množinu autorizovaných pohľadov. V tomto modeli je dopyt najprv testovaný na korektnosť voči množine autorizovaných pohľadov, pričom ak je tento test neúspešný dopyt je zamietnutý a používateľ je o zamietnutí notifikovaný (výnimka). Ak je test úspešný dopyt sa bez modifikácie vykoná. Test je úspešný práve vtedy, ak je tento dopyt podmienečne alebo nepodmienečne správny. Definície podmienečnej a nepodmienečnej správnosti sú obsiahle a je ich možné nájsť v [34], kde sú zároveň definované aj inferenčné (odvodzovacie) pravidlá umožňujúce dokázať podmienečnú, prípadne nepodmienečnú správnosť pre niektoré dopyty. Vo všeobecnosti sú však nepodmienečná, resp. podmienečná správnosť NP-úplný problém[11]. A správnosť dopytov je rozhodnuteľná iba na niektoré triedy dopytov (napríklad nepodmienečná správnosť je rozhodnuteľná pre konjunktívne dopyty[12]). Dopyty, pre ktoré nie je rozhodnuteľné, či sú

10 Niektoré RDBMS (napríklad Oracle) umožňujú vykonanie algoritmu aj namiesto udalosti, tzv. `INSTEAD OF` spúšťače. Iné RDBMS môžu poskytovať vlastné implementácie prepisovania dopytov. Napríklad v PostgreSQL definuje príkaz `RULE`.

11 Dôkaz je možné nájsť v [35]

12 Dopyty, v ktorých sa vo `WHERE` klauzule nachádza iba logická spojka `AND`.

podmienečne alebo nepodmienečne správne, môžeme prepísať pomocou autorizovaných pohľadov. Tieto dopyty budú potom triviálne nepodmienečne správne, nevyužívame však silu inferenčného systému a vytváranie pohľadov znamená prácu navyše.

2.3 Šifrovanie dát v DBMS

Pri vytváraní tejto kapitoly sme čerpali informácie z nasledujúcich zdrojov [1], [5], [7] a [36].

Šifrovanie sa najčastejšie využíva pri komunikácii, kde jeho hlavnou úlohou je zabezpečenie nezabezpečeného komunikačného kanála. Tradičný spôsob šifrovania komunikácie má však len málo charakteristík, ktoré sa dajú aplikovať na DBMS. Šifrovanie komunikácie má tradične dvoch účastníkov – odosielateľa dát, prijímateľa dát. Naproti tomu DBMS umožňuje riadenie prístupu k dátam. Viacero používateľov môže mať prístup k tým istým dátam a teda aj potrebu dešifrovania týchto dát. Kým pre každú komunikáciu je možné vygenerovať nový tajný kľúč (*short-term session key*), ktorý sa po ukončení komunikácie zahodí, v DBMS potrebujeme po zašifrovaní dát, tajný kľúč uschovať, aby sme boli v budúcnosti schopní tieto dáta dešifrovať. Pokiaľ chceme kľúče uložiť v DBMS, do úvahy musíme brať aj DBA, ktorí majú prístup ku všetkým objektom v DBMS. Úspech ochrany dôvernosti dát pomocou šifrovania záleží na tom, či sme schopní ochrániť kľúč.

Kým pri komunikácii prostredníctvom Internetu je dôvod k šifrovaniu zrejmý, v DBMS však tomu tak nie je. Predstavme si nasledujúci príklad. V našej organizácii máme tabuľku zamestnancov `EMP`, ktorá obsahuje citlivé informácie, takže vznikla nasledujúca politika:

Dáta v tabuľke `EMP` sú citlivé a prístup k nej môžu mať len autorizovaní používatelia. Autorizovaní používatelia sú DBA a používateľ, ktorého používateľské meno je v stĺpci `ENAME` v tejto tabuľke. Riadenie prístupu musí byť na úrovni používateľov.

Tieto požiadavky môže veľká časť DBMS splniť aj bez použitia šifrovania, pomocou autentifikačných a autorizačných mechanizmov, prípadne môžeme navyše nastaviť sledovanie operácie `SELECT` nad touto tabuľkou. Keby sme použili šifrovanie, každý autorizovaný používateľ by musel údaje ešte dešifrovať. Neautorizovaný používateľ by nemal prístup k údajom, pre ktoré nebol autorizovaný nezávisle od toho, či by boli v šifrovanej alebo nešifrovanej podobe.

Nie je dobrým zvykom dovoliť pristupovať používateľom k citlivým dátam aj pokiaľ sú tieto dáta šifrované, teda riešiť problém autorizácie šifrovaním.

Šifrovanie skrýva dáta nielen pred používateľmi, ale aj pred DBMS. Nad šifrovanými dátami nie je možné efektívne vykonávať porovnávania, výpočty, triedenie, a iné základné operácie. Navyše potrebuje chrániť kľúč v tajnosti a distribuovať tento kľúč jednotlivým používateľom. Musíme riešiť aj problém archivácie kľúča, pretože inak by sme v prípade jeho straty pravdepodobne trvalo stratili aj údaje chránené týmto kľúčom.

Medzi časté požiadavky patrí ochrana citlivých údajov (napríklad tabuľka miezd zamestnancov) pred DBA. Keďže DBA má prístup ku všetkým objektom v databáze, jednou[13] z možností je využitie šifrovania. Problém ochrany citlivých informácií môžeme transformovať na problém ochrany šifrového kľúča pred DBA, ktorému sa budeme venovať v nasledujúcej kapitole. Vhodné je segregovať oprávnenia DBA. Môžeme napríklad vytvoriť rolu DBA_JR, ktorá bude predstavovať junior DBA, a bude obsahovať len niektoré oprávnenia DBA.

Aj napriek týmto skutočnostiam existujú situácie, kde je šifrovanie vhodné, či nutné použiť. Šifrovanie údajov v databázových tabuľkách je účinnou a vhodnou implementáciou politiky, ktorá nás núti chrániť citlivé informácie v databázových súboroch a zálohách. Šifrovanie dát môže byť aj vynútené zákonom alebo reguláciou.

V nasledujúcom texte budeme často používať pojem transparentné šifrovanie dát (TDE). TDE umožňuje automatické šifrovanie a dešifrovanie dát. Dáta sú uložené na médiu v šifrovanej podobe, pri používaní príkazov `SELECT`, `INSERT`, `UPDATE`, `DELETE` sa automaticky podľa potreby šifrujú, či dešifrujú. TDE je primárne určená na zabezpečenie dát na médiu, keďže pre autorizovaného používateľa sa javia tieto dáta ako nešifrované (transparentne sa dešifrujú).

2.4 Manažment kľúčov v DBMS

Medzi najproblematickejšie oblasti šifrovania patrí manažment kľúčov. Menezes v [5] uvádza nasledujúce definície:

Dohodnutie kľúča (*Key establishment*) je proces, pomocou ktorého sa tajný kľúč stane dostupným pre následné kryptografické použitie.

13 Pre niektoré databázy môžu existovať nástroje, ktoré dokážu obmedziť aj práva DBA. Napríklad pre Oracle DBMS je to Oracle Vault[36]. Nenašli sme však podobný nástroj pre iné DBMS.

Manažment kľúčov (*Key management*) je množina procesov a mechanizmov podporujúcich dohodnutie kľúča a podporu nasledujúcich vzťahov ohľadom kľúčov medzi jednotlivými entitami.

Analyzovali sme možnosti manažmentu kľúčov v kontexte DBMS, pričom sme čerpali informácie z nasledujúcich zdrojov [1], [5], [7] a [37]. Identifikovali sme nasledujúce možnosti manažmentu kľúčov:

Manažovanie kľúčov v DBMS – Existuje niekoľko rôznych spôsobov ako uložiť kľúče v DBMS. Jedným z možných spôsobov je uloženie kľúčov v chránenej tabuľke (napríklad pomocou DAC, FGAC, MAC alebo hlavného kľúča). Niektoré DBMS ponúkajú automatický manažment kľúčov, kde kľúč môže predstavovať databázový objekt. Pridelenie, či odobratie kľúča používateľovi spočíva vo využití štandardného autorizačného mechanizmu DBMS (príkazy `GRANT` a `REVOKE`). Najväčším problémom tohto riešenia je, že DBA má prístup ku všetkým databázovým objektom, môže prezerať, modifikovať, odstrániť všetky tabuľky, môže spustiť všetky úložné funkcie a procedúry.

Manažovanie kľúčov klientom – Kľúč bude vlastniť a manažovať klient (klientský počítač, aplikačný server, používateľ aplikácie). Výhodou tohto prístupu je, že kľúč nie je dostupný pre DBA. Nevýhody sú nasledujúce:

- Pokiaľ je kľúč manažovaný používateľom, je vyššia pravdepodobnosť straty kľúča, a následne aj straty dát.
- Kľúč je zdieľaný a je potrebné ho distribuovať klientom. Tento problém je možné riešiť napríklad použitím CDK (Centrum pre distribúciu kľúčov) [13].
- Je potrebné zabezpečiť archiváciu kľúčov. Aby bolo možné obnoviť v prípade potreby dáta zo záloh. Príklad: Zálohovali sme dáta, a následne nastala zmena kľúča. Ak by sme pôvodný kľúč neuchovali, stratili by sme prístup k zálohovaným dátam. Archivácia je potrebná aj v prípade, že by klient kľúč stratil.

Manažovanie kľúčov v súborovom systéme[14] – Výhodou tohto prístupu je, že kľúč nie je priamo dostupný pre DBA[15]. Pokiaľ však chceme použiť na šifrovanie funkcie DBMS, musíme nejakým spôsobom kľúč pre DBMS sprístupniť. Napríklad pomocou externej

14 Uvažujeme súborový systém, v ktorom je nainštalovaný DBMS.

15 Predpokladáme, že kľúč je uložený v súbore, ktorý nevlastní používateľ, pod ktorým je spustený DBMS a DBA nemá oprávnenie čítať tento súbor, prípadne je chránený heslom, ktoré DBA nepozná.

procedúry alebo tým, že umožníme čítanie súboru. DBA teda bude mať prístup ku kľúču po jeho načítaní do DBMS.

Výber spôsobu manažmentu kľúčov závisí od našich konkrétnych požiadaviek a neexistuje jediné najlepšie riešenie. Pri výbere musíme zvážiť výkonnosť, udržiavateľnosť, možnosti zálohovania, archivácie, obnovy, zmeny kľúča.

2.5 Problémy a možnosti riešenia bezpečnosti v DBMS

V tejto kapitole poukážeme na problémy, ktoré sme v rámci analýzy identifikovali, a ktoré je potrebné riešiť pri zavádzaní a udržiavaní bezpečnosti v DBMS, informácie sme čerpali hlavne z [6], [38], [7], [10], [39], [25], [37], [40], [26].

Riadenie prístupu k údajom – Tomuto problému sme sa už vyčerpávajúco venovali v kapitole 2.2, kde sme popísali možnosti riešenia tohto problému v DBMS. Jedným z problémov, ktoré riadenie prístupu prináša je manažment oprávnení. V súčasnej dobe sa prevažne využíva na riadenie prístupu DAC. V prípade, že máme v DBMS veľký počet používateľov a oprávnení, sa stáva DAC neprehľadným a ťažko manažovateľným. V takejto situácii je výhodné použiť RBAC, teda navrhnúť roly, či prípadne hierarchie rôl. Štandard SQL 2003 definuje v [41] informačnú schému, v rámci ktorej sú zadefinované aj pohľady obsahujúce informácie o pridelených oprávneniach. Je teda možné vytvoriť program, ktorý zanalyzuje pridelené oprávnenia, a na základe tejto analýzy navrhne vytvorenie vhodných rôl, prípadne hierarchie rôl. Implementáciou takéhoto algoritmu v DBMS Oracle sa zaoberal Sýkora v [42].

Ochrana databázových súborov – Na dáta, s ktorými v DBMS pracujeme, zväčša kladieme požiadavku perzistencie. To znamená, že sa musia uložiť na nejaké médium (väčšinou pevný disk). Ďalšou bežnou požiadavkou je zálohovanie dát. Taktiež tieto dáta musíme uložiť na nejaké médium (CD, DVD, magnetické pásky, atď). Dôverné dáta na týchto médiach musia byť chránené. Existuje viacero možností riešenia tohto problému založených na šifrovaní. Šifrovanie môže prebiehať na aplikačnej úrovni, v DBMS (TDE, šifrovacie funkcie DBMS), na úrovni OS (šifrovanie diskov).

Ochrana integrity – Dnešné najpoužívanejšie DBMS[16] majú určité spoločné črty s OS. V obidvoch existujú používatelia, procesy, plánované úlohy (*job*), spustiteľný kód

16 Oracle, SQL Server, Sybase, PostgreSQL, MySQL..

(v DBMS sú to pohľady[17] a úložné procedúry, funkcie, prípadne balíky). Nie je preto prekvapujúce, že je možné vytvoriť vírusy, červy, a iný škodlivý kód aj pre DBMS[37]. Tento škodlivý kód využíva často chyby DBMS, prípadne môže byť zakomponovaný do spustiteľných súborov patriacich DBMS. V snahe zakryť svoju aktivitu (napríklad prítomnosť nového databázového používateľa) môže modifikovať informačnú schému[41]. Na ochranu integrity je potrebné aplikovať najnovšie opravné balíky, ktoré odstraňujú chyby DBMS[18], tiež je vhodné vytvárať pomocou hešovacích funkcií kontrolné sumy databázových objektov[19] a spustiteľných súborov[20].

Konfigurácia DBMS – Častým problémom je prednastavená benevolentná konfigurácia DBMS. Najčastejšie je to existencia implicitných používateľov s prednastaveným heslom, benevolentné nastavenie parametrov, oprávnenia pridelené skupine `PUBLIC`, ktoré pre používateľa nie sú potrebné. Nesprávna konfigurácia a aktivácia bezpečnostných prvkov, ktoré DBMS ponúka, môže navodiť falošný pocit bezpečnosti. DBMS väčšinou poskytujú vo svojej dokumentácii návod, akým spôsobom je potrebné DBMS inštalovať a konfigurovať.

Hrozby spojené s heslami – Okrem problému s príliš prísnou politikou, ktorý sme uviedli v kapitole 2.1.1, sa v súvislosti s heslami vyskytuje aj iný problém. Používatelia si často musia pamätať heslá pre viacero aplikácií, databázových používateľov. Ak je manažment hesiel v ich réžii, veľká časť používateľov si zvolí jeden z nasledovných scenárov:

- zvolia si jednoduché heslá ako meno, fiktívnu postavu alebo slovo, ktoré je možné nájsť v slovníku. Tieto heslá sa dajú ľahko prelomiť pomocou slovníkových útokov.
- zvolia si rovnaké heslo pre všetky systémy, prípadne sa heslo líši len v malom počte znakov a ostatné heslá je teda možné ľahko odvodiť.
- použijú zložité heslá, ktoré si niekde zapíšu.

Existuje viacero možných riešení. Jedným z nich je zvoliť vhodnú politiku pre zložitosť hesiel a kontrolovať túto politiku pomocou používateľských profilov, pokiaľ to DBMS umožňuje. Ďalšou možnosťou je obmedzenie počtu hesiel tým, že si vyberieme iný

17 Pohľad, môže obsahovať funkciu.
18 Záplaty však môžu aj zaviesť novú chybu.
19 Pre niektoré DBMS existujú na tento účel nástroje. Napríklad pre Oracle je to nástroj Repscan[37].
20 Tento problém rieši napríklad nástroj Tripwire[43].

spôsob autentifikácie. Môžeme si zvoliť autentifikáciu pomocou SSO (Kerberos[44], DCE[45], LDAP[23], atď.), čím znížime aj nároky na administráciu, na druhej strane, ak získa útočník identitu nejakého používateľa, bude mať prístup do všetkých systémov, pre ktoré je používateľ autorizovaný. Ďalším možným spôsobom riešenia problémov spojených s heslami je použitie iného typu autentifikácie, napríklad autentifikácie pomocou certifikátov, biometrie, tokenov a podobne.

Nedostatok zodpovednosti – Používatelia DBMS musia niesť zodpovednosť za svoju činnosť. Musí teda existovať spôsob, ako činnosť jednotlivých používateľov sledovať. Niektoré súčasné DBMS na riešenie problému nedostatku zodpovednosti ponúkajú možnosť sledovania jednotlivých príkazov SQL. Možnosti sledovania sa však v jednotlivých DBMS veľmi líšia. SQL 2003 nedefinuje žiadny príkaz umožňujúci sledovanie. Definované sú spúšťače, pomocou ktorých je možné sledovať operácie `UPDATE`, `INSERT` a `DELETE` tak, že v spúšťači bude kód, ktorý bude ukladať informácie do logu alebo tabuľky. Ak by sme ukladali tieto informácie na disk, prípadne v samostatnej transakcii do tabuľky, môžu vzniknúť záznamy, ktoré nezodpovedajú skutočnosti v prípade, že používateľ vykonal `ROLLBACK`[21]. Nie je možné sledovať operáciu `SELECT` pomocou spúšťačov, ktoré definuje SQL 2003. Príklad implementácie pomocou spúšťačov je možné nájsť napríklad v [6].

Sledovať by sa mali všetky administratívne operácie (vytvorenie, odstránenie, modifikovanie databázového používateľa a databázových objektov, pridelenie a odstránenie oprávnení a rôl, atď.). Prístup k dôverným dátam by mal byť sledovaný pre všetky operácie vrátane príkazu `SELECT`. Pri tabuľkách alebo pohľadoch, z ktorých sa často vyberajú údaje, ale modifikujú sa zriedka, by nemal byť sledovaný príkaz `SELECT`, ale skôr príkazy `UPDATE`, `DELETE` a `INSERT`. Čím je počet transakcií nad tabuľkou väčší, tým viac by sa malo obmedziť sledovanie. Postupy a politiky by mali byť vyberané tak, aby požadovaná úroveň sledovania bola minimálna. Príliš jemné sledovanie môže produkovať veľké množstvo dát (ťažko spracovateľné) a vyčerpáva systémové zdroje. Na druhej strane, pri príliš hrubom sledovaní nemusíme zachytiť dôležitú udalosť.

Ďalej si musíme stanoviť periódu kontrol (aby sme odhalili prípadné útoky)

---

21 SQL príkaz, ktorý neaplikuje na databázu príkazy vykonané od začiatku transakcie, prípadne od explicitne zadaného miesta (`SAVEPOINT`). Zároveň zruší transakciu, prípadne zruší príkazy od explicitne zadaného miesta (`SAVEPOINT`) a následne môžeme v transakcii pokračovať od tohto miesta.[30]

a odstraňovania starých záznamov (aby sme nevyčerpali systémové prostriedky), pričom staré záznamy je vhodné pred vymazaním archivovať[22]. Pre kontrolu logov je vhodné zvoliť častý interval, pričom môžeme využiť množstvo existujúcich nástrojov, ktoré umožňujú analýzu týchto logov. Napríklad DB Audit[46], LogMiner[36] a SQL Server Profiler[7]. Pokiaľ je to možné, logy zo sledovania by mali byť v OS, nie v DBMS, aby s nimi nemohol DBA manipulovať[23].

Inferencia (odvodenie) - Inferencia nastáva, pokiaľ je používateľ schopný správnej dedukcie údajov, na ktoré nemá oprávnenie, založenej iba na základe dát, pre ktoré oprávnenie má. Existujú tri typy inferencie:

1. Inferencia založená iba na údajoch databázového používateľa
2. Inferencia založená na údajoch databázového používateľa v spojení s referenčnou integritou a obmedzeniami (*constraint*).
3. Inferencia vyžadujúca okrem údajov databázového používateľa aj nejaké externé informácie.

Prvý typ inferencie je záležitosťou dizajnu dátového modelu a kontroly prístupu. Používateľ dopytuje DBMS a z údajov, pre ktoré ma autorizovaný prístup, odvodí dôvernejšie informácie, ku ktorým by prístup mať nemal. Príkladom môže byť tabuľka `EMP(ENAME, RANK)` obsahujúca meno a pozíciu zamestnanca a tabuľka `SALARY(RANK, SAL)` obsahujúca pozíciu a mzdu, pričom požadujeme aby používateľ nemal možnosť vidieť mzdy ostatných zamestnancov. Túto informáciu si však dokáže jednoducho odvodiť, pokiaľ má prístup k obidvom tabuľkám. Stačí ak vykoná operáciou `NATURAL JOIN`[24] nad týmito dvomi tabuľkami.

Druhý typ inferencie vzniká v dôsledku pokusu o narušenie referenčnej integrity alebo obmedzení. Tento prípad môže nastať pri FGAC. Príkladom môže byť tabuľka `EMP(EMP_ID, ENAME)` obsahujúca zamestnancov, tabuľka `TASK(TASK_ID, DESC)`, obsahujúca zoznam úloh a tabuľka `EMP_TASK(EMP_ID, TASK_ID)`[25], ktorá slúži na prideľovanie úloh zamestnancom, pričom chceme, aby zamestnanec mal prístup ku všetkým týmto tabuľkám, ale v tabuľke `EMP_TASK` by mal vidieť iba svoje úlohy.

22 Povinnosť archivácie môže byť vynútená zákonom.
23 Predpokladáme, že DBA nemá prístup do OS.
24 Binárna operácia, ktorá spojí dve relácie pomocou stĺpcov, ktoré majú rovnaké meno [30].
25 Primárnym kľúčom je dvojica (emp_id, task_id)

Pokiaľ môže tento zamestnanec vytvoriť tabuľku EMP_TASK2(EMP_ID, TASK_ID), kde dvojica (EMP_ID, TASK_ID) je cudzím kľúčom do tabuľky EMP_TASK, môže zistiť, aké úlohy sú pridelené jednotlivým zamestnancom jednoducho tak, že bude vkladať rôzne hodnoty do tabuľky EMP_TASK2. Ak sa vloženie podarí, tak pre vkladanú dvojicu existuje v tabuľke EMP_TASK záznam, ak nie, záznam v tabuľke EMP_TASK neexistuje.

Ako príklad pre posledný typ inferencie by nám mohol poslúžiť informačný systém nemocnice. Používateľ tohto systému, ktorý má oprávnenie vidieť k akému lekárovi pacient chodí alebo aké lieky má pacient predpísané, si môže na základe týchto informácií odvodiť akou chorobou pacient trpí, pričom k tejto informácii by prístup mať nemal. Externou informáciou je v tomto prípade znalosť špecializácie lekára, prípadne znalosť choroby, na ktorú sa daný liek používa.

Problematike inferencie sa venuje napríklad Lunt v [25] alebo Jajodia v [33].

Ochrana dôverných údajov pred DBA – Ochrana dôverných údajov pred DBA je náročná a často dokonca nemožná úloha, keďže DBA má prístup ku všetkým databázovým objektom. Jeden zo spôsobov, ktorým je možné zamedziť DBA čítať údaje je použiť šifrovanie, pričom kľúče musia byť manažované mimo dosahu DBA, teda mimo databázy. Problém však môže nastať aj keď sú kľúče manažované mimo DBMS, ale šifrovanie a dešifrovanie prebieha v DBMS. V takomto prípade musíme zistiť, či je DBA schopný zameniť funkciu, ktorou šifrujeme alebo dešifrujme, vlastnou funkciou s podobnou funkcionalitou, obsahujúcou však škodlivý kód. Ak takáto možnosť existuje musíme šifrovať a dešifrovať údaje mimo databázy, prípadne kontrolovať integritu používanej funkcie. Treba si však uvedomiť, že týmto ochránime údaje iba pred čítaním. DBA môže odstrániť záznamy z tabuľky, prípadne celú tabuľku. Môže dokonca tabuľku modifikovať. Predstavme si, že máme tabuľku zamestnancov EMP, a pred DBA chceme skryť výšku mzdy týchto zamestnancov.

```
SELECT * FROM emp;
ENAME      SAL
---------- ----------
KING       5000
FORD       3000
DBA        3000
```

Príklad 2.5.1: Tabuľka zamestnancov a ich platov.

Pri použití šifry AES[26] s 128 bitovým kľúčom by vyzerala táto tabuľka nasledovne:

```
SELECT * FROM emp;
ENAME      SAL
---------- --------------------------------
KING       CDA9DFA0C90DC97D9EAA09318545C204
FORD       C8330DC506B76B06C2EE7E1BD6BDACBF
DBA        C8330DC506B76B06C2EE7E1BD6BDACBF
```

Príklad 2.5.2: Tabuľka zamestnancov a ich platov, šifrovanie AES, 128 bitový kľúč, mód CBC, rovnaký inicializačný vektor[27].

DBA vie, že zamestnanec KING je na lepšej pozícii a má vyšší plat. Nevie síce kľúč, ktorým sa údaje šifrujú, ale aj napriek tomu môže tieto údaje zmysluplne modifikovať:

```
UPDATE emp
SET sal=(SELECT sal
         FROM emp
         WHERE ename='KING')
WHERE ename='DBA';

SELECT * FROM emp;
ENAME      SAL
---------- --------------------------------
KING       CDA9DFA0C90DC97D9EAA09318545C204
FORD       C8330DC506B76B06C2EE7E1BD6BDACBF
DBA        CDA9DFA0C90DC97D9EAA09318545C204
```

Príklad 2.5.3: Modifikácia hodnoty, bez znalosti kľúča.

Ďalší fakt, ktorý si vie DBA odvodiť z tabuľky, je ten, že zamestnanec FORD mal rovnaký plat ako on, pred tým ako ho zmenil. Riešením tohto problému je použitie rôznych inicializačných vektorov pre jednotlivé riadky, keďže pre použitý mód CBC, dávajú dva rovnaké otvorené texty šifrované rovnakým kľúčom rozdielne šifrové texty, pokiaľ sú inicializačné vektory rôzne. Inicializačný vektor nemusí byť utajený, preto ho môžeme uložiť v tabuľke. Rozšírime teda tabuľku `EMP` o ďalší stĺpec `IV`, do ktorého vygenerujeme náhodné inicializačné vektory. Dešifrujeme stĺpec `SAL`[28], na dešifrované hodnoty použijeme znovu šifru AES[29], tentokrát však pre každý riadok použijeme príslušný inicializačný vektor. Tento postup však nerieši prvý problém, DBA môže s hodnotou v stĺpci `SAL` skopírovať aj hodnotu v stĺpci IV. Vhodná kombinácia bezpečnostných prostriedkov by v tomto prípade spočívala v použití rôznych

26 Typ symetrickej šifry.
27 Reťazec rovnakej dĺžky ako dĺžka bloku. Viac podrobností [3], [13].
28 V tomto kroku ešte nepoužijeme inicializačný vektor.
29 Môžeme použiť ten istý kľúč.

inicializačných vektorov, čím zabránime inferencii, a v definovaní sledovania operácií `DELETE`, `INSERT` a `UPDATE`, čo nám umožní vyvodiť zodpovednosť voči konkrétnemu používateľovi[30].

Dostupnosť – Dôležitou požiadavkou je, aby informácie boli prístupné oprávnenej osobe kedykoľvek ich bude potrebovať. Je potrebné klásť dôraz najmä na rýchlosť zotavenia databázy z rôznych typov chýb a havárií, ktoré môžu mať za následok stratu dát. Najčastejšie typy chýb sú nasledovné:

- Chyba používateľa – Používateľ napríklad náhodou odstráni nesprávny záznam alebo množinu záznamov.
- Chyba vývojára (aplikácie) – Vývojár napíše chybný program, v dôsledku činnosti ktorého prídeme o dáta.
- Chyba DBA – Administrátor môže napríklad odstrániť tabuľku na produkčnom DBMS, pritom chcel odstrániť tabuľku na vývojovom DBMS.
- Chyba spôsobená škodlivým kódom alebo útočníkom – Útočník odstráni údaje z DBMS úmyselne.
- Technická chyba – Dôjde ku chybe hardvéru, napríklad k poruche disku.

Ak nastane niektorá z týchto situácií je dôležité mať zálohy umožňujúce zotavenie sa z chyby, a teda návrat databázy do pôvodného konzistentného stavu. Súčasné DBMS už zväčša ponúkajú nástroje na zálohovanie a zotavenie sa z chyby. Dôležité je mať plán pre zotavenie sa z chyby, aby sme v prípade chyby boli schopní správne postupovať.

Zvýšenie dostupnosti je možné dosiahnuť aj redundanciou diskov – RAID[31] a samotných DBMS, zapojením DBMS do klastrov (*cluster*). Je možné vytvoriť záložný DBMS (*standby DBMS*), na ktorý sa replikujú všetky transakcie z hlavného DBMS. V prípade chyby hlavného DBMS sa presmerujú všetky spojenia na tento záložný DBMS.

Ďalším prvkom pre zvýšenie dostupnosti sú používateľské profily, prípadne globálne nastavenia DBMS, ktoré môžu definovať počet povolených pokusov o prihlásenie, a tým zabrániť niektorým útokom typu DoS. Profily môžu definovať aj množstvo systémových prostriedkov alebo počet transakcií, ktoré môže používateľ použiť v rámci svojho

30 Opäť predpokladáme vytváranie logov v OS, pričom DBA nemá do OS prístup.

31 Redundantné pole nezávislých diskov (*redundant array of independent disks*) je systém používajúci viacero diskov na rozdeľovanie alebo replikáciu dát medzi jednotlivými diskami.

pripojenia. Používanie profilov teda môže taktiež pomôcť k zvýšeniu dostupnosti. Používateľským profilom sa budeme ešte venovať v kapitole 2.6.6.

Manažment používateľov – Pokiaľ DBMS obsahuje veľké množstvo[32] používateľov, môže sa stať menej bezpečným kvôli náročnej správe používateľov. Možným riešením je centralizovanie manažmentu používateľov, pričom je možné využiť napríklad LDAP[23]. Ďalším problémom, súvisiacim s manažmentom používateľov, je požiadavka zaručenia minimálnej množiny privilégií a správneho vytvorenia databázového používateľa. Pre tento účel je vhodné vytvoriť sadu skriptov, ktoré budú vytvárať jednotlivé typy databázových používateľov a podľa politík im nastavia príslušné oprávnenia a príslušný profil. Týmto krokom minimalizujeme možnosť chyby pri vytváraní účtu.

Manažment kľúčov – Tejto problematike sme sa venovali v kapitole 2.4.

Injektovanie do SQL dopytov (*SQL injection*) – Tento problém sa najčastejšie spája s webovými aplikáciami[33], v prípade, že vstupné parametre do dopytu vkladáme pomocou operácie zreťazenia. Injektovanie nastáva v prípade, keď sa útočník pokúsi svojím vstupným údajom zmeniť sémantiku alebo syntax pôvodného SQL príkazu. Uvedieme príklad, v ktorom útočník zmení dopyt tak, že namiesto svojich údajov zobrazí údaje o všetkých zamestnancoch.

```
"SELECT * FROM emp WHERE ename = '" + empName + "'";
"SELECT * FROM emp WHERE ename = 'a' or 1=1--'";
```

Príklad 2.5.4: Jednoduchý príklad injektovania. Útočník zadal vstup „a' or 1=1--“[34].

Pred injektovaním sa môžeme chrániť viacerými spôsobmi. Napríklad používaním viazaných premenných v aplikačnej vrstve a v databázovej vrstve. Existujú aj techniky umožňujúce detekciu injektovania, a pomôcť pri riešení tohto problému môže aj vhodná architektúra aplikácie. V aplikácii by sme sa nemali pripájať do schémy vlastníka objektov, ale delegovať potrebné operácie na inú schému a pripájať sa do tejto schémy. Týmto spôsobom vieme zmierniť následky v prípade útoku. Útočník napríklad nebude môcť odstrániť objekty, pretože nebude mať potrebné oprávnenie. Injektovaním a detekciou injekcie sa zaoberajú podrobnejšie napríklad tieto články [47], [48], [49] a [50].

---

32 Rádovo tisíce

33 Môže sa však vyskytnúť aj vo funkciách, ktoré poskytuje DBMS, pokiaľ sú implementované pomocou SQL. Príkladom môže byť DBMS Oracle, v ktorom sa vyskytlo viacero zraniteľných balíkov, pomocou ktorých môže používateľ získať oprávnenia DBA použitím injektovania[37].

34 Dve pomlčky „--“ označujú začiatok komentára.

2.6 Prehľad bezpečnostných mechanizmov vo vybraných DBMS

V súčasnej dobe existuje veľké množstvo rôznych DBMS, ktoré sa samozrejme líšia v úrovni poskytovania jednak typov bezpečnostných mechanizmov, a jednak v poskytovaných implementáciách týchto mechanizmov. Aby sme čitateľovi priblížili túto situáciu, rozhodli sme sa porovnať šesť DBMS, ktoré patria momentálne medzi najrozšírenejšie. Toto porovnanie sme vytvorili na základe našich skúseností s DBMS Oracle, PostgreSQL a MySQL, na základe technických dokumentácií k jednotlivým DBMS [38], [7], [51], [39], [8], [52] a na základe nasledujúcich zdrojov: [6], [53], [54], [9], [28]. Výsledky našej analýzy sme zhrnuli do dvoch tabuliek, ktoré môže čitateľ nájsť v prílohe 1.

Najmenej bezpečnostných mechanizmov poskytuje DBMS SQLite, ktorá dokonca neobsahuje ani žiadny autentifikačný a autorizačný mechanizmus. Keďže ide o jediný embedded DBMS v našom porovnaní, toto zistenie nie je pre nás prekvapujúce. Embedded databázy sú zväčša veľmi malé DBMS, primárne určené pre jedného používateľa. Autorizáciu v SQLite je možné riešiť iba pomocou nastavenia vlastníka a práv databázovému súboru. Pomocou komerčných rozšírení je možné šifrovať citlivé údaje, dokonca rozšírenie SQLCrypt ponúka TDE, čím sme však vyčerpali všetky možnosti SQLite, a preto v nasledujúcom texte nebudeme tento DBMS brať do úvahy.

2.6.1 Hešovanie a šifrovanie

Ako prvé sme porovnávali možnosti hešovania. Zaujímavým zistením pre nás bolo, že v Sybase nie je možné v SQL jazyku použiť žiadnu hešovaciu funkciu, čo je možné vo všetkých porovnávaných DBMS. Z hešovacích funkcií, uvedených v prílohe 1 je vhodnejšie používať hešovacie funkcie typu SHA, než funkcie MD2, MD4 a MD5, na ktoré existujú efektívnejšie útoky, napríklad [55]. Najširšiu paletu hešovacích funkcií prostredníctvom modulu pgcrypto ponúka PostgreSQL. Dokonca v prípade, že bol tento modul skompilovaný s podporou pre OpenSSL, je možné automaticky použiť všetky hešovacie funkcie, ktoré sú implementované v OpenSSL. Pokiaľ ide o neštandardné funkcie, jedinú poskytuje MySQL prostredníctvom funkcie `PASSWORD`, ktorá sa v MySQL využíva na hešovanie hesiel pre používateľov, MySQL však neodporúča používanie tejto funkcie v aplikáciách[38].

Ďalej sme zisťovali možnosti symetrického a asymetrického šifrovania v DBMS. Opäť

najviac možností prostredníctvom modulu pgcrypto ponúka PostreSQL (pokiaľ bol skompilovaný s podporou pre OpenSSL). Všetky porovnávané DBMS umožňujú šifrovanie pomocou šifry AES, ktorá je v čase písania tejto práce považovaná za dostatočne bezpečnú a patrí medzi najpoužívanejšie. Naopak, z uvedených šifier nie je odporúčané používať nasledujúce: DES, 3DES_2_KEY, RC2, ktoré DBMS poskytujú z dôvodu spätnej kompatibility.

2.6.2 Identifikácia a autentifikácia

Vo všetkých porovnávaných DBMS môže používateľ preukázať svoju identitu zadaním používateľského mena a okrem PostgreSQL aj pomocou využitia PKI. Oracle ako jediný z porovnávaných DBMS umožňuje identifikáciu pomocou biometrie prostredníctvom protokolu RADIUS[56].

Ako je vidieť z tabuliek uvedených v prílohe 1, každý z porovnávaných systémov umožňuje autentifikáciu pomocou používateľského mena a hesla a okrem MySQL aj pomocou SSO (Kerberos[44], DCE[45], LDAP[23]).

Oracle a MySQL a SQL Server navyše umožňujú autentifikáciu klienta pomocou SSL, tým, že vynútia overenie klientského certifikátu, môžeme teda využiť PKI.

Okrem MySQL a Sybase porovnávané DBMS umožňujú autentifikáciu pomocou lokálneho OS, to znamená, že ak sa vie používateľ prihlásiť do lokálneho OS, DBMS nebude od neho vyžadovať ďalšiu identifikáciu a autentifikáciu[35].

Oracle a PostgreSQL podporujú aj autentifikáciu pomocou vzdialeného OS, čo znamená, že sa nebude vyžadovať dodatočná autentifikácia ani od používateľov, ktorí sa pokúšajú prihlásiť do DBMS zo vzdialeného OS. V tomto prípade je potrebné si uvedomiť, že ak útočník získa prístup ku ktorémukoľvek OS, odkiaľ sa môže používateľ prihlásiť do DBMS, získa aj prístup do DBMS.

PostgreSQL a Sybase ponúkajú aj možnosť autentifikácie pomocou PAM [57], ktorý umožňuje integráciu rôznych autentifikačných mechanizmov (DCE, Kerberos, atď).

2.6.3 Riadenie prístupu

Všetky porovnávané DBMS umožňujú riadenie prístupu pomocou DAC prostredníctvom príkazov `GRANT` a `REVOKE`.

35 Bezpečnosť DBMS teda závisí od zabezpečenia OS.

Funkcionalita implementácií RBAC sa v porovnávaných DBMS odlišuje. Všetky DBMS implementujúce RBAC však spĺňajú predpoklady pre základný a hierarchický model NIST RBAC[27], a všetky tieto DBMS umožňujú vytvoriť hierarchiu rôl, aktivovať viacero rôl súčasne a prideliť rolám systémové i objektové oprávnenia. Jedine Sybase z porovnávaných DBMS umožňuje aj SSD a DSD[36]. SQL Server, na rozdiel od ostatných DBMS, neumožňuje používateľovi definovať množinu rôl aktívnu po prihlásení (prednastavené roly). Dôkladnejšie porovnanie RBAC je možné nájsť v [28].

FGAC je možné v DBMS Oracle, SQL Server, Sybase a MySQL implementovať pomocou modifikovateľných pohľadov. Sybase navyše definuje typ databázového objektu „pravidlo prístupu“ (access rule). Existujú dva typy pravidiel prístupu:

- AND pravidlá
- OR pravidlá

Tieto pravidlá obsahujú predikát tvorený jednou viazanou premennou a hodnotou alebo používateľskou funkciou. Do viazanej premennej sa priradí kvalifikovaný názov stĺpca (pomocou príkazu `sp_bindrule`), pričom pre jeden stĺpec môže existovať iba jedno pravidlo. V prípade, že sa tabuľka, ktorá má stĺpce, na ktoré existujú vytvorené pravidlá, vyskytne v dopyte, príslušná časť dopytu sa prepíše tak, aby tieto pravidlá obmedzovali selekciu údajov z tejto tabuľky. Pokiaľ sa na tabuľku vzťahujú AND pravidlá A_1 až A_n a OR pravidlá O_1 až O_n, selekcia z tabuľky sa obmedzí podmienkou:

A_1 AND A_1 AND … A_n AND (O_1 OR O_2 OR … O_n)

Tieto pravidlá sa však vzťahujú iba na príkaz `SELECT`, nie je možné pomocou nich obmedziť príkazy `UPDATE`, `INSERT` a `DELETE` [8].

PostgreSQL poskytuje v podobe príkazu `RULE` mechanizmus na prepisovanie dopytov. Príkaz `RULE` definuje pre tabuľku a pre udalosť[37] postupnosť príkazov[38], ktorá sa má vykonať, buď namiesto udalosti, alebo súčasne s udalosťou[51].

Problematike FGAC v Oracle sme sa venovali v kapitole 3.2, okrem modifikovateľných pohľadov je možné FGAC implementovať aj pomocou VPD a OLS.

36 Tvrdenie je čiastočne založené na porovnaní DBMS v práci [28]. Aj keď sme porovnávali novšie verzie DBMS, nenastala v spôsobe implementácie RBAC v porovnávaných systémoch žiadna zmena.
37 Možné udalosti sú `SELECT`, `INSERT`, `UPADATE` a `DELETE`.
38 Postupnosť môže byť aj prázdna.

2.6.4 Manažment kľúčov

V Sybase je kľúč reprezentovaný databázovým objektom (`ENCRYPTION KEY`). Vytvorené kľúče sú uložené v informačnej schéme v tabuľke `SYSENCRYPTKEYS`, šifrované pomocou 128 bitového kľúča, ktorý bol vygenerovaný na základe systémového hesla pre šifrovanie (*system encryption password*) a soli. Šifrovanie je možné použiť iba v rámci TDE. Vlastník kľúča môže prideliť oprávnenie `SELECT` na tento kľúč inému používateľovi pomocou príkazu `GRANT`. Oprávnenie je potrebné prideliť iba ak chceme používateľovi umožniť použitie kľúča v príkazoch `CREATE TABLE`, `ALTER TABLE` a `SELECT INTO`. Používatelia, ktorí sú autorizovaní pre operácie `INSERT`, `DELETE`, `SELECT` alebo `UPDATE` pre tabuľku, v ktorej sú šifrované stĺpce, potrebujú mať na tieto stĺpce oprávnenie `DECRYPT` aby mohli vkladať, odstraňovať, čítať alebo modifikovať údaje.

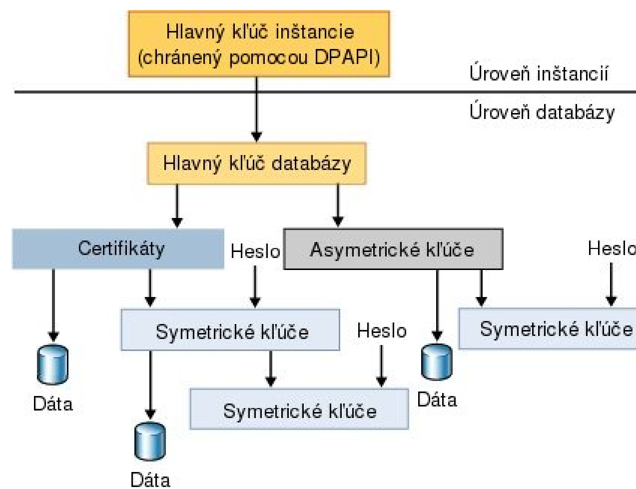
Oracle umožňuje manažment kľúčov podobne ako v Sybase len pri použití TDE. Kľúče však nie sú reprezentované databázovými objektami, sú uložené v schéme SYS v tabuľke `ENC$,` a sú zašifrované hlavným kľúčom (*master key*), ktorý sa nachádza v kľúčenke (*wallet*), čo je štruktúra uložená mimo DBMS spĺňajúca štandard PKCS #12[58], ktorý špecifikuje formát pre ukladanie a prenos kľúčov a certifikátov. Kľúčenka je chránená heslom. Pri štarte DBMS musí DBA otvoriť kľúčenku, čím sprístupní hlavný kľúč pre použitie.

V SQL Serveri, podobne ako v Sybase, sú kľúče reprezentované databázovými objektami[39]. Narozdiel od predchádzajúcich dvoch DBMS, poskytuje SQL Server najprepracovanejší manažment kľúčov.

Na obrázku 2.6.4.1 je hierarchia, ktorá znázorňuje, akým spôsobom môžu byť chránené kľúče v SQL Serveri. Každý syn v tejto hierarchii môže byť chránený (šifrovaný) svojím rodičom. Hlavný kľúč inštancie SQL Servera[40] (*service master key*) je chránený pomocou Windows Data Protection API (DPAPI). Tento kľúč je automaticky manažovaný SQL Serverom[41]. Každá databáza má najviac jeden databázový hlavný kľúč (*database master key*), ktorým sú šifrované kľúče a certifikáty v rámci databázy. Databázový hlavný kľúč musí byť chránený heslom a uložený v databáze, pre ktorú je hlavným kľúčom.

---

39 `MASTER KEY`, `ASYMMETRIC KEY`, `SYMMETRIC KEY`, `CERTIFCATE`.
40 Je to symetrický kľúč.
41 Je ho však možné zálohovať do súboru, obnoviť zo súboru a pregenerovať.



Obrázok 2.6.4.1: Manažment kľúčov v DBMS SQL Server 2005.

Databázový hlavný kľúč môže byť navyše chránený aj hlavným kľúčom inštancie a uložený v hlavnej databáze. Pomocou certifikátov a asymetrických kľúčov a symetrických kľúčov môžu byť chránené symetrické kľúče. Každý z kľúčov môže byť navyše chránený heslom, ktoré manažuje používateľ. SQL Server umožňuje ako jediný DBMS aj podpisovanie modulov certifikátom alebo verejným kľúčom (`ADD SIGNATURE`). Kľúče je možné sprístupniť, resp. znepřístupniť pomocou príkazu `OPEN MASTER KEY` a `OPEN SYMMETRIC KEY`, respektíve `CLOSE MASTER KEY` a `CLOSE SYMMETRIC KEY`.

2.6.5 Deklaratívne TDE

Deklaratívne TDE je možné z porovnávaných DBMS iba v Oracle a Sybase, pričom TDE v Oracle sme venovali samostatnú kapitolu.

V Sybase je šifrovanie stĺpca možné zapnúť pri vytváraní tabuľky (`CREATE TABLE`), alebo pomocou modifikácie tabuľky (`ALTER TABLE`). Je možné zmeniť kľúč pre TDE pre daný stĺpec tak, že vytvoríme nový kľúč (`CREATE ENCRYPTION KEY`) a nastavíme tento nový kľúč ako šifrovací pre príslušný stĺpec tabuľky (`ALTER TABLE`). Šifrované stĺpce môžu byť indexované, iba ak nebol pri vytváraní kľúča použitý inicializačný vektor a výplň (*padding*).

2.6.6 Používateľské profily

Používateľské profily sú dôležitým nástrojom umožňujúcim zvýšiť bezpečnosť DBMS, tým, že definujú rôzne obmedzenia[42] pre používateľa.

Z porovnávaných DBMS najmenej možností poskytuje PostgreSQL, ktorý dokáže obmedziť iba počet paralelných prihlásení pre jedného používateľa a nastaviť dátum exspirácie hesla. V MySQL môžeme používateľovi definovať počet dopytov za hodinu, počet modifikácii za hodinu a počet paralelných prihlásení. V Sybase môžeme používateľovi nastaviť maximálny počet neplatných pokusov o prihlásenie, minimálnu dĺžku hesla, dobu exspirácie. Trochu kurióznou možnosťou je vynútenie kontroly, či heslo obsahuje aspoň jednu číslicu. SQL Server umožňuje nastaviť heslo pre práve vytvoreného používateľa ako exspirované a máme možnosť uplatniť pre heslá komplexné politiky OS Windows na ktorom je spustený DBMS[43]. Oracle ponúka z porovnávaných DBMS najviac možností. Pre používateľa môžeme nastaviť politiky pre heslá a množstvo systémových zdrojov, ktoré môže použiť. Politiky pre heslá zahŕňajú:

- Povolený počet neúspešných pokusov o prihlásenie.
- Počet dní, počas ktorých je heslo platné.
- Počet dní, počas ktorých nie je možné použiť rovnaké heslo.
- Doba od dosiahnutia maximálneho počtu neúspešných pokusov o prihlásenie, počas ktorej bude účet uzamknutý.
- Počet dní, počas ktorých je používateľ upozornený, že heslo je potrebné zmeniť. Pokiaľ nedôjde k zmene hesla počas tejto doby, účet bude uzamknutý.
- Verifikačná funkcia pre silu hesla.

Zo systémových parametrov je možné pre databázového používateľa nastaviť počet paralelných prihlásení, procesorový čas pre jedno spojenie, procesorový čas pre jeden dopyt, dobu na spojenie, dobu nečinnosti, veľkosť pamäte, ktorú má používateľ prístupnú.

42 Obmedzenia na množstvo systémových prostriedkov, ktoré môže používateľ použiť, obmedzenia na silu hesla, atď.

43 Platí iba pre Windows Server 2003.

3 Implementácia zabezpečenia na platforme Oracle

V kapitole 2.5 sme analyzovali problémy, ktoré sa vo svete bezpečnosti DBMS vyskytujú najčastejšie, pričom sme načrtli aj možnosti riešenia týchto problémov. Tieto riešenia sú aplikovateľné aj na DBMS Oracle.

Problému zavedenia a udržiavania bezpečnosti v DBMS Oracle sa venuje niekoľko webových stránok [59], [60], [61] knižných publikácií [6], [62], [63] a samozrejme v určitej miere aj dokumentácia [39]. Každý z týchto zdrojov ponúka viac či menej vyčerpávajúci zoznam bezpečnostných politík, prípadne krokov, ktorými je možné tento DBMS zabezpečiť. Väčšinou sa tieto kroky a politiky zameriavajú na správnu inštaláciu, zabezpečenie OS a počítačovej siete, redukciu nadbytočných oprávnení, správnu konfiguráciu, zálohovanie a zotavenie sa z chyby či havárie, inštaláciu záplat a podobne. Problematike udržiavania bezpečnosti pomocou návrhu aplikácií s ohľadom na bezpečnosť je však venované podstatne menej priestoru. My sme sa teda v našej práci rozhodli sústrediť práve na túto problematiku, pričom sme si dali cieľ analyzovať možnosti využitia FGAC a šifrovania v tomto procese.

3.1 Aplikačný kontext

Aplikačný kontext je menný priestor (*namespace*) obsahujúci množinu usporiadaných dvojíc (atribút, hodnota), je uložený v pamäti, pričom je vždy zviazaný s nejakým balíkom. Len pomocou tohto balíka je možné nastaviť hodnoty jednotlivých atribútov aplikačného kontextu. Aplikačný kontext sa v dopyte správa ako viazaná premenná (*bind variable*), čím umožňuje znovupoužitie exekučného plánu dopytu[44]. Na prístup k atribútom aplikačného kontextu sa používa SQL funkcia `SYS_CONTEXT`.

Existujú dva druhy aplikačného kontextu:

1. Lokálny – Tento druh aplikačného kontextu existuje len v rámci jedného spojenia a z iných spojení nie je doňho prístup.
2. Globálny – Globálny aplikačný kontext sa nachádza v zdieľanej pamäti a je prístupný zo všetkých spojení.

---

44 Pokiaľ máme dva podobné dopyty, ktoré sa líšia len v literále vo `WHERE` klauzule, tieto dva dopyty Oracle považuje za rôzne. Existujú teda dva exekučné plány. Pokiaľ by sme namiesto literálu v týchto dopytoch použili aplikačný kontext, bude existovať len jeden, rovnaký exekučný plán pre obidva dopyty.

Pre implementáciu FGAC je vhodný lokálny aplikačný kontext, preto v ďalšom v texte budeme pod pojmom aplikačný kontext myslieť lokálny aplikačný kontext.

V prílohe 2 sme uviedli príklad, ktorý ilustruje vytvorenie a manipuláciu s aplikačným kontextom.

```
CREATE CONTEXT sample_context USING sample_context_package;

-- pred vykonanim dopytu nastavime atribut aplikacneho kontextu
EXECUTE sample_context_package.set_client_id(10);

-- vykoname dopyt
SELECT * FROM transaction
WHERE client_id=sys_context('sample_context', 'client_id');
```

Príklad 3.1.1: Vytvorenie aplikačného kontextu a príklad použitia. Celý príklad je možné nájsť v prílohe.

V Oracle existuje aplikačný kontext `USERENV`, ktorý je automaticky nastavený pri prihlásení do DBMS. Tento aplikačný kontext obsahuje atribúty[45], ktoré je možné využiť pri návrhu aplikácie.

Aplikačný kontext môžeme inicializovať dvoma spôsobmi:

1. Používateľ po prihlásení explicitne zavolá príslušné procedúry balíka, ktorý je zviazaný s aplikačným kontextom.
2. Definujeme databázový spúšťač, ktorý sa spustí po prihlásení do DBMS (*logon trigger*), pričom v definícii tohto spúšťača budú volania procedúr balíka, ktorý je zviazaný s aplikačným kontextom.

3.2 FGAC

3.2.1 Implementácia autorizovaných pohľadov

V kapitole 2.2 sme podrobnejšie definovali tri spôsoby, ktorými je možné FGAC implementovať. Prvý z týchto spôsobov spočíval vo vytvorení autorizovaných pohľadov a spridelení oprávnení na tieto pohľady namiesto oprávnení na tabuľky, pričom sme uviedli spôsob, ktorým je toto možné dosiahnuť pomocou štandardu SQL 2003 – pomocou modifikovateľných pohľadov.

V DBMS Oracle je pohľad modifikovateľný, ak spĺňa nasledujúce podmienky:

45 Sú to napríklad IP adresa, z ktorej sa používateľ pripája, údaje o spôsobe autentifikácie, meno momentálne prihláseného používateľa a mnohé ďalšie[39].

- Každý stĺpec v klauzule `SELECT` v pohľade musí zodpovedať stĺpcu v tabuľke – nemôže obsahovať napríklad skalárne dopyty[46].
- Pohľad nemôže obsahovať množinové operácie[47], operátor `DISTINCT`, agregačné a analytické funkcie, `GROUP BY`, `ORDER BY`, `CONNECT BY` a `START WITH`[48]. Tieto operácie sa však môžu nachádzať v poddopytoch vo `WHERE` klauzule.
- Až na niekoľko výnimiek pohľad nemôže obsahovať operácie typu `JOIN` v klauzule `FROM`[49]. Táto operácia je však povolená v poddopytoch vo `WHERE` klauzule – napríklad v operátore `EXISTS`.

Pohľad obsahujúci operáciu `JOIN` v klauzule `FROM` je modifikovateľný, iba ak príkaz `INSERT`, `UPDATE` alebo `DELETE` ovplyvňuje iba jednu tabuľku z tejto operácie `JOIN`, pričom nesmie byť pohľad vytvorený s klauzulou `WITH CHECK OPTION`, a teda nie je možné overiť, či používateľ modifikuje len svoje záznamy.

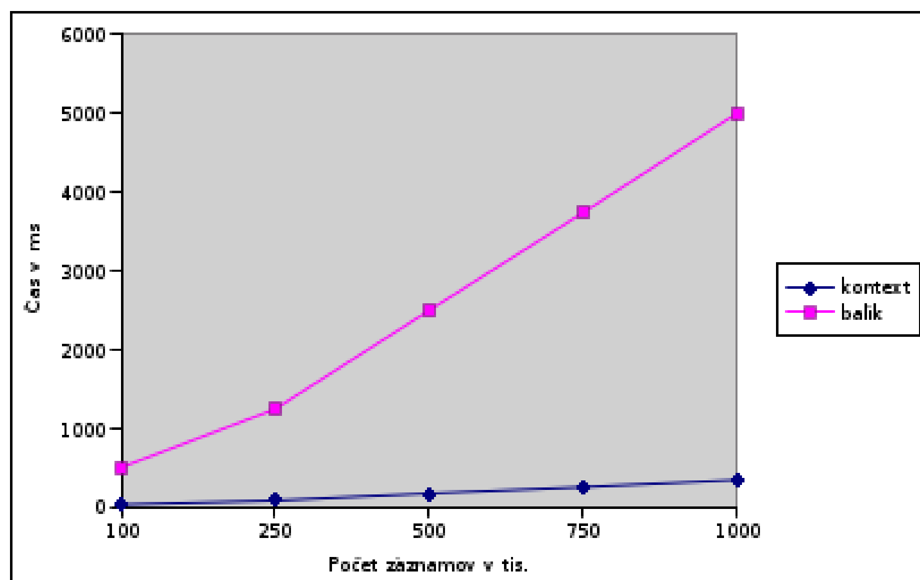
Keď už máme k dispozícii modifikovateľné pohľady, musíme nájsť ešte spôsob, ktorým implementujeme funkcie `IMMUTABLE_PARAMETER`, `MUTABLE_PARAMETER` a `SET_MUTABLE_PARAMETER`. Identifikovali sme dva priamočiare spôsoby.

Prvým z nich je implementácia pomocou aplikačného kontextu, ktorý sme si predstavili v predchádzajúcej podkapitole. Vytvoríme dva aplikačné kontexty `IMMUTABLE` a `MUTABLE`, a následne pridelíme aj oprávnenie `EXECUTE` na balík, ktorý umožňuje nastavovať aplikačný kontext `MUTABLE` používateľovi, ktorý má mať toto oprávnenie. Funkciu `IMMUTABLE_PARAMETER` a `MUTABLE_PARAMETER` nahradíme potom volaním funkcie `SYS_CONTEXT('immutable',...)`, respektíve `SYS_CONTEXT('mutable',...)`. Volanie funkcie `SET_MUTABLE_PARAMETER`, nahradíme volaním príslušnej metódy balíka, ktorý nastavuje aplikačný kontext.

Druhý spôsob spočíva v definícii dvoch balíkov `IMMUTABLE` a `MUTABLE` obsahujúcich vo svojom tele[50] množinu globálnych premenných a pre každú premennú funkciu, ktorá ju nastaví na požadovanú hodnotu, a funkciu, ktorá zistí jej hodnotu. Používateľovi

46 Dopyty v klauzule `SELECT`.
47 `UNION`, `UNION ALL`, `INTERSECT`, `MINUS`, ...
48 `CONNECT BY` a `START WITH` umožňujú vytvárať rekurzívne dopyty.
49 Za `JOIN` považujeme aj kartézsky súčin.
50 Pozorného čitateľa by mohlo napadnúť definovať premenné v špecifikácii balíka a použiť tieto premenné v pohľade. Oracle však neumožňuje použitie globalnej premennej definovanej v balíku v pohľade.

pridelíme právo `EXECUTE` na balík `MUTABLE`. Volanie funkcie `IMMUTABLE_PARAMETER('param')` nahradíme `IMMUTABLE.GETPARAM`, analogicky postupujeme aj v prípade `MUTABLE`. Tento spôsob je však mennej výkonný, pretože s každým testovaním v pohľade sa volá funkcia[51]. Vykonali sme jednoduchý výkonnostný test (zdrojový kód testu je dostupný v prílohe 2 a namerané časy v prílohe 3). V tomto teste sme vytvorili tabuľku s veľkým počtom záznamov a vytvorili sme dva pohľady nad touto tabuľkou. Jeden pohľad využíval vo `WHERE` klauzule na získanie parametra aplikačný kontext a druhý pohľad balík. Následne sme vykonali dopyt, ktorý vynútil prehľadávanie celej tabuľky, teda predikát sa aplikoval na každý riadok tabuľky.



Obrázok 3.2.1.1: Graf závislosti doby vykonávania dopytu od počtu záznamov v tabuľke a použitej metódy.

Používanie dvoch aplikačných kontextov[52] je však nepraktické, ak máme viacero používateľov alebo skupín používateľov, ktorí majú mať povolené nastavovanie rôznych množín parametrov, pretože zvyšuje počet autorizovaných pohľadov, ktoré je potrebné vytvoriť. Našli sme spôsob, ktorý nám umožní používať jeden aplikačný kontext. Riešenie spočíva v tom, že nepridelíme oprávnenie na celý balík, ktorý je zviazaný s aplikačným kontextom, ale iba na podmnožinu požadovaných funkcií. Toto docielime

51 `SYS_CONTEXT` sa líši od obyčajnej PL/SQL funkcie, správa sa ako viazaná premenná, teda vyhodnotí sa iba raz.
52 Tvrdenie platí aj pre balíky. Nasledujúci opis je taktiež aplikovateľný aj na balíky.

tak, že pre každú takúto podmnožinu funkcií definujeme samostatný balík. Tento balík bude obsahovať iba deklarácie požadovaných funkcií zhodných s deklaráciami v balíku, ktorý je zviazaný s aplikačným kontextom[53]. Definícia takejto funkcie v tele balíka bude tvorená volaním prislúchajúcej funkcie z balíka zviazaného s aplikačným kontextom. Balík obsahujúci takéto funkcie budeme nazývať proxy balík a samotné funkcie proxy funkcie. Jednotlivým používateľom teda pridelíme oprávnenie `EXECUTE` iba na ten proxy balík, ktorý obsahuje proxy funkcie nastavujúce iba tie parametre, ktoré majú právo títo používatelia nastavovať.

Keďže medzi cieľmi našej práce bolo overiť navrhnuté riešenie na príklade, rozhodli sme sa implementovať dátový model[54] pre jednoduchú modelovú aplikáciu, v rámci ktorého implementujeme FGAC pomocou autorizovaných pohľadov. V tomto príklade ukážeme aj využitie proxy balíkov.

Ako príklad sme si vybrali jednoduchý model informačného systému banky[55]. Určili sme štyri typy používateľov, ktorí budú tento systém používať:

1. Zákazník (client).
2. Pracovník v banke (teller).
3. Správca konfigurácie (admin).
4. Vzdialený terminál (terminal) – bankomat alebo terminál.

Každý zákazník môže vlastniť ľubovoľný počet bankových účtov. Existovať môže viacero typov účtov, ktoré môže vytvárať iba správca konfigurácie. V našom modeli existujú dva typy zákazníkov – biznis zákazníci a obyčajní zákazníci. Pre každý z týchto typov zákazníkov je definovaný typ pracovníka. Existujú teda dva typy pracovníkov – pracovníci pre biznis zákazníkov[56] a pracovníci pre obyčajných zákazníkov[57].

Zákazník môže mať ku každému účtu platobnú kartu. Zákazník vlastní jednu grid kartu[58]. Môže zadať platobný príkaz zo svojho účtu na iný účet, ktorého výška nesmie presiahnuť stanovený limit – platba typu „EBAN_PAY“. Môže prezerať svoje transakcie. Môže si zadefinovať preddefinované platby, ktoré mu umožnia zrýchliť zadávanie platby. Môže

53 Deklarácie nemusia byť úplne zhodné, môžu mať napríklad iné meno.
54 Nie samotnú aplikáciu.
55 Nejde o pokus simulácie reálnej funkcionality bankového systému.
56 Typ pracovníka v tabuľke je „BTELLER“, typ zákazníka v tabuľke je „B2B“
57 Typ pracovníka v tabuľke je „CTELLER“, typ zákazníka v tabuľke je „COMMON“
58 Tabuľka kódov, ktorá slúži na autentifikáciu pri zadávaní transakcie z internet bankingu.

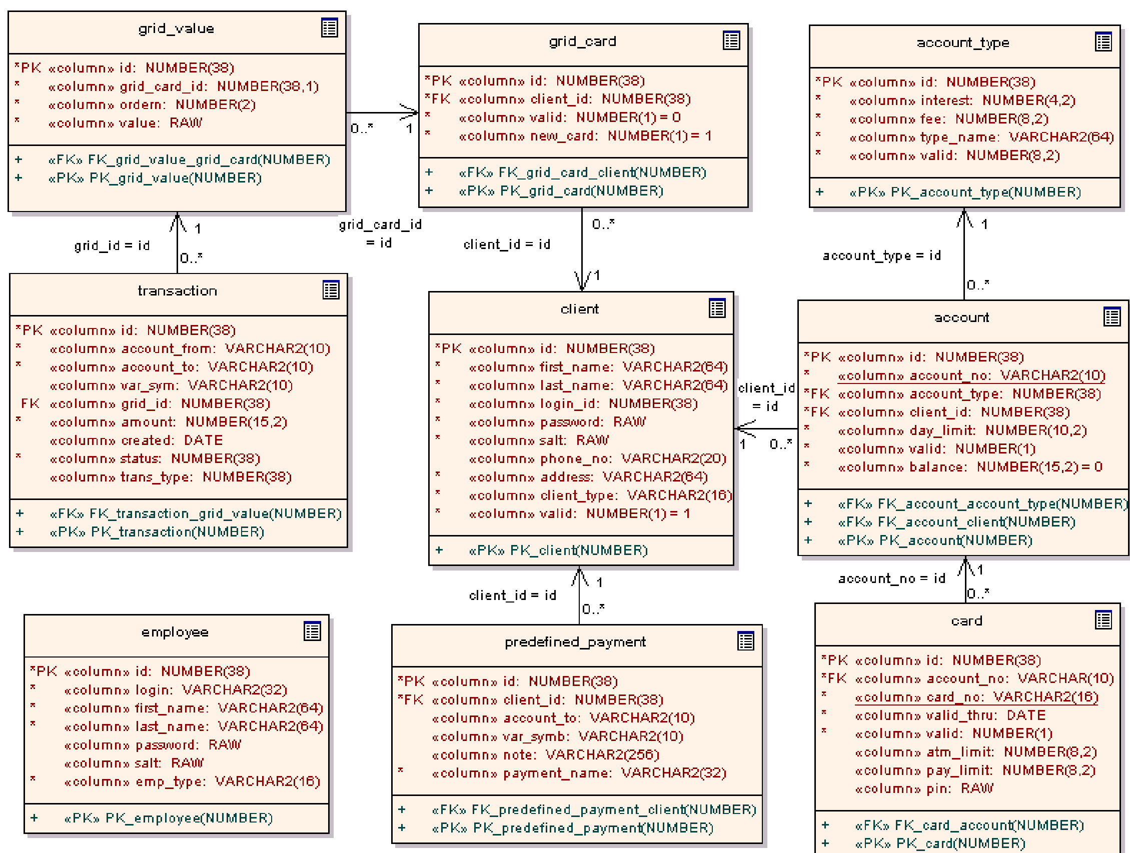
vidieť platobné karty, ktoré mu boli vydané k jednotlivým účtom. Môže zmeniť svoje telefónne číslo vo svojom profile.

Pracovník pre obyčajných zákazníkov môže vytvoriť účet pre obyčajného zákazníka. Zadaním čísla účtu obyčajného zákazníka môže prezerať informácie k tomuto účtu. Môže zadať platbu typu „TELLER_PAY“ z tohto účtu aj nad povolený limit. Môže zmeniť adresu a telefónne číslo zákazníka. Ďalej môže zrušiť používateľský aj bankový účet zákazníka, aktivovať grid kartu a platobnú kartu alebo zrušiť ich platnosť.

Pracovník pre biznis zákazníkov má dostupnú rovnakú funkcionalitu ako pracovník pre obyčajných zákazníkov, môže však operovať iba nad biznis zákazníkmi.

Správca konfigurácie môže vytvárať typy účtov, nastavovať úrok a mesačný poplatok. Správca môže navyše vytvárať zamestnanecké používateľské účty.

Vzdialený terminál alebo bankomat môže vytvoriť transakciu vzťahujúcu sa na účet, ku ktorému patrí použitá platobná karta – platby typu „CARD_ATM“ a „CARD_PAY“.

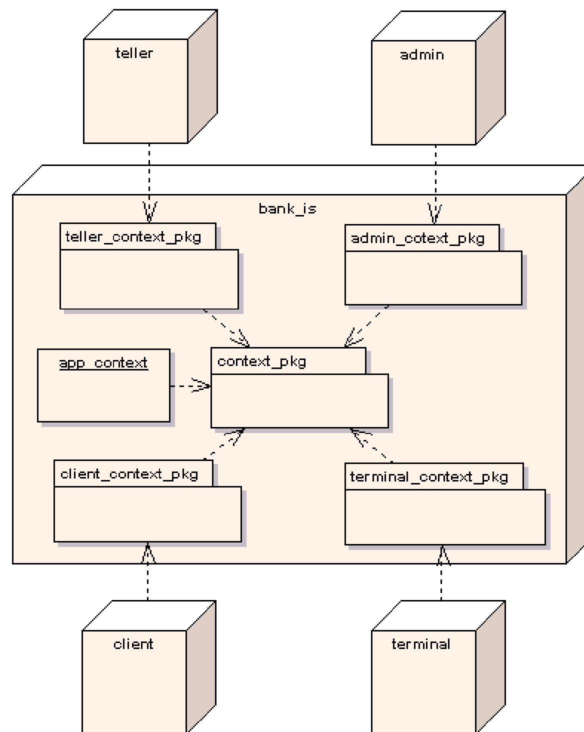


Obrázok 3.2.1.2: Dátový model modelovej aplikácie.

Na obrázku 3.2.1.2 je entitno-relačný diagram modelu, ktorý sme navrhli. Popis

jednotlivých tabuliek môže nájsť čitateľ v prílohe 4, snažili sme sa však voliť názvy tabuliek a stĺpcov tak, aby bol ich význam aj intuitívne jasný.

K tomuto dátovému modelu sme vytvorili množinu autorizovaných pohľadov. Vytvorili sme balík, ktorý nastavuje parametre aplikačného kontextu a vytvorili sme aj štyri proxy balíky obsahujúce podmnožinu proxy funkcií na balík zviazaný s aplikačným kontextom – pre každý typ používateľa jeden proxy balík.



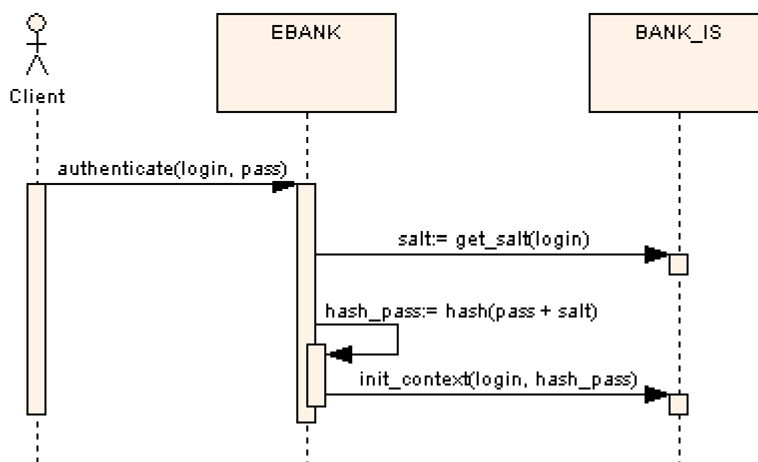
Obrázok 3.2.1.3: Architektúra modelovej aplikácie a nastavovanie aplikačného kontextu.

Obrázok 3.2.1.3 znázorňuje architektúru aplikácie, ktorá využíva náš dátový model. `BANK_IS` je schéma obsahujúca všetky tabuľky, autorizované pohľady a balíky pre nastavovanie aplikačného kontextu a aj všetky proxy balíky. `CLIENT`, `TERMINAL`, `TELLER` a `ADMIN` sú databázoví používatelia[59] – teda vlastnia rovnomennú schému. Týmto entitám sa prideľuje oprávnenie `EXECUTE` na proxy balík, ktorý im prislúcha, a taktiež im prideľujeme jednotlivé oprávnenia na autorizované pohľady. Aplikácia pre klienta sa potom pripojí do schémy `CLIENT` a využíva svoje autorizované pohľady. Pokiaľ aplikácia nepoužíva priamo autorizované pohľady, ale volanie PL/SQL funkcií, ktoré pracujú nad autorizovanými pohľadmi, odporúčame nepripájať sa do tejto schémy, ale vytvoriť ďalšiu schému, do ktorej sa bude aplikácia pripájať, a ktorej pridelíme

59 Bolo by možné aj použitie rôl.

oprávnenia EXECUTE na príslušné PL/SQL funkcie[60]. Analogicky sa správajú aplikácie pre terminál, zamestnanca banky a správcu konfigurácie.

Prostredníctvom jednotlivých proxy balíkov prebieha aj autentifikácia, v rámci ktorej sa nastaví aplikačný kontext. V našom modeli uvažujeme jednoduchú autentifikáciu pomocou identifikátora a hesla. Aplikácia, ktorá využíva schému využívajúcu autorizované pohľady, si najprv vyžiada soľ. Túto soľ zreťazí s heslom, vytvorí heš výsledku zreťazenia a tento heš pošle schéme BANK_IS, prostredníctvom volania funkcie proxy balíka. BANK_IS porovná prijatý heš s hešom, ktorý sa nachádza v príslušnej tabuľke, a na základe tohto porovnania nastaví aplikačný kontext, alebo vyhlási výnimku. Dôvod, prečo nehešujeme pomocou PL/SQL v schéme BANK_IS je uvedený neskôr v kapitole 3.3.2.



Obrázok 3.2.1.4: Autentifikácia zákazníka. Pri úspešnej autentifikácii sa nastaví aplikačný kontext.

Na obrázku 3.2.1.4 je príklad autentifikácie zákazníka, ktorý pošle aplikácii EBANK svoje prihlasovacie údaje[61]. Aplikácia EBANK využíva schému CLIENT, a teda má oprávnenie vypýtať si soľ a následne nastaviť pre klienta aplikačný kontext. Spôsob nastavenia kontextu ukazuje nasledujúci príklad. V prípade nesprávneho používateľského mena a hesla, príkaz SELECT nenájde v tabuľke záznam a vyhodí výnimku NO_DATA_FOUND, v tomto prípade sme ju odchytili a vyhodili sme vlastnú výnimku.

```
PROCEDURE init_client_context(p_login    IN client.login%TYPE,
                              p_password IN client.password%TYPE) IS
```

60 Je to v súlade s dodržiavaním princípu minimálnej množiny oprávnení.
61 Údaje musia byť posielané cez zabezpečený komunikačný kanál. Na zabezpečenie kanála je vhodné napríklad SSL.

```
  l_client_id   client.id%TYPE;
BEGIN
  -- overenie mena a hesla
  SELECT c.id
  INTO   l_client_id
  FROM   client c
  WHERE  c.login = p_login
  AND    c.password = p_password;

  -- nastavenie kontextu
  sys.dbms_session.set_context(namespace => 'bank_context',
                               attribute => 'account_no',
                               VALUE     => NULL);

  sys.dbms_session.set_context(namespace => 'bank_context',
                               attribute => 'user_type',
                               VALUE     => 'CLIENT');

  sys.dbms_session.set_context(namespace => 'bank_context',
                               attribute => 'client_id',
                               VALUE     => l_client_id);

EXCEPTION
  -- v pripade ze SELECT nenasiel data vratime vynimku
  WHEN no_data_found THEN
    raise_application_error(-20667, 'Authentication error');
END init_client_context;
```

Príklad 3.2.1.1: Príklad procedúry z balíka zviazaného s aplikačným kontextom implementujúcej inicializáciu kontextu pre zákazníka.

Príklad 3.2.1.1 je súčasťou nášho návrhu FGAC, konkrétne balíka `BANK_CONTEX_PKG`, ktorý je možné nájsť v prílohe 2. V tomto balíku sa nachádza aj nastavovanie kontextu zamestnanca banky, správcu konfigurácie a pre terminál.

V nasledujúcom texte ukážeme niekoľko autorizovaných pohľadov a popíšeme možné problémy, na ktoré sme pri implementácii narazili. Najprv sme vytvorili aplikačný kontext `BANK_CONTEX`, ktorý sme zviazali s balíkom `BANK_CONTEX_PKG`.

V našej implementácii je možné nájsť tieto typy autorizovaných pohľadov:

1. Pohľady pre viacero používateľov a operácií.
2. Pohľady pre jednu operáciu a viacero používateľov.
3. Pohľady pre jedného používateľa a viacero operácií.
4. Špecifické pohľady pre jedného používateľa a jednu operáciu.

```
CREATE OR REPLACE VIEW v_all_client AS
SELECT id, first_name, last_name, login, phone_no, address,
client_type, valid
FROM   client
WHERE  id = sys_context('bank_context', 'client_id')
WITH CHECK OPTION;

-- pridelime prislusne opravnenia zakaznikovi
GRANT SELECT ON v_all_client TO client ;
GRANT UPDATE(phone_no) TO client ;

--pridelime prislusne opravnenia na pohlad pracovnikovi
GRANT UPDATE(phone_no, address, valid) on v_all_client TO teller ;
GRANT SELECT ON v_all_client TO teller ;

--pridelime pracovnikovi pravo generovat id zo sekvencie
GRANT SELECT ON seq_client_id TO teller;
```
Príklad 3.2.1.2: Príklad pohľadu pre viacero používateľov a operácií.

Príkladom na prvý typ pohľadu je pohľad `V_ALL_CLIENT` uvedený v príklade 3.2.1.2. Tento pohľad umožňuje zákazníkovi vidieť svoje informácie[62] a zmeniť svoje telefónne číslo. Pracovník môže taktiež vidieť informácie o používateľskom účte zákazníka[63], navyše môže zmeniť telefónne číslo zákazníka a jeho adresu. Taktiež môže zneplatniť používateľský účet nastavením validity. Tu sme sa však stretli s prvým problémom. Pomocou tohto pohľadu nemôže pracovník vložiť nový používateľský účet, pretože pracovník nemôže mať nastavený identifikátor používateľa vo svojom kontexte, keďže záznam s týmto identifikátorom ešte neexistuje. Ďalším problémom, prečo to nie je možné je, že v pohľade nechceme projektovať stĺpce `PASSWORD` a `SALT`[64].

Musíme teda vytvoriť pre zamestnanca špecifický pohľad[65], ktorý mu umožní vloženie nového používateľského účtu zákazníka. Jeden zo spôsobov, ktorým je to možné docieliť, je v príklade 3.2.1.3. Premenná `EMPLOYEE_TYPE` sa inicializuje pri inicializácii aplikačného kontextu. Funkcia `DECODE`[39] v tomto prípade zmení reťazec „CTELLER“ na reťazec „COMMON“ a reťazec „BTELLER“ na reťazec „B2B“. V podmienke sa overí, či zamestnanec vytvoril používateľa, ktorého mal právo vytvoriť. Nevýhodou tohto prístupu, okrem toho, že sme nútení vytvárať ďalší pohľad, je nemožnosť použitia

62 Zákazník môže nastaviť ako `CLIENT_ID` iba svoj identifikátor (viď príklad 3.2.1.1).
63 To, či si môže pracovník nastaviť určitý identifikátor klienta, sa rozhodne pri nastavovaní identifikátora v aplikačnom kontexte, kde sa overí o aký typ zákazníka ide (biznis alebo obyčajný).
64 Oracle nepodporuje prideľovanie oprávnenia na príkaz `SELECT` na jednotlivé stĺpce.
65 Tento pohľad je teda štvrtého typu.

klauzuly RETURNING[66], pokiaľ nemáme aj právo na SELECT na tento pohľad.

```
CREATE OR REPLACE VIEW v_teller_i_client AS
SELECT id, first_name, last_name, login, password, salt, phone_no,
address, client_type
FROM   client
WHERE  client_type = decode(sys_context('bank_context',
                                        'employee_type'),
                            'CTELLER',
                            'COMMON',
                            'BTELLER',
                            'B2B')
WITH CHECK OPTION
GRANT INSERT ON v_teller_i_client TO teller;
```

Príklad 3.2.1.3: Príklad umožňujúci vytvorenie používateľského účtu zákazníka. Špecifický pohľad pre jednu operáciu a jedného používateľa.

Ak by nám nevadila projekcia stĺpcov PASSWORD a SALT, bolo by možné aj iné riešenie. Doplnili by sme do aplikačného kontextu premennú USER_TYPE, ktorá sa nastavuje pri inicializácii aplikačného kontextu. Do WHERE klauzuly by sme potom doplnili podmienku, ktorá by sa vyhodnotila ako pravdivá v prípade, že by išlo o zákazníka[67], alebo ak by išlo o zamestnanca, ktorý môže pristupovať k údajom zákazníka[68]. Výhoda tohto prístupu je v tom, že máme jeden pohľad. Nevýhodou je trochu zložitejšia WHERE klauzula. Príklad a popis takéhoto prístupu je možné nájsť v prílohe 5, kde sme tento prístup aplikovali na operáciu vloženia transakcie, tento príklad je zároveň aj príkladom druhého typu pohľadu.

Ďalšou možnosťou je implementácia funkcie MASK, ktorá namiesto danej hodnoty vráti NULL. Túto funkciu by sme použili v pohľade na projektované stĺpce PASSWORD a SALT. Problémom je, že tieto dva stĺpce už nie sú modifikovateľné. Tento problém sa dá vyriešiť definíciou spúšťača typu INSTEAD OF pre operácie UPDATE a INSERT (viď príklad 3.4.1.3). Stačil by nám teda aj jeden pohľad. Cenou je však nižšia výkonnosť.

Posledný typ pohľadov je reprezentovaný príkladom 3.2.1.4. Ide o pohľad pre zákazníka. Tento pohľad mu umožňuje vložiť, odstrániť, zmeniť jedine svoju preddefinovanú platbu.

66 Táto klauzula umožňuje po vložení záznamu pomocou príkazu INSERT uložiť do premennej alebo viacerých premenných hodnotu jedného alebo viacerých stĺpcov z práve vloženého záznamu. Toto je výhodné, pokiaľ sa do vkladaného záznamu automaticky generuje nejaký údaj (napríklad identifikátor), ktorého hodnotu chceme zistiť.
67 Aby sme nezobrali zákazníkovi možnosť prístupu k údajom pomocou tohto pohľadu.
68 Podobná podmienka ako v príklade 3.2.1.3.

```
CREATE OR REPLACE VIEW v_client_predefined_payment AS
SELECT id, client_id, account_to, var_symb, note, payment_name
FROM   predefined_payment
WHERE  client_id = sys_context('bank_context', 'client_id')
WITH CHECK OPTION

GRANT SELECT ON v_client_predefined_payment TO client
GRANT DELETE ON v_client_predefined_payment TO client
GRANT INSERT ON v_client_predefined_payment TO client
GRANT UPDATE ON v_client_predefined_payment TO client
```

Príklad 3.2.1.4: Príklad autorizovaného pohľadu pre zákazníka umožňujúci manipuláciu s jeho preddefinovanými platbami.

V prílohe 2 uvádzame kompletné zdrojové kódy vytvoreného dátového modelu a k nemu vytvoreného FGAC pomocou autorizovaných pohľadov. V tejto prílohe je aj niekoľko príkladov, ktoré demonštrujú spôsob využitia autorizovaných pohľadov. V prílohe 6 sme sa rozhodli uviesť jednu ukážku využitia autorizovaných pohľadov, ktorá demonštruje, akým spôsobom by mohla aplikačná vrstva využívať tieto autorizované pohľady.

Cieľom vytvoreného FGAC príkladu je demonštrovať možnosti autorizovaných pohľadov, a nie úplne pokryť funkcionalitu modelovej aplikácie.

Pri implementácii FGAC sme sa snažili o ideálny stav – mať k jednej tabuľke iba jeden autorizovaný pohľad, pomocou ktorého by bolo možné kontrolovať všetky operácie. Jedným z najväčších problémov, ktorému sme pri napĺňaní tohto cieľa museli čeliť, je fakt, že DBMS Oracle neumožňuje definíciu názvov stĺpcov pri prideľovaní oprávnenia typu `SELECT`. Toto je jedna z najčastejších príčin vzniku nových autorizovaných pohľadov. Táto situácia vzniká napríklad pri vkladaní nových záznamov pomocou operácie `INSERT`. Ak má používateľ povolené vložiť záznam, ktorý nemá právo vidieť, je nutné vytvoriť nový pohľad. Analogicky pri operácii `UPDATE`, pokiaľ má mať používateľ právo napríklad zmeniť stav záznamu, pričom záznam s novým stavom by nemal byť preňho viditeľný, je opäť potrebné vytvoriť nový pohľad. Autorizovaný pohľad teda môže byť použitý pre všetky operácie, iba ak tieto operácie sú aplikovateľné na rovnaký predikát definovaný vo `WHERE` klauzule. Podmienka vo `WHERE` klauzule je tak často zložitejšia, pokiaľ pohľad využívame pre viacero operácií, a teda ďalšou nevýhodou je väčšia náročnosť na vyhodnotenie tejto podmienky, a tým aj možný negatívny dopad na výkonnosť dopytu.

Druhou stránkou tohto problému je, že autorizovaný pohľad musí obsahovať podmienku,

ktorá sa správne vyhodnotí pre všetkých používateľov. Tento problém je možné vyriešiť definíciou dodatočného parametra v aplikačnom kontexte – typu používateľa. Typ používateľa by sa potom nastavil pri inicializácii aplikačného kontextu. V tomto prípade sa v dopyte vyhodnotia najprv všeobecné predikáty, platné pre všetkých používateľov, a následne sa vykoná vyhodnotenie predikátu pre používateľa, ktorého typ je nastavený v aplikačnom kontexte. Samozrejme musíme zvážiť, či dovolíme používateľovi zmeniť svoj typ. Nevýhodou tohto prístupu je, že podmienka vo `WHERE` klauzule môže byť zložitá, najmä ak máme veľký počet typov používateľov. V prílohe 5 je uvedený príklad takéhoto prístupu, kde vidieť, že podmienka môže byť už relatívne komplikovaná aj pri malom počte typov používateľov.

Alternatívnym prístupom je mať viacej jednoduchších autorizovaných pohľadov. V tomto prípade nevýhoda spočíva v tom, že tvorca aplikácie musí pracovať s veľkým počtom pohľadov, pričom výsledný kód môže byť mätúci pre programátora – napríklad pri odstraňovaní a modifikácii záznamov z tej istej tabuľky sa môžu použiť dva rôzne pohľady.

3.2.2 Implementácia pomocou VPD

Ďalšou metódou implementácie FGAC je prepisovanie dopytov, tento spôsob je implementovaný v Oracle pomocou VPD. Prepisovanie dopytov pomocou VPD umožňuje dynamicky, v čase vykonávania, priradiť predikát dopytom vykonávaným nad databázovou tabuľkou alebo pohľadom. Môžeme teda vyhodnotiť, aký používateľ, za akých okolností spúšťa dopyt, a na základe toho pridať predikát k vykonávanému dopytu. VPD podobne ako autorizované pohľady využíva aplikačný kontext. Okrem aplikačného kontextu však využíva aj ďalšiu konštrukciu:

Bezpečnostná politika (*Security policy*) je funkcia, ktorá vracia reťazec reprezentujúci predikát. Tento predikát sa používa na filtrovanie dát pri vykonávaní dopytu. Bezpečnostná politika (ďalej len politika) je naviazaná na databázovú tabuľku alebo pohľad. Pri nadväzovaní politiky na tabuľku alebo pohľad sa určí množina príkazov, na ktoré sa táto bezpečnostná politika aplikuje. Politika môže byť aplikovaná na príkazy `SELECT`, `INSERT`, `UPADTE`, `DELETE` a `INDEX`[69]. Predikát, ktorý vracia, sa aplikuje na určené príkazy, ktoré k tejto tabuľke či pohľadu pristupujú. V rámci tejto funkcie

69 `INDEX` ovplyvňuje príkazy `CREATE INDEX` a `ALTER INDEX`.

môžeme využiť aplikačný kontext, buď v algoritme rozhodujúcom o návratovom predikáte, alebo v samotnom návratovom predikáte. Na jednu tabuľku alebo pohľad je možné aplikovať aj viacero politík. V tomto prípade sa aplikujú na príkaz pristupujúci k tabuľke či pohľadu všetky tieto politiky, pričom predikáty, ktoré vracajú, sa spoja logickou spojkou AND. Teda ak máme napríklad politiku, ktorá vracia predikát: SYS_CONTEXT('ctx', 'user_type')= 'EMP' a máme ďalšiu politiku, ktorá vracia predikát AMOUNT < 1000, pričom tieto dve politiky sú aplikované na ten istý databázový objekt a príkaz, potom výsledný predikát bude vyzerať nasledovne:

SYS_CONTEXT('ctx', 'user_type')='EMP' AND AMOUNT < 1000.

Rozhranie pre pridávanie a odoberanie politík k tabuľkám a pohľadom poskytuje balík DBMS_RLS. V prípade, že k objektu chránenému politikou pristupuje používateľ SYS, politika sa neaplikuje.

Pokiaľ chceme, aby používateľ nevidel žiadne záznamy, môžeme aplikovať politiku, ktorá vráti predikát, ktorý sa vždy vyhodnotí ako FALSE, napríklad “1=0”. Týmto spôsobom je možné upraviť tabuľku aj tak, aby bola iba na čítanie – pre príkazy INSERT, UPDATE a DELETE vráti politika predikát identicky rovný FALSE . Pokiaľ však vrátime prázdny reťazec alebo NULL[70] ,docielime presne opačný efekt. Príkaz sa vykoná bez aplikácie nejakého predikátu, môžu byť teda ovplyvnené všetky záznamy v tabuľke alebo pohľade.

Balík DBMS_RLS aplikovanie politiky na pohľad alebo tabuľku procedúry ADD_POLICY. Popis parametrov tejto funkcie je možné nájsť v prílohe 8. Viac informácií je možné nájsť v napríklad v [1], [39] a [36].

Existuje aj možnosť kešovania predikátov pomocou vhodného nastavenia typu politiky. Typy politík určuje parameter POLICY_TYPE. V našom príklade implementácie FGAC pomocou VPD budeme používať nasledovné typy politík:

STATIC – Funkcia implementujúca politiku je vykonaná iba raz a výsledný reťazec obsahujúci predikát je uložený v zdieľanej pamäti. Tento predikát sa potom použije pri každom príkaze k danému objektu.

SHARED_STATIC – Toto nastavenie má rovnaký význam ako STATIC, umožňuje však

70 V Oracle je prázdny reťazec vyhodnotený ako NULL – čo podľa SQL 2003 nie je správne.

navyše zdieľať predikát medzi objektami používajúcimi rovnakú politiku. Ak pri nastavení STATIC dva objekty používajú rovnakú politiku a použijeme príkaz na obidva tieto objekty, pričom sa na tento príkaz vzťahuje daná politika, funkcia implementujúca politiku sa zavolá dvakrát[71]. Pri použití SHARED_STATIC by sa zavolala funkcia iba raz.

DYNAMIC – Prednastavená politika. Funkcia implementujúca politiku sa zavolá vždy počas vykonávania príkazu.

Funkcia implementujúca politiku musí mať nasledujúci interfejs:

```
FUNCTION policy_function (object_schema IN VARCHAR2,
                          object_name      VARCHAR2) RETURN VARCHAR2
```

kde OBJECT_SCHEMA je meno schémy, ktorá vlastní objekt, na ktorý sa aplikuje politika a OBJECT_NAME je meno tohto objektu.

Z hľadiska výkonnosti by sme sa mali snažiť o minimalizáciu počtu volaní bezpečnostných politík, a teda čo najviac využívať statické politiky.

Na demonštráciu implementácie FGAC použijeme dátový model definovaný v kapitole 3.2.1[72]. Využijeme aj rovnakú architektúru aplikácie, spôsob autentifikácie a nastavovanie aplikačného kontextu, pričom uvedieme len implementáciu FGAC pomocou VPD iba pre jednu tabuľku. Implementáciu pre ostatné tabuľky je možné nájsť v prílohe 2. Vybrali sme si tabuľku CLIENT, pretože na nej môžeme najlepšie demonštrovať, problémy ktoré sme museli pri implementácii riešiť.

Problém inicializácie aplikačného kontextu. Ak chceme inicializovať aplikačný kontext na základe údajov z tabuľky, nad ktorou je bezpečnostná politika, vzniká "chicken and egg" problém. Naša tabuľka CLIENT obsahuje prihlasovacie meno (LOGIN) heslo (PASSWORD). Tieto údaje sa využívajú pri nastavovaní aplikačného kontextu pre zákazníka (viď príklad 3.2.1.1). Pokiaľ by nad touto tabuľkou bola politika založená obmedzení záznamov na základe parametra CLIENT_ID nastaveného v aplikačnom kontexte (príklad 3.2.2.1), zákazník nemá akým spôsobom nastaviť tento parameter. Pri pokuse overiť prihlasovacie meno a heslo, by sa do dopytu doplnil predikát, ktorý vracia politika uvedená v príklade 3.2.2.1, ktorá by spôsobila, že by sa obmedzila

71 Samozrejme iba pri prvom použití, keď sa reťazec obsahujúci predikát ešte nenachádza v pamäti.
72 Jedinou zmenou je premenovanie stĺpca ID na CLIENT_ID v tabuľke CLIENT a pridanie stĺpca CLIENT_ID do tabuľky CARD.

množina záznamov na prázdnu množinu, pretože parameter CLIENT_ID nie je v kontexte nastavený.

```
CREATE OR REPLACE FUNCTION client_id_policy(p_schema IN VARCHAR2,
                                            p_object IN VARCHAR2)
RETURN VARCHAR2 IS
BEGIN
  RETURN 'client_id = sys_context(''bank_context'', ''client_id'')';
END client_id_policy;
```

Príklad 3.2.2.1: Politika obmedzujúca množinu záznamov

Riešením je použitie pohľadu na túto tabuľku a následná aplikácia politiky na tento pohľad, a nie na pôvodnú tabuľku. Pôvodná tabuľka sa použije iba na získanie údajov pre inicializáciu aplikačného kontextu. Je možné použiť namiesto pohľadu aj inú tabuľku. Pri tomto prístupe však musíme zabezpečiť synchronizáciu údajov. Pohľad teda považujeme za lepšie riešenie, pretože jednak nie je nutná synchronizácia, a jednak nezaberáme zbytočne diskový priestor.

```
CREATE OR REPLACE VIEW v_client AS
SELECT client_id, first_name, last_name, login, password, salt,
       phone_no, address, client_type, valid
FROM client
```

Príklad 3.2.2.2: Pohľad na tabuľku CLIENT.

Na pohľad musíme aplikovať iba politiky vzťahujúce sa na príkaz SELECT, ostatné politiky môžeme aplikovať na pôvodnú tabuľku. Pohľad potom „zdedí“ politiky aplikované na tabuľku.

```
dbms_rls.add_policy(object_schema => 'bank_is',
                    object_name => 'v_client', -- pohľad
                    policy_name => 'client_s_policy',
                    function_schema => 'bank_is',
                    policy_function => 'client_id_policy',
                    statement_types => 'select',
                    policy_type => dbms_rls.shared_static)
```

Príklad 3.2.2.3: Pridelenie politiky na pohľad V_CLIENT pre príkaz SELECT. Pri dopyte na tento pohľad sa aplikuje aj predikát, ktorý vracia funkcia CLIENT_ID_POLICY.

Problém obmedzení oprávnení na stĺpce. Pridelili sme síce politiku pre pohľad V_CLIENT, ale tento pohľad umožní zobrazovanie hesla a soli, čo nie je žiadúce. Ako je možné vidieť z prílohy 1, Oracle neumožňuje definovať stĺpce, na ktoré sa má vzťahovať oprávnenie SELECT. Identifikovali sme dva spôsoby riešenia tohto problému. Jedným

spôsobom je vyradiť tieto dva stĺpce z množiny projektovaných stĺpcov, čím ale stratíme možnosť modifikácie a vkladania hodnôt pre tieto stĺpce. Budeme teda musieť používať okrem pohľadu aj pôvodnú tabuľku.

Druhým riešením je pridelenie politiky pre príkaz `SELECT` vzťahujúcej sa na konkrétne stĺpce. Nech politika `NOT_ALLOWED_POLICY` vracia predikát, ktorá sa vyhodnotí vždy ako `FALSE`, napríklad „1=0“, potom príklad 3.2.2.4, ukazuje akým spôsobom je možné zabrániť zobrazeniu hesla a soli. Politika `NOT_ALLOWED_POLICY` sa aplikuje na dopyt na pohľad `V_CLIENT`, ak sa v tomto pohľade vyskytne použitie stĺpca `PASSWORD` alebo `SALT`.

```
dbms_rls.add_policy(object_schema => 'bank_is',
                    object_name => 'v_client',
                    policy_name => 'client_s_col_policy',
                    function_schema => 'bank_is',
                    policy_function => 'not_allowed_policy',
                    statement_types => 'select',
                    policy_type => dbms_rls.shared_static,
                    sec_relevant_cols => 'password,salt',
                    sec_relevant_cols_opt => dbms_rls.all_rows);
```

Príklad 3.2.2.4: Politika aplikovaná na príkaz `SELECT` v prípade, že sa v dopyte vyskytne stĺpec `PASSWORD` alebo `SALT`.

Ďalšou požiadavkou je, aby zákazník mohol meniť svoje telefónne číslo a zamestnanec mohol meniť telefónne číslo, adresu a validitu. Na splnenie tejto požiadavky musíme najskôr aplikovať politiku `CLIENT_ID_POLICY` aj na príkaz `UPDATE`, analogicky ako v príklade 3.2.2.3.

Ďalej potrebujeme obmedziť stĺpce, na ktoré môže byť `UPDATE` aplikovaný. Na tento účel je najvhodnejší príkaz `GRANT`. Modifikovateľnosť stĺpcov je síce možné obmedziť aj analogicky ako v príklade 3.2.2.4 tým, že definujeme zoznam „zakázaných“ stĺpcov. Problémom však je, že tieto zakázané stĺpce nemôžeme použiť ani vo `WHERE` klauzule príkazu `UPDATE`. Ak je napríklad kľúč medzi zakázanými stĺpcami nemusíme vedieť modifikovať ľubovoľný záznam.

Pre príkaz `INSERT` je možné obmedzenie stĺpcov, buď pomocou príkazu `GRANT`, alebo aplikovaním analogickej politiky, ako v príklade 3.2.2.4.

Problém cyklickosti.

S týmto problémom sme sa stretli, keď sme chceli implementovať politiku analogickú k autorizovanému pohľadu v prílohe 5. Problém spočíva v tom, že ak sa snažíme v politike, ktorá sa má aplikovať na tabuľku TRANSACTION, dopytovať tabuľku TRANSACTION, nastane chyba. Identifikovaným riešením je opäť vytvorenie pohľadu, na ktorý aplikujeme politiku, ktorá už umožní dopytovať tabuľku TRANSACTION. Politiku pre pohľad V_TRANSACTION pre príkaz INSERT sme uviedli v prílohe 9. Zároveň ide o príklad dynamickej politiky, pretože táto politika sa musí vyhodnotiť vždy pri spúšťaní dopytu, keďže jej návratová hodnota závisí od toho, kto dopyt spúšťa. Ďalším možným riešením by bolo zvážiť, či chceme overovať limity pomocou politiky alebo toto overovanie budeme vykonávať v aplikačnej logike. V prípade, že by nám stačilo overovanie limitov v aplikačnej logike, nemuseli by sme v politike dopytovať tabuľku TRANSACTION, a teda ani vytvárať pohľad pre túto tabuľku, pretože politika by už nebola cyklická.

Okrem týchto problémov, s ktorými sme sa stretli my, sa čitateľ pri implementácii FGAC pomocou VPD môže stretnúť ešte s týmito problémami:

Problém inferencie. Referenčná integrita obchádza VPD. Môžeme dokázať existenciu alebo neprítomnosť dát v rodičovskej[73] tabuľke tak, že sa budeme pokúšať vkladať hodnoty do stĺpca tabuľky, ktorá je synom z hľadiska cudzieho kľúča. Ak sa nám podarí vložiť záznam do tejto tabuľky, tak potom v rodičovskej tabuľke exituje záznam, ktorý má v príslušnom stĺpci, ktorý je súčasťou cudzieho kľúča, túto hodnotu. Ak sa táto operácia nepodarí, takýto záznam neexistuje. Toto je možné, aj keď nemáme oprávnenie SELECT na rodičovskú tabuľku.

Môžeme odstrániť dáta z tabuľky, ktorá je z hľadiska cudzieho kľúča synom, pokiaľ je v tejto tabuľke definovaná akcia CASCADE, ktorá sa vykoná pri odstránení záznamu z rodičovskej tabuľky (ON DELETE CASCADE). Toto je možné, aj keď nemáme oprávnenie odstrániť záznam z tabuľky, ktorá je synom.

Analogicky ako pri odstránení dát, môžeme modifikovať tabuľku, ktorá je synom z hľadiska cudzieho kľúča. Konkrétne môžeme modifikovať stĺpec, ktorý je súčasťou cudzieho kľúča tak, že ho nastavíme na NULL. Toto je možné v prípade, ak je v tabuľke, ktorá je synom, definovaný cudzí kľúč s klauzulou ON UPDATE SET NULL. Opäť, táto

73 Z hľadiska cudzieho kľúča.

akcia je možná, aj keď nemáme oprávnenie `UPDATE` na príslušný riadok tabuľky, respektíve na celú tabuľku.

Problém s agregačnými funkciami - Ak má mať používateľ oprávnenie vykonávať agregačné funkcie nad celou tabuľkou, pričom existuje politika, ktorá limituje jeho prístup k tejto tabuľke, výsledok agregačnej funkcie sa bude vzťahovať iba na tie dáta, ktoré mu umožní vidieť politika. V tomto prípade musíme vytvoriť pohľad, na ktorý aplikujeme všetky politiky, ktoré sú aplikované na tabuľku, nad ktorou sa majú vykonávať agregačné dopyty. Ďalej musíme vytvoriť špecifické pohľady pre každý agregačný dopyt. Používateľovi potom pridelíme oprávnenie na pohľady implementujúce agregačné dopyty a oprávnenie na pohľad, na ktorý sme aplikovali pôvodné politiky. Oprávnenia na tabuľku, z ktorej tieto pohľady čerpajú dáta, používateľovi odoberieme.

3.2.3 Porovnanie autorizovaných pohľadov a VPD

Pri použití VPD sa narozdiel od autorizovaných pohľadov môžu aplikovať politiky priamo na tabuľky, pričom sa tieto politiky budú aplikovať aj na DBA, pokiaľ tento DBA nemá oprávnenie `EXEMPT ACCESS POLICY`. Z tohto dôvodu môžu nastať problémy pri zálohovaní dát – nezálohujú sa všetky dáta, ale iba dáta, ktoré neboli odstránené z výsledku dopytu z dôsledku aplikácie politiky. Použitie oprávnenia a jeho prideľovanie však odporúčame sledovať, pretože ide o veľmi silné oprávnenie. Pri autorizovaných pohľadoch môže DBA pri zálohovaní dopytovať pôvodné relácie.

Ak sa pripájajú pomocou jedného databázového používateľa všetci aplikační používatelia, musíme pri obidvoch týchto riešeniach explicitne zavolať procedúru autorizovanú nastaviť aplikačný kontext, keďže nemáme dostatočné informácie o tom, ktorý z aplikačných používateľov sa pripojil. Pokiaľ nemáme veľký počet aplikačných používateľov, môžeme vytvoriť databázového používateľa pre každého aplikačného používateľa, a následne môžeme definovať spúšťač pre operáciu prihlásenia sa do sdatabázy, ktorý nastaví príslušnému používateľovi jemu zodpovedajúci aplikačný kontext.

Jednou z najväčších výhod využívania VPD oproti autorizovaným pohľadom je granularita. Pokiaľ chceme mať k jednej tabuľke iba jeden autorizovaný pohľad, musíme vytvoriť pohľad, na ktorý je možné aplikovať všetky požadované príkazy, pričom ako sme videli, táto úloha nie je niekedy možná. Naproti tomu VPD umožňuje zadefinovať

rôzne politiky pre jednotlivé príkazy vzťahujúce sa na tabuľku. Teda vystačíme si väčšinou s jednou tabuľkou. Problém nastáva, pokiaľ chceme inicializovať aplikačný kontext na základe tabuľky, ku ktorej obmedzujeme prístup pomocou tohto kontextu – „chicken and egg“ problém, prípadne ak je politika cyklická. Vtedy musíme vytvoriť ďalší databázový objekt. Pohľad musíme vytvoriť, aj keď má mať používateľ možnosť spúšťať dopyty využívajúce agregačné funkcie nad všetkými záznamami tabuľky, pričom politiky mu obmedzujú množinu týchto záznamov.

Kladnou stránkou obidvoch týchto prístupov je, že sa snažia o riadenie prístupu na úrovni databázy. Všetky aplikácie môžu teda „dediť“ bezpečnostné politiky implementované v databázovej vrstve. V prípade, že sa bezpečnostná politika zmení, nie je potrebné meniť všetky aplikácie, ale stačí zmeniť politiku pre VPD alebo modifikovať autorizovaný pohľad. Problém môže nastať pri autorizovaných pohľadoch, ak si zmena politiky vyžiada vznik nového pohľadu. V tomto prípade je potom potrebné zmeniť aj aplikácie využívajúce tento pohľad.

Dôležitou úlohou pri návrhu FGAC pre dátový model je aj stanovenie hranice medzi riadením prístupu a aplikačnou logikou. Musíme si uvedomiť, že FGAC k danému dátovému modelu môže využívať aj viacero aplikácií, prípadne postupom času môžu pribudnúť aj ďalšie aplikácie využívajúce dátový model, pre ktorý FGAC implementujeme.

Pokiaľ voláme autorizované pohľady z PL/SQL objektov a zmeníme tento pohľad, je potrebné prekompilovať tieto databázové objekty, ktoré sú závislé na tomto pohľade. Pri použití VPD toto nie je potrebné.

Nevýhodou VPD oproti autorizovaným pohľadom je komplikovanejšie hľadanie chyby, jednak preto, že kód aplikujúci na politiky sa v prípade VPD vykonáva na pozadí, a jednak preto, že predikáty sú reprezentované reťazcami. Ak spravíme nejakú syntaktickú chybu, je ťažšie ju odhaliť pri VPD ako pri použití autorizovaných pohľadov.

Ďalšou výhodou implementácie FGAC je neexistencia „zadných dvierok“. Pokiaľ máme riadenie prístupu implementované v aplikačnej vrstve, používateľ by sa mohol pripojiť do DBMS pomocou inej aplikácie alebo nástroja, napríklad pomocou SQL*PLUS, JDBC ovládača a podobne, čím by mohol získať prístup k údajom, ku ktorým by prístup mať nemal. VPD umožňuje definovať politiky aj pre samotného vlastníka objektu. Pri použití akéhokoľvek nástroja či aplikácie môže používateľ operovať iba nad tými istými údajmi.

Pri využití autorizovaných pohľadov odporúčame po implementácii uzamknúť schému vlastníka tabuliek a pohľadov, čím vynútime prístup k tabuľkám iba prostredníctvom autorizovaných pohľadov, prípadne oprávnení. Rovnaký postup doporučujeme aj pri implementácii FGAC pomocou VPD v prípade, že sme museli riešiť „chicken and egg", problém s cyklickými politikami alebo problém s agregačnými funkciami. Teda problémy, ktoré si vynucujú vznik pohľadu.

Spôsob implementácie FGAC pomocou autorizovaných pohľadov, ktorý sme navrhli, je aplikovateľný aj na iné DBMS, ktoré implementujú modifikovateľné pohľady a umožňujú prideliť oprávnenia na stĺpce pohľadov. Tieto požiadavky by mali spĺňať všetky DBMS, ktoré sa snažia dodržiavať štandard SQL 2003. Prehľad DBMS, ktoré tieto požiadavky spĺňajú, je možné nájsť v prílohe 1. Ďalšou požiadavkou umožňujúcou obmedzenie počtu pohľadov, je možnosť použitia nejakého typu globálnej premennej – napríklad aplikačného kontextu alebo globálnej premennej v balíku.

3.3 Šifrovanie a hešovanie

Ako sme už spomínali, pred použitím šifrovania je potrebné zvážiť, či na splnenie našich požiadaviek nestačia iné mechanizmy, ktoré DBMS poskytuje (autorizácia, identifikácia a autentifikácia). Primárnym použitím šifrovania dát v DBMS by mala byť ochrana databázových súborov a záloh, kde sú informácie dostupné ako otvorený text, prípadne šifrovanie môže byť poslednou bezpečnostnou vrstvou v prípade prelomenia autentifikačných a autorizačných mechanizmov. Keďže rýchlosť šifrovania je v DBMS dôležitou požiadavkou, najčastejšie sa využíva symetrické šifrovanie.

Šifrovanie a hešovanie je v Oracle možné pomocou balíkov `DBMS_CRYPTO` a `DBMS_OBFUSCATION_TOOLKIT`, ktorý však obsahuje iba podmnožinu funkcionality obsiahnutej v balíku `DBMS_CRYPTO` a existuje len z dôvodu spätnej kompatibility, preto sa ním ďalej nebudeme zaoberať.

3.3.1 Balík DBMS_CRYPTO

Šifrovanie pomocou funkcionality DBMS je možné prostredníctvom balíka `DBMS_CRYPTO`. Tabuľka 3.3.1.1 poskytuje prehľad možností šifrovania a hešovania balíka `DBMS_CRYPTO`.

Šifry (Dĺžka kľúča)	DES (56), 3DES (168), AES (128, 192, 256), RC4, 3DES_2KEY (112)
Vypchávka	PKCS5, nuly
Módy činnosti pre blokové šifry	CBC, CFB, ECB, OFB
Hešovacie funkcie	MD4, MD5, SHA-1
Hešovacie funkcie s kľúčom	HMAC_MD5, HMAC_SH1
Podporované typy	`CLOB`, `BLOB`, `RAW`

Tabuľka 3.3.1.1: Prehľad možností šifrovania a hešovania balíka `DBMS_CRYPTO`.

Balík `DBMS_CRYPTO` umožňuje aj generovanie náhodných kľúčov pomocou funkcie `RANDOMBYTES`[74]. Existuje aj balík `DBMS_RANDOM`, ktorý však nie je vhodný na kryptografické použitie[75].

Balík `DBMS_CRYPTO` síce umožňuje šifrovanie, ale žiadnym spôsobom nerieši manažment kľúčov, a pokiaľ chceme použiť tento balík, manažment kľúčov bude v našej réžii.

Pokiaľ by sme chceli manažovať kľúče v DBMS, máme niekoľko možností, akým spôsobom to môžeme docieliť, ktoré sme uviedli v kapitole 2.4. Žiadna z týchto možností však nedokáže ochrániť kľúč pred DBA.

V [39] je popísaný ešte jeden spôsob manažmentu kľúčov. Kľúče nebudú nikde uložené, ale budú sa vypočítavať pre každý záznam pomocou nejakého algoritmu. Výpočet budú vykonávať procedúry, ktoré šifrujú a dešifrujú dáta a zdrojový kód týchto procedúr bude obfuskovaný. Oracle umožňuje obfuskáciu[76] zdrojového kódu naprílad pomocou balíka `DBMS_DDL`[77]. (viď príklad 3.3.1.1).

```
execute SYS.DBMS_DDL.CREATE_WRAPPED(function_returning_PLSQL_code());
```

Príklad 3.3.1.1: Jeden z možných spôsobov umožňujúcich vytvorenie obfuskovaného objektu.

Výhodou takéhoto prístupu je, že odpadá problém s miestom uloženia kľúčov, a každý záznam je šifrovaný iným kľúčom. Nevýhodou je, že ak útočník zistí algoritmus, ktorým sa vypočítava kľúč, bude vedieť tieto dáta dešifrovať. Ďalším problémom tohto prístupu

74 Funkcia volá generátor náhodných čísel, ktorý bol certifikovaný organizáciou RSA a je vhodný na kryptografické použitie.
75 Tento balík síce spĺňa odporúčania uvedené v [64], tie však patria už medzi zastaralé[65].
76 Obfuskácia je jeden zo spôsob ochrany proti dekompilácii, ktorý sa snaží skrývať význam a spôsob implementácie kódu. Viac informácií o obfuskácii je možné získať napríklad v [66].
77 Existujú aj iné spôsoby, napríklad nástroj umožňujúci obfuskáciu z príkazového riadku – wrap.

je, že obfuskácia implementovaná v Oracle je prelomiteľná, a existuje aj niekoľko nástrojov, ktoré umožňujú získať z obfuskovaného kódu zdrojový kód[78]. Tieto nástroje však nie sú bežne dostupné[79].

Príkladom algoritmu na výpočet kľúča pre záznam môže byť tento postup:

1. Nad primárnym kľúčom záznamu vykonáme operáciu XOR, pričom druhým operandom bude nejaká konštanta, napríklad reťazec „diplomovapraca".
2. Na hodnotu vypočítanú v kroku 1 aplikujeme hešovaciu funkciu SHA-1.
3. Hodnotu vypočítanú v kroku 2 použijeme ako kľúč na šifrovanie alebo dešifrovanie požadovaného stĺpca.

Prístup k týmto procedúram je potrebné chrániť. Ani tento prístup neochráni údaje pred DBA, keďže DBA môže spustiť každú funkciu. Nebezpečenstvo tohto prístupu spočíva aj v tom, že z informačnej schémy je možné zistiť, na akých objektoch je balík závislý (viď príklad 3.3.2.3), a aplikovať postup uvedený v nasledujúcej kapitole na tieto objekty a z toho odvodiť algoritmus pre výpočet kľúča. Problém závislostí je možné vyriešiť tak, že použijeme príkaz EXECUTE IMMEDIATE, ktorý umožňuje vykonať kód, ktorý je predaný tomuto príkazu ako reťazec.

3.3.2 Problémy súvisiace s použitím balíka DBMS_CRYPTO

Zistili sme, že nezávisle od miesta manažmentu kľúčov (DBMS, OS, klient), nie je možné pri použití tohto balíka ochrániť údaje pred DBA, pretože kľúč a otvorený text je predávaný funkciám a procedúram balíka DBMS_CRYPTO prostredníctvom parametra, pričom samotný balík neimplementuje šifrovacie a hešovacie algoritmy, ale len volá ďalší balík DBMS_CRYPTO_FFI, ktorému opäť predáva šifrový kľúč a otvorený text ako parametre. Spôsob šifrovania pomocou balíka DBMS_CRYPTO je v znázornený v nasledujúcom príklade.

```
DECLARE
   l_msg       VARCHAR2(200) :=  'Tajna sprava'; -- otvoreny text
   l_enc_msg   RAW(2000);       -- sifrovana sprava
   l_key       RAW(32);         -- 256-bit sifrovy kluc
   l_enc_type  PLS_INTEGER :=   -- Typ sifroveho algoritmu
```

78 Túto informáciu sme získali z [37] a [66].

79 Nepodarilo sa nám ani intenzívnym hľadaním na internete objaviť ich implementáciu, našli sme iba zmienky o ich existencii v [37] a [66].

```
                              DBMS_CRYPTO.ENCRYPT_AES256 -- AES 256 bitov
                            + DBMS_CRYPTO.CHAIN_CBC      -- mod cinnosti:
CBC
                            + DBMS_CRYPTO.PAD_PKCS5;     -- typ vypchavky
BEGIN
   -- vygenerujeme 256 bitovy kluc
   l_key := DBMS_CRYPTO.RANDOMBYTES(256/8);
   -- Prekonvertujeme otvoreny text na typ RAW a znakovu sadu zmenime na
UTF8
   -- a nasledne tento text zasifrujeme vygenerovanym klucom
   encrypted_raw := DBMS_CRYPTO.ENCRYPT(
       src => UTL_I18N.STRING_TO_RAW(l_msg,  'AL32UTF8'),
       typ => l_enc_type,
       key => l_key);
END;
```
Príklad 3.3.2.1: Ukážka šifrovania pomocou balíka DBMS_CRYPTO

Balík DBMS_CRYPTO sa nachádza v schéme SYS. Pre tento balík existuje verejné synonymum (PUBLIC SYNONYM) DBMS_CRYPTO, ktoré umožňuje použitie balíka bez prefixu schémy SYS. Keď voláme balík DBMS_CRYPTO bez prefixu DBMS postupuje nasledovne:

1. Zistí, či existuje objekt DBMS_CRYPTO v aktuálnej schéme (CURRENT_SCHEMA). Ak existuje, použije sa tento objekt, inak sa pokračuje krokom 2.
2. Zistí, či existuje synonymum DBMS_CRYPTO v aktuálnej schéme. Ak existuje, použije objekt, pre ktorý bolo toto synonymum vytvorené, inak sa pokračuje krokom 3.
3. Zistí, či existuje verejné synonymum DBMS_CRYPTO. Ak existuje, použije sa objekt, pre ktorý bolo toto verejné synonymum vytvorené, inak sa vyhlási chyba indikujúca neexistenciu objektu.

Tým, že sme v príklade nepoužili prefix SYS, ale sme využili verejné synonymum, sme dali potenciálnemu útočníkovi, ktorý chce získať šifrový kľúč, možnosť vytvoriť proxy balík, ktorý obsahuje rovnaké deklarácie konštánt, funkcií a procedúr ako balík DBMS_CRYPTO. V tele tohto balíka zadefinuje však funkcie a procedúry tak, že sa vykoná najprv škodlivý kód, a následne sa zavolá pôvodná procedúra, funkcia z balíku DBMS_CRYPTO.

```
...
FUNCTION  Encrypt  (src IN RAW,
                    typ IN PLS_INTEGER,
                    key IN RAW,
                    iv  IN RAW DEFAULT NULL) RETURN RAW IS
BEGIN
  -- vykoname skodlivy kod
  log(src, typ, key, iv);
  -- vykoname povodnu funkciu
  RETURN SYS.DBMS_CRYPTO.Encrypt(src, typ, key, iv);
END;
...
```

Príklad 3.3.2.2: Ukážka definície funkcie v proxy balíku.

Útočník má viacero možností, ktorými môže takýto proxy balík podvrhnúť:

1. Vytvorí proxy balík s názvom `DBMS_CRYPTO` v schéme, v ktorej tento balík voláme. Útočník musí mať oprávnenie pre vytvorenie balíka v tejto schéme. Teda používateľ, ktorému schéma patrí, musí mať oprávnenie `CREATE PROCEDURE` a útočník musí byť schopný autentifikovať sa v DBMS pod týmto používateľom alebo útočník musí mať oprávnenie `CREATE ANY PROCEDURE` umožňujúce vytvoriť procedúru, funkciu alebo balík v ľubovoľnej schéme. Toto oprávnenie by mal mať iba DBA.
2. Vytvorí synonymum pre proxy balík s názvom `DBMS_CRYPTO` v schéme, v ktorej voláme balík `DBMS_CRYPTO`, a pridelí oprávnenie na spustenie proxy balíka tejto schéme. Postup je analogický ako v predchádzajúcom príklade, príslušné oprávnenia sú `CREATE SYNONYM` a `CREATE ANY SYNONYM`. Útočník musí mať navyše možnosť vytvoriť proxy balík v nejakej schéme, ktorá má oprávnenie `EXECUTE` na balík `DBMS_CRYPTO`. Tento spôsob útoku je nenápadnejší ako prvý spôsob.
3. Zmení verejné synonymum `DBMS_CRYPTO` tak, aby ukazovalo na proxy balík. Útočník musí mať oprávnenie `CREATE PUBLIC SYNONYM` a musí byť schopný vytvoriť proxy balík v nejakej schéme, ktorá má oprávnenie `EXECUTE` na balík `DBMS_CRYPTO`. Následne pridelí útočník právo na `EXECUTE` pre tento proxy skupine `PUBLIC`. Narozdiel od predchádzajúcich dvoch typov útokov, týmto spôsobom môže útočník získať kľúče, ktoré sa používajú pri šifrovaní pomocou balíka `DBMS_CRYPTO`, od všetkých používateľov.

Pozorný čitateľ si mohol všimnúť, že tieto tri spôsoby nemusia vyžadovať oprávnenia DBA, pričom však s oprávneniami DBA je jednoduchšie vykonať ktorýkoľvek z týchto spôsobov útoku. Možným spôsobom riešenia je použiť prefix SYS, čo môžeme vynútiť[80] odstránením verejného synonyma. Používaním plného mena zaručíme, že útočník musí mať oprávnenia DBA, aby mohol získať šifrový kľúč. DBA môže okrem spomínaných troch spôsobov premiestniť balík DBMS_CRYPTO zo schémy SYS do inej schémy (napríklad SYSTEM), a v schéme SYS vytvoriť proxy balík s názvom DBMS_CRYPTO, ktorý bude volať pôvodný balík. Balík DBMS_CRYPTO volá pomocný balík DBMS_CRYPTO_FFI, ktorému predáva šifrový kľúč, a ktorý volá knižnicu implementujúcu šifrové algoritmy. DBA môže vytvoriť aj teda aj proxy balík pre DBMS_CRYPTO_FFI.

Ako je možné vidieť z tabuľky 3.3.1.1, balík DBMS_CRYPTO nepodporuje šifrovanie typu VARCHAR2, ktorý patrí medzi najpoužívanejšie. Preto sa premenné typu VARCHAR2 musia pred šifrovaním konvertovať na typ RAW. Konverzie medzi týmito typmi umožňujú balíky UTL_I18N a UTL_RAW. Na tieto balíky je možné použiť práve opísané spôsoby útoku, takže aj pre ne by sme mali používať prefix schémy SYS a nie verejné synonymá, ak ich využívame na konverziu dôverných informácií. Tieto balíky sú narozdiel od balíka DBMS_CRYPTO štandardne pridelené skupine PUBLIC, a teda sú dostupné pre každého databázového používateľa. Vytvorenie proxy balíka je preto jednoduchšie, keďže ho môžeme vytvoriť v ľubovoľnej schéme.

Drvivá väčšina príkladov dostupných na internete, vrátane príkladu v dokumentácii k DBMS Oracle [39], prefix SYS nepoužíva. Ani pre balík DBMS_CRYPTO, ani pre balíky UTL_I18N, respektíve UTL_RAW.

Na základe týchto zistení nemôžeme odporučiť používanie balíka DBMS_CRYPTO, pokiaľ je cieľom ochrániť údaje aj pred útočníkom, ktorý má alebo vie získať oprávnenia DBA. V prípade, že sa však čitateľ rozhodne tento balík použiť, odporúčame uvádzať prefix SYS pre balíky, ktorým sú predávané kľúče a dôverné informácie ako parametre, čím obmedzíme množinu potenciálnych útočníkov iba na DBA. Odporúčame odstrániť verejné synonymum pre balík DBMS_CRYPTO. Pred odstránením tohto synonyma však musíme z informačnej schémy zistiť, ktoré objekty v používateľských schémach

80 Ak nemá používateľ oprávnenie opäť vytvoriť synonymum (CREATE [PUBLIC] SYNONYM)

využívajú toto synonymu. Ukážka, akým spôsobom je možné zistiť závislosti, je v príklade 3.3.2.3. Ak tento prefix nepoužívajú, je ich potrebné pred odstránením synonyma najprv upraviť. Predchádzajúci postup je možné aplikovať aj na balíky `UTL_RAW` a `UTL_I18N`. Problémom je, že tieto verejné synonymá sú používané aj objektami, ktoré sú vytvorené samotným DBMS, pričom sú tieto objekty obfuskované, teda nie je možné ich zmeniť[81]. Riešenie spočíva v tom, že v týchto schémach vytvoríme synonymum odkazujúce na príslušný balík v schéme `SYS`. Po tomto kroku môžeme odstrániť verejné synonymum. Ďalším krokom by malo byť sledovanie integrity objektov v schéme `SYS`, spôsob riešenia tohto problému sme uviedli v kapitole 2.5. Sledovanie integrity v schéme `SYS` nám umožní odhaliť modifikáciu balíka. Dôležité je aj sledovanie operácií, ktoré vytvárajú nové objekty, čím môžeme odhaliť útočníka, ktorý môže vytvoriť proxy balík alebo verejné synonymu.

```
SELECT o.owner parent_schema,
       o.object_name parent,
       o.object_type parent_type,
       o2.owner child_schema,
       o2.object_name child,
       o2.object_type child_type
FROM sys.dependency$ d
     INNER JOIN sys.dba_objects o ON (d.p_obj# = o.object_id)
     INNER JOIN sys.dba_objects o2 ON (d.d_obj# = o2.object_id)
WHERE o.object_name=UPPER('&1') AND
      o.owner IN ('PUBLIC')
```

Príklad 3.3.2.3: Príkaz, ktorý z informačnej schémy zistí objekty závislé na danom verejnom synonyme. Skript je dostupný v prílohe 2.

V príklade 3.3.2.2 sme uviedli ukážku proxy funkcie obsahujúcej škodlivý kód, reprezentovaný funkciou `LOG`. Táto funkcia nemusí byť implementovaná iba pomocou operácie `INSERT`. Po inštalácii DBMS má skupina `PUBLIC` právo `EXECUTE` na balíky `UTL_HTTP` a `UTL_SMTP` umožňujúce komunikáciu pomocou protokolu HTTP, resp. SMTP[82]. Mali by sme teda zvážiť aj odobranie týchto oprávnení, čím obmedzíme počet spôsobov implementácie funkcie `LOG`.

81 Navyše, zasahovanie do interných objektov DBMS môže byť dôvodom pre stratu podpory zo strany Oracle.
82 SMTP je protokol používaný pri posielaní elektronickej pošty.

3.4 Transparentné šifrovanie dát

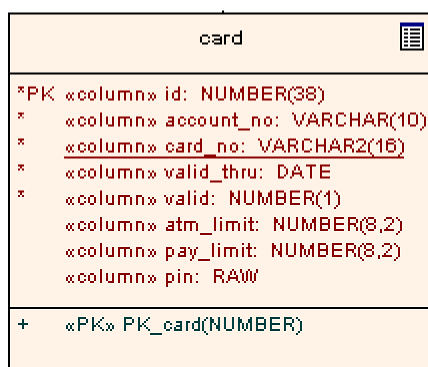
Transparentné šifrovanie dát je možné v DBMS Oracle implementovať dvoma spôsobmi:

- Pomocou pohľadov.
- Deklaratívnym spôsobom pomocou príkazov a klauzúl príkazov poskytovaných samotným DBMS.

V nasledujúcich podkapitolách si predstavíme tieto dva spôsoby.

3.4.1 TDE pomocou pohľadov

V tejto kapitole si ukážeme, akým spôsobom je možné implementovať TDE pomocou pohľadov, spúšťačov a balíka `DBMS_CRYPT`. Uvažujme tabuľku `CARD` z modelu kapitoly 3.2.1. Pre túto tabuľku implementujeme transparentné šifrovanie stĺpca `PIN`.



Obrázok 3.4.1.1:Tabuľka `CARD`

Najprv si zadefinujeme funkcie, pomocou ktorých budeme stĺpec `PIN` šifrovať a dešifrovať. V týchto funkciách budeme používať na šifrovanie a dešifrovanie iba jeden konštantný kľúč[83]. V príklade 3.4.1.1je implementácia funkcie pre šifrovanie, analogicky je implementovaná aj funkcia pre dešifrovanie, ktorú čitateľ môže nájsť v prílohe 2.

```
CREATE OR REPLACE FUNCTION encrypt(p_value IN VARCHAR2) RETURN RAW IS
  l_enc_type NUMBER := sys.dbms_crypto.ENCRYPT_AES128 +
                       sys.dbms_crypto.CHAIN_CBC +
                       sys.dbms_crypto.PAD_PKCS5;
BEGIN
  RETURN
   sys.dbms_crypto.encrypt(sys.utl_i18n.string_to_raw(p_value,
```

83 Keďže v tomto prípade je naším cieľom demonštrovať spôsob implementácie TDE, nie riešiť manažment kľúčov.

```
                                                    'AL32UTF8'),
                           l_enc_type,
                           hextoraw('F0802924CF517BE7F02B16018CA7C944'));
END;
```

Príklad 3.4.1.1: Funkcia pre šifrovanie. Zašifruje danú hodnotu pomocou konštantného kľúča a následne ju vráti.

Následne vytvoríme pohľad, pomocou ktorého budeme dáta transparentne šifrovať a dešifrovať. Tento pohľad je uvedený v príklade 3.4.1.2. V definícii tohto pohľadu je zaujímavá funkcia `DECRYPT`, ktorá dešifruje stĺpec `PIN`.

```
CREATE OR REPLACE VIEW V_CARD AS
SELECT id,
       account_no,
       card_no,
       valid_thru,
       valid,
       atm_limit,
       pay_limit,
       decrypt(c.pin) AS pin
FROM   card
```

Príklad 3.4.1.2: Implementácia pohľadu preTDE.

Pomocou tohto pohľadu je zatiaľ možné dáta iba dešifrovať, pretože tento pohľad je síce modifikovateľný, ale iba čiastočne – nie je možné modifikovať stĺpec `PIN`. Keby sme sa o to pokúsili pomocou príkazu `INSERT` alebo `UPDATE`, tieto príkazy by sa skončili chybovým hlásením[84]. Musíme teda nejakým spôsobom zabezpečiť vloženie šifrovanej hodnoty do tabuľky prostredníctvom tohto pohľadu. Spôsob, ktorým je toto možné docieliť, je definícia spúšťača typu `INSTEAD OF` pre operácie `UPDATE` a `INSERT`, ktorý sa vykoná namiesto týchto operácií v prípade, že chceme modifikovať záznam, respektíve vkladať nový záznam.

```
CREATE OR REPLACE TRIGGER v_card_trg
  INSTEAD OF INSERT OR UPDATE ON v_card
  FOR EACH ROW
DECLARE
BEGIN
  IF (updating) THEN
    UPDATE card
    SET    valid     = :NEW.valid,
           atm_limit = :NEW.atm_limit,
           pay_limit = :NEW.pay_limit,
```

84 ORA-01733[39]

```
          pin       = encrypt(:NEW.pin)
    WHERE  id = :NEW.id;
  ELSIF (inserting) THEN
    INSERT INTO card
      (id, account_no, card_no, valid_thru, valid, atm_limit,
       pay_limit, pin)
    VALUES
      (:NEW.id,
       :NEW.account_no,
       :NEW.card_no,
       :NEW.valid_thru,
       :NEW.valid,
       :NEW.atm_limit,
       :NEW.pay_limit,
       encrypt(:NEW.pin));
  END IF;
END v_card_trg;
```

Príklad 3.4.1.3: Spúšťač typu `INSTEAD OF` pre TDE.

V príklade 3.4.1.3 sme definovali spúšťač, ktorý sa vykoná v prípade, že použijeme príkaz `UPDATE` alebo `INSERT` na pohľad `V_CARD`. Tento spúšťač má dve logické časti. Pokiaľ bol vyvolaný príkazom `UPDATE`, vykoná sa prvá časť, v ktorej je definovaný príkaz `UPDATE`. Pokiaľ bol spúšťač vyvolaný príkazom `INSERT`, vykoná sa iba časť, kde je definovaný príkaz `INSERT`. V týchto príkazoch sa namiesto vloženia otvoreného textu do tabuľky vykoná najprv funkcia `ENCRYPT`, ktorá tento otvorený text zašifruje, a až následne sa šifrovaný text vloží do tabuľky. V spúšťači sme umožnili modifikáciu iba pre niektoré stĺpce tým, že sme v príkaze `UPDATE` v klauzule `SET` neuviedli všetky možné stĺpce.

Použitie pohľadu pre TDE znázorňuje príklad 3.4.1.4. Ako si môžeme na tomto príklade všimnúť, použitie pohľadu sa nijakým spôsobom neodlišuje od použitia obyčajnej databázovej tabuľky.

```
INSERT INTO v_card
  (id, account_no, card_no, valid_thru, valid, pin)
VALUES
  (1, '123456789', '1111222233334444', SYSDATE, 1, '12345678')

UPDATE v_card
SET    pin = pin || '0' -- pridáme k pinu retazec '0'
```

Príklad 3.4.1.4: Vloženie záznamu do pohľadu a modifikácia záznamu v pohľade, ktorý implementuje TDE.

Kompletný príklad aj s demonštráciou vkladania a modifikácie je možné nájsť v prílohe 2. Tento príklad používa balík `DBMS_CRYPTO`. Ako sme uviedli v predchádzajúcej kapitole, nie je možné ochrániť takto šifrované dáta pred DBA pri akomkoľvek manažmente kľúčov. Jednak DBA môže získať kľúč, ktorým sú dáta šifrované, a jednak môže prezerať všetky tabuľky a pohľady. Teda aj pre DBA sa údaje transparentne šifrujú a dešifrujú.

3.4.2 TDE deklaratívnym spôsobom

Implementácia TDE je možná aj deklaratívnym spôsobom, pomocou príkazov, ktoré poskytuje DBMS Oracle[85]. Pri deklaratívnom spôsobe nám DBMS umožňuje iba automatický manažment kľúčov.

Kľúče sú uložené v systémovej tabuľke `ENC$`, zašifrované pomocou hlavného kľúča (*master key*), ktorý je uložený mimo databázy – v kľúčenke. Ak chceme získať dáta z tabuľky, ktorá používa TDE, DBMS automaticky vyberie príslušný šifrovaný kľúč z tabuľky `ENC$`, vyberie[86] hlavný kľúč z kľúčenky, ktorým dešifruje šifrovaný kľúč, a týmto dešifrovaným kľúčom dešifruje dáta v stĺpci, ktorý používa šifrovanie. Pre jednu tabuľku je možné použiť iba jeden kľúč, ktorý sa použije pre každý šifrovaný stĺpec.

Pokiaľ sa útočníkovi podarí získať databázové súbory, nebude môcť dešifrovať kľúče, ktorými sú šifrované údaje v tabuľkách, pretože nebude mať hlavný kľúč, a teda nebude vedieť ani dešifrovať údaje v databázových tabuľkách. Pokiaľ sa mu podarí získať aj kľúčenku, musel by získať heslo, ktorým je kľúčenka chránená

Ak chceme implementovať TDE deklaratívne, musí existovať kľúčenka, do ktorej musí mať DBMS prístup, aby z nej mohol načítať, prípadne do nej uložiť hlavný kľúč. Kľúčenka sa štandardne[87] nachádza v súbore $ORACLE_HOME/admin/$ORACLE_SID/wallet/ewallet.p12, kde $ORACLE_HOME je adresár, v ktorom je nainštalovaná databáza a $ORACLE_SID je meno databázy.

Následne musíme vygenerovať hlavný kľúč, ktorým budú zašifrované kľúče v tabuľke `ENC$`. Príkaz v príklade 3.4.2.1 vytvorí[88] a otvorí[89] kľúčenku, vygeneruje hlavný kľúč a

85 Iba od verzie 10g Release 2.
86 v tabuľke `ENC$` je identifikátor tohto kľúča, pomocou ktorého vie DBMS vybrať správny kľúč.
87 Umiestnenie je možné nastaviť v súbore sqlnet.ora.
88 Musí existovať adresár $ORACLE_HOME/admin/$ORACLE_SID/wallet. Kľúčenka sa vytvorí, iba ak neexistovala, pričom nastaví sa jej heslo z klauzuly `IDENTIFIED BY`.
89 Samozrejme iba v prípade, že kľúčenka ešte nebola otvorená.

uloží ho v kľúčenke. V klauzule IDENTIFIED BY musí administrátor uviesť správne heslo pre kľúčenku.

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY "CaseSensitive";
```

Príklad 3.4.2.1: Vygenerovanie hlavného kľúča v kľúčenke.

Pri šifrovaní a dešifrovaní pomocou hlavného kľúča môžeme využiť aj PKI. Môžeme vytvoriť certifikát pomocou nástroja orapki, ktorý je súčasťou DBMS. Ukážka príkazu, ktorý vytvorí certifikát[90] a uloží ho v kľúčenke, je v príklade 3.4.2.2.

```
orapki wallet add -wallet $ORACLE_HOME/admin/$ORACLE_SID/wallet -dn
"CN=my-user-cert, OU=FMFI, O=UK, L=BA, ST=BA, C=SK" -keysize 2048
-self_signed -validity 365 -user_cert
```

Príklad 3.4.2.2: Vytvorenie certifikátu a jeho pridanie do kľúčenky.

Aby sme mohli certifikát použiť na šifrovanie kľúčov v tabuľke ENC$, musíme zistiť identifikátor vytvoreného certifikátu. Identifikátory certifikátov sa nachádzajú v informačnej schéme v pohľade V$WALLET. Keď už poznáme identifikátor certifikátu, môžeme zmeniť hlavný kľúč. V príklade 3.4.2.3 je ukážka príkazu, pomocou ktorého je možné nastaviť certifikát ako hlavný kľúč.

```
ALTER SYSTEM SET ENCRYPTION KEY
"AkU0fjCr0I1WCEedwyWR8loAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" IDENTIFIED BY
"CaseSensitive";
```

Príklad 3.4.2.3: Nastavenie certifikátu ako hlavného kľúča.

To, že kľúč sa naozaj používa na šifrovanie jednotlivých kľúčov, môžeme overiť v informačnej schéme v tabuľke ENC$, kde je ku každému identifikátoru objektu uvedený identifikátor kľúča v kľúčenke.

Pri každom štarte databázy je kľúčenku potrebné otvoriť[91], inak nebude dostupný hlavný kľúč a nebude možné dešifrovať údaje. Pokiaľ sa pokúsime pristúpiť k šifrovanému stĺpcu, a nie je otvorená kľúčenka, príkaz skončí chybou. V príklade 3.4.2.4 uvádzame príkaz, ktorým je možné otvoriť kľúčenku. Analogicky je možné kľúčenku zavrieť[92]. a teda znepístupniť kľúče, ktoré sa využívajú na šifrovanie stĺpcov v jednotlivých tabuľkách.

90 Tento certifikát je podpísaný vlastným súkromným kľúčom (*self-signed*).

91 Pomocou nástroja orapki je možné nastaviť pre kľúčenku príznak auto_login, následne sa vytvorí obfuskovaná kópia kľúčenky, ktorá nevyžaduje heslo. Tento prístup však neodporúčame, pretože útočník, ktorý získa kľúčenku a dátové súbory, vie ľahko získať zašifrované údaje.

92 Namiesto klauzuly OPEN uvedieme klauzulu CLOSE.

```
ALTER SYSTEM SET ENCTYPTION WALLET OPEN IDENTIFIED BY "CaseSensitive";
```
Príklad 3.4.2.4: Otvorenie kľúčenky.

Určiť, ktoré stĺpce sa budú šifrovať, môžeme v príkazoch `CREATE TABLE` a `ALTER TABLE` pomocou klauzuly `ENCRYPT`. Je možné špecifikovať, aký algoritmus použijeme, pričom si môžeme vybrať buď 3DES (168 bitov), alebo AES (128, 192 a 256 bitov), a ďalej je možné špecifikovať príznak, či sa má šifrovať so soľou alebo bez. Ak použijeme soľ, nie je možné vytvoriť nad stĺpcom index.

Implementácia TDE deklaratívnym spôsobom pre tabuľku `CARD` z kapitoly 3.4.1 je uvedená v príklade 3.4.2.5. Tabuľku `CARD` modifikujeme tak, že najprv zmeníme typ stĺpca z `RAW(2000)` na `VARCHAR2(8)`, a následne na tento stĺpec aplikujeme klauzulu `ENCRYPT`.

```
ALTER TABLE card MODIFY (pin VARCHAR2(8) ENCRYPT USING 'AES128')
```
Príklad 3.4.2.5: Deklaratívna implementácia TDE.

Deklaratívna implementácia TDE má nasledujúce obmedzenia:

- Nie je možné šifrovať objekty, ktoré vlastní používateľ `SYS`.
- Nie je možné šifrovať stĺpec, ktorý je súčasťou cudzieho kľúča.
- Index[93] je možné použiť, iba ak nebola použitá soľ, a použije sa iba v prípade testu na rovnosť alebo nerovnosť.
- Podporované typy sú `CHAR`, `NCHAR`, `VARCHAR2`, `NVARCHAR2`, `NUMBER`, `DATE` a `RAW`. Teda nie je možné týmto spôsobom šifrovať stĺpce typu `BLOB` a `CLOB`.

3.4.3 Vyhodnotenie

Ako sme si mohli všimnúť, deklaratívna implementácia TDE je jednoduchšia. Jediným náročnejším úkonom je inicializácia kľúčenky pri prvom použití tohto spôsobu implementácie. Ďalšou výhodou oproti použitiu pohľadov je automatický manažment kľúčov, a tým aj jednoduchšia zmena kľúča – pomocou klauzuly `REKEY` v príkaze `ALTER TABLE`. Pri spôsobe implementácie TDE pomocou pohľadov by sme museli naprogramovať funkciu `DECRYPT_OLD`, ktorá by dešifrovala údaje pomocou starého kľúča, a funkciu `ENCRYPT_NEW`, ktorá by šifrovala údaje pomocou nového kľúča.

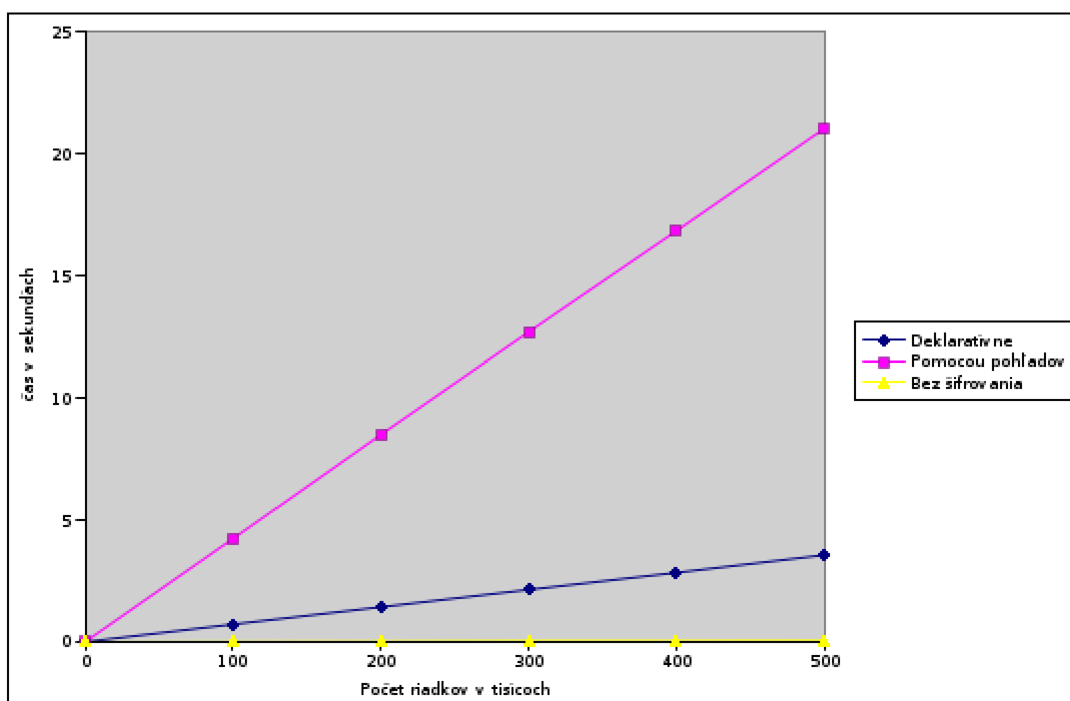
93 Je možné použiť iba B-Tree index. Nie je teda možné použiť bitmapové, funkčné a hešované indexy.

```
ALTER TABLE v_card REKEY USING 'AES256'

UPDATE v_card set
SET pin=ENCRYPT_NEW(DECRYPT_OLD(pin))
```

Príklad 3.4.3.1: Zmena kľúča v rôznych implementáciachTDE.

Porovnávali sme tieto dve implementácie aj z hľadiska výkonnosti. Vykonali sme dva testy. Prvým testom bolo meranie času pri selekcii z tabuľky obsahujúcej stotisíc až päťstotisíc záznamov, pričom jeden stĺpec v tejto tabuľke bol šifrovaný pomocou TDE. Vykonali sme tento test aj nad tabuľkou, ktorá nepoužívala TDE, aby sme ilustrovali, akou mierou TDE prispieva k zníženiu výkonnosti. Výsledky meraní sú znázornené na obrázku 3.4.3.1[94].



Obrázok 3.4.3.1: Porovnanie rýchlosti dopytov nad TDE.

V tomto teste bola efektívnejšia deklaratívna implementácia. Nepriaznivý výsledok implementácie používajúcej pohľady je spôsobený aj tým, že pre každý záznam sa musí zavolať PL/SQL funkcia, čo je pomerne náročná operácia. Keďže na šifrovanie používame balík `DBMS_CRYPTO`, pre každý záznam sa volá funkcia niekoľkokrát. Prvé volanie je volanie našej funkcie `DECRYPT`, táto funkcia volá funkciu z balíka `DBMS_CRYPTO`, funkcia z tohto balíka následne volá funkciu z balíka `DBMS_CRYPTO_FFI`, a až tá nakoniec zavolá knižnicu implementujúcu dešifrovací

94 Namerané hodnoty je možné nájsť v prílohe 7.

algoritmus.

V druhom teste sme porovnávali vplyv použitia typu šifry na výkonnosť. Tentokrát sme použili príkaz `INSERT`, pričom sme vkladali do tabuľky stotisíc záznamov. Opäť sme vykonali test aj nad tabuľkou, nad ktorou nebolo implementované TDE. Pri deklaratívnej implementácii sme dosiahli očakávané výsledky – doba potrebná na vykonanie príkazu sa predlžovala so zväčšujúcou sa dĺžkou kľúča, namerané údaje je možné nájsť v prílohe 7. Pri implementácii pomocou pohľadov má opäť nepriaznivý vplyv na výkonnosť najmä skutočnosť, že pre každý vložený riadok sa musí vykonať spúšťač, čo je PL/SQL procedúra.

Teda celkový výsledok nášho porovnania vyznieva priaznivejšie pre deklaratívnu implementáciu TDE. Obidve tieto implementácie sú určené hlavne pre ochranu dátových súborov. Ani jeden z nich nedokáže ochrániť údaje pred DBA. V prípade deklaratívnej implementácie DBA síce musí poznať heslo do kľúčenky, aby ju mohol otvoriť, a tým sprístupniť ostatné kľúče. Ak sú však tieto kľúče sprístupnené, každý kto je autorizovaný pristupovať k tabuľke obsahujúcej šifrované údaje, vie tieto údaje prečítať. Keďže DBA je autorizovaný pristupovať ku každej tabuľke, po otvorení kľúčenky bude môcť získať údaje z ľubovoľnej tabuľky používajúcej TDE. Pri implementácii pomocou pohľadov môže DBA použiť niektorú z techník útokov popísaných v kapitole 3.3.2, alebo priamo získať údaje z pohľadu.

4 Záver

Naším prvým cieľom bolo analyzovať problematiku zavedenia a udržiavania bezpečnosti v DBMS. Ucelený pohľad na túto problematiku sme poskytli v druhej kapitole, kde sme identifikovali možné problémy a načrtli sme možnosti riešenia týchto problémov. Vzhľadom na ďalšie ciele sme sa v rámci tejto kapitoly zamerali na problematiku šifrovania a riadenia prístupu, pričom v rámci riadenia prístupu sme skúmali existujúce modely FGAC. Naším prínosom je návrh implementácie autorizovaných pohľadov s využitím štandardu SQL 2003. Uvedený postup je možné aplikovať na každý DBMS, ktorý spĺňa vlastnosti umožňujúce FGAC špecifikované v SQL 2003. My sme tento postup aplikovali v DBMS Oracle v tretej kapitole.

V rámci druhej kapitoly sme skúmali aj to, aké bezpečnostné prvky poskytujú vybrané súčasné DBMS pre riešenie problému zavedenia a udržiavania bezpečnosti. Na základe tohto skúmania sme vytvorili porovnanie jednotlivých DBMS.

Ďalším cieľom bolo navrhnúť možnosti riešenia problému udržiavania a zavedenia bezpečnosti v DBMS Oracle. Ako sme uviedli v úvode tretej kapitoly, zistili sme, že riešenia tohto problému sú už navrhnuté vo viacerých zdrojoch, pričom málo pozornosti sa venuje problematike udržiavania bezpečnosti pomocou návrhu aplikácií s ohľadom na bezpečnosť. Preto sme sa rozhodli venovať sa tomuto problému, v rámci ktorého sme v súlade s našimi cieľmi analyzovali možnosti využitia FGAC a TDE. Navrhli sme spôsoby implementácie FGAC a TDE a implementovali sme príklady, ktoré tieto spôsoby demonštrujú. Analyzovali sme problémy, ktoré sa môžu vyskytnúť pri implementácii, vytvorili sme porovnania jednotlivých implementácií a sformulovali niekoľko odporúčaní a riešení pre identifikované problémy. Praktickú použiteľnosť navrhnutého riešenia sme demonštrovali na vytvorených výkonnostných testoch. V rámci TDE sme uviedli, akým spôsobom je možné využiť PKI. Možnosti využitia PKI v DBMS Oracle sme uviedli aj v porovnaní jednotlivých DBMS.

Myslíme si teda, že sme splnili všetky stanovené ciele a veríme, že táto práca môže inšpirovať vývojárov databázových aplikácií a prispeje k implementácii bezpečnejších aplikácií a k rýchlejšiemu riešeniu problémov, ktoré sme identifikovali.

Pri skúmaní spôsobov implementácie FGAC sme narazili na zaujímavý model FGAC – Netrumanovský model definovaný v kapitole 2.2, ktorého implementáciou sme sa v tejto

práci nezaoberali. Pokračovaním tejto práce by teda mohlo byť napríklad skúmanie tried dopytov, skúmanie rozhodnuteľnosti podmienečnej a nepodmienečnej správnosti týchto tried, a navrhnutie inferenčného systému, ktorý by odvodil ich správnosť. Ďalej by bolo možné implementovať tento inferenčný systém napríklad ako modul do nejakej open-source databázy napríklad PostgreSQL alebo MySQL.

5 Zoznam bibliografických odkazov

[1] Jeloka S. *Oracle Security Guide 10g Release 2.* Oracle. 2005.

[2] Bisták Ľ. *Technológie pre webové služby (diplomová práca).* Bratislava: *FMFI UK.* 2006.

[3] Stanek M. *Základy kryptológie (skriptá).* Bratislava: FMFI UK. 2004

[4] Wenbo M. *Modern Cryptography, Theory and Practice.* Prentice Hall. 2003. ISBN 978-0130669438.

[5] Menezes A., et al. *Handbook of Applied Cryptography.* CRC Press. 1996. ISBN 978-0849385230.

[6] Theriault M., Newman A. *Bezpečnost v Oracle.* Computer Press. 2004. ISBN 80-7226-979-8.

[7] *SQL Server 2005 – Security and* Protection [online]. Dostupné na internete: http://www.microsoft.com/technet/prodtechnol/sql/2005/library/security.mspx

[8] *Adaptive Server Enterprise 15.0* [online]. Dostupné na internete: http://sybooks.sybase.com/nav/detail.do?docset=1299

[9] *Domovská stránka spoločnosti Application Security, Inc.* [online]. Dostupné na internete: http://www.appsecinc.com

[10] *Stránka o bezpečnosti* [online]. Dostupné na internete: http://www.governmentsecurity.org

[11] Handley M., Rescorla E. *RFC4732: Internet Denial-of-Service Considerations.* [online], 2006. Dostupné na internete: http://www.ietf.org/rfc/rfc4732.txt

[12] *PKCS #5: Password-Based Cryptography Standard* [online]. 1999. Dostupné na internete: http://www.rsa.com/rsalabs/node.asp?id=2127

[13] Levický D. *Kryptografia v informačnej bezpečnosti.* Elfa. 2005. ISBN 80-8086-022-X.

[14] *NIST Special publications* [online]. Dostupné na internete: http://csrc.nist.gov/publications/nistpubs/index.html

[15] *Štandard ISO/IEC 17799:2005.* 2005.

[16] Domovská stránka organizácie HIPAA [online]. Dostupné na internete: http://www.hipaa.org

[17] Hazelwood V. *Defense-in-Depth* [online]. 2006. Dostupné na internete: http://security.sdsc.edu/DefenseInDepthWhitePaper.pdf

[18] *Virtual Private Network Consortium* [online]. Dostupné na internete: http://www.vpnc.org

[19] Srisuresh P., Egevang K. *RFC 3022: Traditional IP Network Address Translator* [online]. 2001. Dostupné na internete: http://www.ietf.org/rfc/rfc3022.txt

[20] Kent S., Seo K. *RFC 4301: Security Architecture for the Internet Protocol* [online]. 2005. Dostupné na internete: http://www.ietf.org/rfc/rfc4301.txt

[21] Noonan W. *Hardening Network Infrastructure.* McGraw-Hill Osborne Media, 2004, ISBN 978-0072255027.

[22] Cole E., et al. *Network Security Bible*. Wiley, 2005, ISBN 978-0764573972

[23] *Domovská stránka komunity The Internet Engineering Task Force* [online]. Dostupné na internete: http://www.ietf.org

[24] Griffiths P.P., Wade B. W. *An authorization mechanism for relational database system.* ACM Press. 1976. ISSN 0362-5915.

[25] Lunt T.F., Fernandez E.B. *Database security, SIGMOD Rec., vol 19.* ACM Press, 1990. ISSN 0163-5808.

[26] Fugini M. G., Castano S., Martella G. *Database Security*. Addison-Wesley Professional, 1994, ISBN 978-0201593754.

[27] *Proposed NIST Standard for Role-BasedAccess Control* [online]. 2004 Dostupné na internete: http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf

[28] Teťák Ľ. *Role Based Access Control System* (Diplomová práca). Bratislava: *FMFI UK,* 2006

[29] *Štandard ISO/IEC 9075-1:2003 (SQL/Framework).* 2003

[30] Štandard *ISO/IEC 9075-2:2003 (SQL/Foundation)*. 2003

[31] Harrison M.A., Ruzzo W.L., Ullman J.D. *Protection in operating systems, Commununications of the ACM vol. 19*. ACM Press, 1976. ISSN 0001-0782.

[32] Bertino E., Jajodia S., Samarati P. *A flexible authorization mechanism for relational data management systems*. ACM Press, 1999. ISSN 1046-8188.

[33] Jajodia S., Sandhu R. *Toward a multilevel secure relational data model*. ACM

Press, 1991. ISSN 0163-5808.

[34] Rizvi S., Mendelzon A., et al. *Extending Query Rewriting Techniques for FGAC. ACM Press.* 2004. ISBN 1-58113-859-8.

[35] Levy A. *Answering queries using views*, VLDB Journal. 2001

[36] *Oracle Technology Network* [online]. Dostupné na internete: http://otn.oracle.com

[37] *Domovská stránka spoločnosti Red-Database-Security* [online] Dostupné na internete: http://www.red-database-security.com

[38] *MySQL 5.1 Reference manual* [online]. Dostupné na internete: http://dev.mysql.com/doc/refman/5.1/en/index.html

[39] *Oracle Database 10g Release 2 (10.2) Documentation* [online]. Dostupné na internete: http://www.oracle.com/technology/documentation/database10gR2.html

[40] *Domovská stránka Very Large Data Base Endowment* [online] Dostupné na internete: http://www.vldb.org

[41] Štandard *ISO/IEC 9075-11:2003 (SQL/Schemata)*. 2003

[42] Sýkora M. *Presadzovanie bezpečnostnej politiky v databázach (Diplomová práca)*. FIIT STU. 2005.

[43] *Domovská stránka spoločnosti Tripwire* [online]. Dostupné na internete: http://www.tripwire.com

[44] *Kerberos: The Network Authentication Protocol* [online]. Dostupné na internete: http://web.mit.edu/Kerberos

[45] *Stránka o DCE* [online] Dostupné na internete: http://www.opengroup.org/dce

[46] *Stránka produktu DB Audit* [online]. Dostupné na internete: http://www.softtreetech.com/idbaudit.htm

[47] *Detecting SQL Injection in Oracle* [online]. Dostupné na internete: http://www.securityfocus.com/infocus/1714

[48] Halfond W.G.J., Orso A. *AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks*. ACM Press. 2005. ISBN 1-59593-993-4.

[49] Halfond W.G.J., Orso A. *Preventing SQL injection attacks using AMNESIA*. ACM Press. 2006. ISBN 1-59593-375-1.

[50] Halfond W.G.J., Orso A., Manolios P. *Using syntax-aware evaluation to counter*

SQL injection attacks. ACM Press. 2006. ISBN 1-59593-468-5

[51] *PostgreSQL 8.2.3 Documentation* [online]. Dostupné na internete: http://www.postgresql.org/docs/8.2/static/index.html

[52] *SQLite Documentation* [online]. Dostupné na internete: http://www.sqlite.org/docs.html

[53] Domovská stránka produktu SQLCrypt [online]. Dostupné na internete: http://www.sqlcrypt.com

[54] Domovská stránka produktu XP_Crypt [online]. Dostupné na internete: http://www.xpcrypt.com

[55] Wang X. *How to Break MD5 and Other Hash Functions*. 2005

[56] *RFC 2865: Remote Authentication Dial In User Service (RADIUS)* [online]. Dostupné na internete: http://www.ietf.org/rfc/rfc2865.txt

[57] Unified login with pluggable authentication modules[online]. Dostupné na internete: www.opengroup.org/tech/rfc/mirror-rfc/rfc86.0.ps

[58] *PKCS #12: Personal Information Exchange Syntax Standard* [online].Dostupné na internete: http://www.rsa.com/rsalabs/node.asp?id=2138

[59] Domovská stránka Pete Finnigan [online] Dostupné na internete: http://www.petefinnigan.com/orasec.htm

[60] *Oracle Security Technology Center* [online] Dostupné na internete: http://www.oracle.com/technology/deploy/security/index.html

[61] *Domovská stránka inštitútu SANS* [online]. Dostupné na internete: http://sans.org

[62] Kyte T. *Expert One-on-One Oracle*. Apress. 2005. ISBN 978-1590595251

[63] Finnigan P. *Oracle Security Step-by-Step*. SANS Press. 2004. ISBN 978-0974372747

[64] Eastlake D., Crocker S., Schiller J. *RFC 1750: Randomness Recommendations for Security* [online]. 1994. Dostupné na internete: http://www.ietf.org/rfc/rfc1750.txt

[65] Eastlake D., Crocker S., Schiller J. *RFC 4086: Randomness Requirements for Security* [online]. 2005. Dostupné na internete: http://www.ietf.org/rfc/rfc4086.txt

[66] Barak B., Goldreich O., et al. *On the (Im)possibility of Obfuscating Programs*. Springer-Verlag. 2001. ISBN 3-540-42456-3.

6 Prílohy

Príloha 1: Porovnanie bezpečnostných prvkov DBMS

	Sybase ASE 15	*SQLite 3.3.10*	*MySQL 5.1*
Hešovanie	Nie	Nie	MD5, SHA-1, password()
Symetrické šifrovanie	AES	Nie[95]	AES, DES
Asymetrické šifrovanie	Nie	Nie	Nie
Identifikácia	Používateľ	Nie	Používateľ, PKI
Autentifikácia	Heslo, Kerberos, NTLM, LDAP, PAM	Nie	Heslo, SSL
Riadenie prístupu	DAC, RBAC, FGAC	Nie	DAC, FGAC
Sledovanie	Áno	Nie	Nie
Manažment kľúčov	Áno	Nie	Nie
Užívateľské profily	Áno	Nie	Áno
Uzamykanie účtov	Áno	Nie	Áno
Deklaratívne TDE	Áno	Nie[96]	Nie
Nemenní systémoví používatelia	sa	Nie	Nie
Nástroje na zálohovanie	Áno	Nie	Áno
Oprávnenia na stĺpce	`SELECT`, `UPDATE`, `REFERENCES`, `DECRYPT`	Nie	`SELECT`, `UPDATE`, `INSERT`
Modifikovateľný pohľad	Áno	Nie	Áno

95 existujú komerčné rozšírenia (napríklad SQLcrypt).
96 existuje komerčné rozšírenie SQLcrypt umožnujúce transparentné šifrovanie.

	Oracle 10g Rel. 2	*SQL Server 2005*	*PostgreSQL 8.2*[97]
Hešovanie	MD5, SHA-1	MD2, MD4, MD5, SHA-1	MD5, SHA-1, SHA-2(244, 256, 384, 512) a iné[98]
Symetrické šifrovanie	DES, 3DES, AES, RC4, 3DES_2KEY	DES, DESX, 3DES, AES, RC2, RC4	Blowfish, AES, PGP, DES, 3DES, DESX, CAST5[99]
Asymetrické šifrovanie	Nie	RSA (512, 1024, 2048)	PGP
Identifikácia	Používateľ, PKI Biometria	Používateľ, PKI	Používateľ
Autentifikácia	Heslo, OS, vzdialený OS, Keberos, LDAP, SSL, RADIUS, DCE	Heslo, OS, Kerberos, NTLM, SSL	Trust, Heslo, OS, Kerberos, LDAP, PAM
Riadenie prístupu	DAC, RBAC, FGAC	DAC, RBAC, FGAC	DAC, RBAC, FGAC
Sledovanie	Áno	Áno	Nie
Manažment kľúčov	Áno	Áno	Nie
Užívateľské profily	Áno	Áno	Áno
Uzamykanie účtov	Áno	Áno	Áno
Deklaratívne TDE	Áno	Nie[100]	Nie
Nemenní systémoví používatelia	sys, system	sa	Nie
Nástroje na zálohovanie	Áno	Áno	Áno
Oprávnenia na stĺpce	`UPDATE`, `INSERT`, `REFERENCES`	`SELECT`, `UPDATE`, `REFERENCES`	Nie
Modifikovateľný pohľad	Áno	Áno	Nie[101]

97 Hešovacie a šifrovacie funkcie sú implementované v samostatnom prídavnom module pgcrypto.
98 Ak bol modul pgcrypto skompilovaný s OpenSSL (prepínač –with-openssl). V takom prípade sú dostupné aj všetky hešovacie funkcie podporované OpenSSL (neplatí to pre šifrovacie funkcie).
99 DES, 3DES, DESX a CAST5 sú dostupné, iba ak bol modul pgcrypto skompilovaný s OpenSSL.
100 Existujú komerčné rozšírenia umožňujúce TDE (napríklad DbEnrypt a XP_Crypt).
101 Ilúziu modifikovateľného pohľadu je možné vytvoriť pomocou definovania prepisovacích pravidiel (príkaz `RULE`) na daný pohľad pre požadované operácie (`INSERT`, `DELETE`, `UPDATE`).

Príloha 2: CD s ukážkovými príkladmi

```
.
|-- src
|   |-- Kapitola 2
|   |   `-- 2.2 Riadenie pristupu
|   |       |-- Priklad 2.2.1
|   |       `-- Priklad 2.2.2
|   `-- Kapitola 3
|       |-- 3.1 Aplikacny kontext
|       |-- 3.2 FGAC
|       |   |-- 3.2.1 Autorizovane pohlady
|       |   |   |-- bank_is
|       |   |   |-- client
|       |   |   |-- is_admin
|       |   |   |-- teller
|       |   |   `-- terminal
|       |   |-- 3.2.2 VPD
|       |   |   |-- bank_is
|       |   |   |-- client
|       |   |   |-- is_admin
|       |   |   |-- teller
|       |   |   `-- terminal
|       |   `-- Kontext vs balik
|       |-- 3.3 Sifrovanie
|       |   `-- 3.3.2 Problemy s DBMS_CRYPTO
|       |       `-- Priklad 3.3.2.3
|       `-- 3.4 TDE
|           |-- 3.4.1 TDE pomocou pohladov
|           |   `-- schema
|           `-- 3.4.2 TDE deklarativnym sposobom
|               `-- schema
`-- text
```

Adresárová štruktúra CD.

CD obsahuje dva hlavné adresáre src a text.

V adresári src je zdrojový kód implementovaných príkladov. Príklady sú rozdelené do podadresárov, ktorých označenie zodpovedá názvu kapitoly, v ktorej sa daný príklad vyskytol. V jednotlivých adresároch sa nachádzajú súbory README, ktoré popisujú obsah týchto adresárov.

V adresári text je možné nájsť elektronickú verziu tejto práce.

Príloha 3: Porovnanie výkonnosti aplikačného kontextu a funkcie balíka v dopyte

Počet riadkov	***Meranie***					***Medián***	***Priemer***
	č. 1	***č. 2***	***č. 3***	***č. 4***	***č. 5***		
100 000	40	40	50	40	60	40	
250 000	210	90	90	95	90	90	
500 000	180	180	230	180	180	180	
750 000	320	260	260	260	260	260	
1 000 000	400	360	340	340	350	350	

Tabuľka 1: Čas v milisekundách potrebný na vykonanie dopytu prechádzajúceho celú tabuľku (full scan) za použitia predikátu obsahujúceho funkciu `SYS_CONTEXT`.

Počet riadkov	***Meranie***					***Medián***	***Priemer***
	č. 1	***č. 2***	***č. 3***	***č. 4***	***č. 5***		
100 000	540	510	520	500	510	510	
250 000	1280	1260	1230	1250	1250	1250	
500 000	2470	2510	2510	2480	2500	2500	
750 000	3740	3740	3740	3770	3680	3740	
1 000 000	5040	5040	4990	5000	4970	5000	

Tabuľka 2: Čas v milisekundách potrebný na vykonanie dopytu prechádzajúceho celú tabuľku (full scan) za použitia predikátu obsahujúceho funkciu definovanú v balíku.

Testy boli vykonané na nasledujúcej konfigurácii:

Procesor: Intel Core Duo T2250 (1.73 Ghz)

Pamäť: 2x512 DDR2

Disk: SATA 150, 5.400 ot./min

Príloha 4: Popis dátového modelu vzorového príkladu

grid_value – tabuľka obsahujúca hodnoty kódov v grid karte	
id	Identifikátor záznamu.
grid_card_id	Identifikátor grid karty, ku ktorej sa kód vzťahuje.
ordern	Súradníce na karte. napr. riadok = order div 8, stĺpec = ordern mod 8.
value	Hodnota kódu.

grid_card – tabuľka grid kariet	
id	Identifikátor záznamu.
client_id	Identifikátor zákazníka, ktorému patrí grid karta.
valid	Príznak, či je grid karta validná.
new_card	Príznak, či ide o novú kartu.

account_type – tabuľka typov účtov	
id	Identifikátor záznamu.
interest	Úročenie účtu.
fee	Poplatok za účet.
type_name	Meno typu účtu.
valid	Príznak, či je typ validný.

transaction – tabuľka transakcií	
id	Identifikátor záznamu.
account_from	Účet, z ktorého peniaze odchádzajú v rámci transakcie.
account_to	Účet, na ktorý peniaze prichádzajú.
var_sym	Variabilný symbol platby.
grid_id	Identifikátor hodnoty na grid karte použitej pri platbe typu EBANK.
amount	Suma transakcie.
created	Čas vytvorenia.
status	Stav, v ktorom sa transakcia nachádza – NEW, SIGNED, PROCESSED, CANCELED.
trans_type	Typ transakcie – EBANK, TELLER_PAY, CARD_PAY, ATM_PAY.

client – Tabuľka obsahujúca údaje o zákazníkoch	
id	Identifikátor záznamu.
first_name	Meno zákazníka.
last_name	Priezvisko zákazníka.
login	Prihlasovacie meno.
password	Hešované heslo.
salt	Soľ.
phone_no	Telefónne čislo.
address	Adresa.
client_type	Typ zákazníka – COMMON, B2B.
valid	Príznak, či je účet platný.

account – tabuľka bankových účtov	
id	Identifikátor záznamu.
account_no	Číslo účtu.
account_type	Typ účtu.
client_id	Identifkátor zákazníka.
day_limit	Limit pre platbu typu EBANK.
valid	Príznak, či je účet platný.
balance	Peňažný stav na účte.

employee – tabuľka zamestnancov.	
id	Identifikátor záznamu.
login	Prihlasovacie meno zamestnanca.
first_name	Meno zamestnanca.
last_name	Priezvisko zamestnanca.
password	Hešované heslo.
salt	Soľ.
emp_type	Typ zamestnanca – ADMIN, BTELLER, CTELLER.

predefined_payment – tabuľka preddefinovných platieb.	
id	Identifikátor záznamu.
client_id	Identifikátor klienta, pre ktorého je preddefinovaná platba.
account_to	Číslo účtu, na ktorý sa platba bude vzťahovať.
var_symb	Variabliný symbol, ktorý sa použije pri platbe.
note	Poznámka.
payment_name	Názov platby.

card – tabuľka kariet.	
id	Identifikátor záznamu.
account_no	Číslo účtu, pre ktorý bola karta vydaná.
card_no	Číslo karty.
valid_thru	Platnosť karty.
valid	Platnosť záznamu.
atm_limit	Denný limit na výber z bankomatu – platba typu CARD_ATM.
pay_limit	Denný limit na platbu kartou – platba typu CARD_PAY.
pin	Pin.

Príloha 5: Príklad autorizovaného pohľadu

Nasledujúci autorizovaný pohľad umožňuje používateľom typu zákazník, pracovník a terminál vložiť záznam do tabuľky transakcií.

Musí ísť o nový záznam, jednotliví používatelia musia byť schopní nastaviť si číslo účtu, z ktorého sa má transakcia vykonať. Suma nových transakcií vrátane práve pridávanej transakcie musí byť menšia alebo rovná peňažnému stavu na účte.

Ďalej sa aplikujú podmienky pre jednotlivých používateľov. Pracovník môže vložiť transakciu typu „TELLER_PAY" v akejkoľvek výške.

Používateľ môže vložiť transakciu typu „EBANK_PAY", predstavujúcu platbu napríklad z internetového bankovníctva, ktorá nepresiahne povolený denný limit.

Terminál môže zadať transakciu typu „CARD_PAY", predstavujúcu platbu platobnou kartou, a „ATM_PAY", predstavujúcu výber z bankomatu. Na obidva tieto typy transakcií sa podobne ako na „EBANK_PAY" vzťahuje denný limit.

```
CREATE OR REPLACE VIEW v_all_i_transaction AS
SELECT id, account_from, account_to, var_sym, amount, created, status,
       trans_type
FROM   transaction
WHERE  -- podmienky, ktoré musia spĺňať všetci používatelia
       account_from = sys_context('bank_context', 'account_no')
AND    status = 'NEW' --musí ísť o novú transakciu

       -- peňažný stav na účte, od ktorého je odčítaná suma nových transakcií
       -- musí byť väčší ako transakcia, ktorú chceme vykonať
AND    amount <= (SELECT balance
                  FROM   account acc
                  WHERE  acc.account_no = sys_context('bank_context',
                                                      'account_no')) -
                 (SELECT nvl(sum(amount), 0)
                  FROM   transaction
                  WHERE  account_from = sys_context('bank_context',
                                                    'account_no')
                  AND    status = 'NEW')

AND   -- podmienky pre jednotlivých používateľov
      -- podmienka pre zamestnanca
       ((sys_context('bank_context', 'user_type') = 'EMP'
         AND trans_type = 'TELLER_PAY') OR

        -- podmienka pre zákaznika
        (sys_context('bank_context', 'user_type') = 'CLIENT' AND
         (trans_type = 'EBANK_PAY' AND
          amount <= (SELECT day_limit
                     FROM   account acc
```

```
                    WHERE  acc.account_no = sys_context('bank_context',
                                                        'account_no')) -
                   (SELECT nvl(sum(amount), 0)
                    FROM   transaction
                    WHERE  account_from = sys_context('bank_context',
                                                      'account_no')
                    AND    created > trunc(SYSDATE)
                    AND    status <> 'CANCELED'
                    AND    trans_type = 'EBANK_PAY'))) OR

      -- podmienka pre terminal
      (sys_context('bank_context', 'user_type') = 'TERMINAL' AND

      -- podmienka pre terminal pre výber z bankomatu
      ((trans_type = 'CARD_ATM' AND
      amount <= (SELECT atm_limit
                FROM   card
                WHERE  account_no = sys_context('bank_context',
                                                'account_no')
                AND    valid = 1) -
               (SELECT nvl(sum(amount), 0)
                FROM   transaction
                WHERE  account_from = sys_context('bank_context',
                                                  'account_no')
                AND    created > trunc(SYSDATE)
                AND    status <> 'CANCELED'
                AND    trans_type = 'CARD_ATM')) OR

-- podmienka pre terminál pre platbu kartou
      (trans_type = 'CARD_PAY' AND
      amount <= (SELECT pay_limit
                FROM   card
                WHERE  account_no = sys_context('bank_context', 'account_no')
                AND    valid = 1) -
               (SELECT nvl(sum(amount), 0)
                FROM   transaction
                WHERE  account_from = sys_context('bank_context','account_no')
                AND    created > trunc(SYSDATE)
                AND    status <> 'CANCELED'
                AND    trans_type = 'CARD_PAY')))))
WITH CHECK OPTION

GRANT INSERT ON v_all_i_transaction TO teller
GRANT INSERT ON v_all_i_transaction TO client
GRANT INSERT ON v_all_i_transaction TO terminal
```

Príloha 6: Príklad použitia autorizovaného pohľadu

Nasledujúci príklad ukazuje, akým spôsobom môže klientská aplikácia implementujúca funkcionalitu pre zamestnanca banky využívať autorizované pohľady. Definované v prílohe 2. Príklad ukazuje, ako prebieha autentifikácia, nastavenie aplikačného kontextu, a operácie nad autorizovanými pohľadmi. V nasledujúcom príklade predpokladáme pre používateľa s identifikátorom 1 existenciu účtu s číslom 178106304. V tomto príklade využívame autorizovaný pohľad definovaný v prílohe 5. Predpokladáme, že na účte je 5000 korún.

```
-- Získame soľ pre zamestnanca s prihlasovacím menom 'teller'.
-- Zreťazíme túto soľ s heslom a na výsledok použijeme hešovaciu
-- funkciu. Pomocou prihlasovacieho mena a hešu sa autentifikujeme, čím
-- inicialuzujeme aplikačný kontext
execute bank_is.employee_context_pkg.init_context('teller', hash( 'password') ||
bank_is.employee_context_pkg.get_salt('teller')) ;

-- Tieto dopyty vratia prázdnu monožinu záznamov, pretože sme
-- nenastavili príslušné údaje v aplikačnom kontexte
SELECT * FROM bank_is.v_all_client ;
SELECT * FROM bank_is.v_all_account;

-- Nastavíme teda identifikátor klienta.
execute bank_is.employee_context_pkg.set_client_id(1);

-- V tomto momente vie zamestnanec zíkať informácie o používateľskom
-- účte zákazníka a jeho bankových účtoch.
SELECT * FROM bank_is.v_all_client ;
SELECT * FROM bank_is.v_all_account;

-- Nastavíme účet, nad ktorým chceme vytvárať transakcie v aplikačnom
-- kontexte, aby sme mohli nad ním operovať.
execute bank_is.employee_context_pkg.set_account_no('178106304') ;

-- Pracovník može potom vložiť transakciu, ktorá prevedie z nastaveného -- účtu
zákazníka 2000 SK na učet 111111111.
INSERT INTO
bank_is.v_all_i_transaction(id,account_from,account_to,var_sym,amount,created,st
atus,trans_type)
VALUES (1, '178106304', '111111111', null, 2000, SYSDATE, 'NEW', 'TELLER_PAY') ;

-- Pokiaľ by chcel pracovník nasledujúcu vykonať transakciu (previesť
-- 11000 kotún) príkaz skončí chybou, pretože zákazník nemá na účte
-- dostatok prostriedkov.
INSERT INTO
bank_is.v_all_i_transaction(id,account_from,account_to,var_sym,amount,created,st
atus,trans_type)
VALUES (2, '178106304', '111111111', null, 11000, SYSDATE, 'NEW', 'TELLER_PAY') ;


-- Pokiaľ by chcel pracovník nasledujúcu transakciu iného typu,
-- napríklad EBANK_PAY. Vloženie sa taktiež nepodarí.
```

```
INSERT INTO
bank_is.v_all_i_transaction(id,account_from,account_to,var_sym,amount,created,st
atus,trans_type)
VALUES (2, '178106304', '111111111', null, 2000, SYSDATE, 'NEW', 'EBANK_PAY') ;
```

Implementáciu podobných príkladov je možné nájsť v prílohe 2.

Príloha 7: Porovnanie výkonnosti implementácií TDE

Počet riadkov	***Meranie***					***Medián***	***Priemer***
	č. 1	***č. 2***	***č. 3***	***č. 4***	***č. 5***		
100 000	0.71	0.72	0.73	0.73	0.72	0.72	0.72
200 000	1.43	1.43	1.42	1.43	1.44	1.43	1.43
300 000	2.15	2.16	2.14	2.13	2.14	2.14	2.14
400 000	2.84	2.85	2.85	2.84	2.83	2.84	2.84
500 000	3.58	3.54	3.56	3.56	3.51	3.56	3.55

Tabuľka 1: Čas v sekundách potrebný na vykonanie dopytu prechádzajúceho celú tabuľku s deklaratívne implementovaným TDE.

Počet riadkov	***Meranie***					***Medián***	***Priemer***
	č. 1	***č. 2***	***č. 3***	***č. 4***	***č. 5***		
100 000	4.26	4.24	4.22	4.23	4.23	4.23	4.24
200 000	8.53	8.51	8.47	8.50	8.45	8.50	8.49
300 000	12.67	12.70	12.69	12.71	12.72	12.70	12.70
400 000	16.82	16.82	16.84	16.93	16.92	16.84	16.87
500 000	21.14	21.01	21.03	21.02	21.04	21.03	21.05

Tabuľka 2: Čas v sekundách potrebný na vykonanie dopytu prechádzajúceho celú tabuľku s TDE implementovaným pomocou pohľadov.

Počet riadkov	***Meranie***					***Medián***	***Priemer***
	č. 1	***č. 2***	***č. 3***	***č. 4***	***č. 5***		
100 000	0.02	0.01	0.01	0.01	0.02	0.01	0.01
200 000	0.03	0.02	0.03	0.02	0.02	0.02	0.02
300 000	0.04	0.03	0.04	0.04	0.04	0.04	0.04
400 000	0.05	0.05	0.04	0.04	0.04	0.04	0.04
500 000	0.05	0.05	0.06	0.05	0.05	0.05	0.05

Tabuľka 3: Čas v sekundách potrebný na vykonanie dopytu prechádzajúceho celú tabuľku bez TDE.

Použitá šifra	*Meranie*					*Medián*	*Priemer*
	č. 1	*č. 2*	*č. 3*	*č. 4*	*č. 5*		
AES128	24.81	26.13	27.65	25.15	23.73	25.15	25.49
AES192	27.19	26.72	25.19	25.84	27.18	26.72	26.42
AES256	26.18	26.49	26.83	28.69	26.49	26.49	26.94

Tabuľka 4: Čas v sekundách potrebný na vloženie 100 000 riadkov do tabuľky s transparentným šifrovaním pomocou pohľadov.

Použitá šifra	*Meranie*					*Medián*	*Priemer*
	č. 1	*č. 2*	*č. 3*	*č. 4*	*č. 5*		
AES128	5.79	5.06	6.94	5.82	5.92	5.92	5.91
AES192	5.89	6.09	6.21	6.98	7.09	6.98	6.45
AES256	9.61	9.00	9.94	9.06	8.91	9.06	9.3

Tabuľka 5: Čas v sekundách potrebný na vloženie 100 000 riadkov do tabuľky s deklaratívne implementovaným transparentným šifrovaním implementovaným deklaratívne.

Meranie					*Medián*	*Priemer*
č. 1	*č. 2*	*č. 3*	*č. 4*	*č. 5*		
4.30	4.27	3.64	3.72	3.76	3.72	3.94

Tabuľka 6: Čas v sekundách potrebný na vloženie 100 000 riadkov do tabuľky, bez transparentného šifrovania.

Testy boli vykonané na nasledujúcej konfigurácii:

Procesor: Intel Core Duo T2250 (1.73 Ghz)

Pamäť: 2x512 DDR2

Disk: SATA 150, 5.400 ot./min

Príloha 8: Popis parametrov funkcie ***ADD_POLICY***

Meno parametra	*Popis*
OBJECT_SCHEMA	Meno schémy, v ktorej sa nachádza objekt, na ktorý aplikujeme politiku. Ak neuvedieme, použije sa aktuálna schéma.
OBJECT_NAME	Meno pohľadu alebo tabuľky, na ktorú aplikujeme politiku.
POLICY_NAME	Meno politiky.
FUNCTION_SCHEMA	Schéma, v ktorej sa nachádza funkcia implementujúca politiku. Ak neuvedieme, použije sa aktuálna schéma
POLICY_FUNCTION	Meno funkcie implementujúcej pditiku.
STATEMENT_TYPES	Zoznam príkazov, na ktoré sa politika aplikuje. Ak neuvedieme, politika sa aplikuje na všetky príkazy okrem príkazu INDEX.
UPDATE_CHECK	Parameter je aplikovateľný iba na príkazy INSERT a UPDATE, a má rovnaký význam ako klauzula WITH CHECK OPTION v pohľadoch. Prednastavená hodnota je FALSE.
ENABLE	Príznak, či je politika aktivovaná. Prednastavená hodnota je TRUE.
POLICY_TYPE	Ak je tento parameter NULL, použije sa STATIC_POLICY. Inak sa použije typ politiky definovaný týmto parametrom.
LONG_PREDICATE	Príznak, či môže mať návratový predikát väčšiu dĺžku ako 4000 znakov. Ak je TRUE, dĺžka predikátu môže byť až 32767 znakov. Prednastavená hodnota je FALSE.
SET_RELEVANT_COLS	Tento parameter môže obsahovať množinu názvov stĺpcov. Politika sa aplikuje, ak sa nejaký z týchto stĺpcov vyskytne v príkaze zadefinovanom v parametri STATEMENT_TYPES.
SET_RELEVANT_COLS _OPT	Pokiaľ má tento parameter hodnotu DBMS_RLS.ALL_ROWS, zamaskujú sa hodnoty stĺpcov uvedených v predchádzajúcom parametri SET_RELEVANT_COLS.

Príloha 9: Príklad dynamickej politiky.

```
CREATE OR REPLACE FUNCTION transaction_insert_policy(p_schema IN VARCHAR2,
                                                     p_object IN VARCHAR2)
                          RETURN VARCHAR2 IS
  l_predicate VARCHAR2(4000) := NULL;
BEGIN
  --Najprv do predikátu dáme podmienky, ktoré musí spĺňať každý.
  l_predicate := 'account_from = sys_context(''bank_context2'', ''account_no'')
                 AND status=''NEW''';
  l_predicate := l_predicate || ' AND ( amount <= (SELECT balance
   FROM   account acc
   WHERE  acc.account_no = sys_context(''bank_context2'', ''account_no'')) -
  (SELECT nvl(SUM(amount), 0)
   FROM   TRANSACTION
   WHERE  account_from = sys_context(''bank_context2'', ''account_no'')
   AND    status = ''NEW'') )';

  -- Na základe aplikačného kontextu rozlíšime, kto spúšťa politiku, a teda
  -- akými podmienkami ešte doplníme vysledný predikát.

  -- Ak spúšťa dopyt zamestnanec, umožníme mu len platbu typu TELLER_PAY
  IF sys_context('bank_context2', 'user_type') = 'EMP' THEN
    RETURN l_predicate || ' AND trans_type = ''TELLER_PAY''';

  -- Ak spúšťa dopyt zákazník, umožníme mu len platbu EBANK_PAY a overíme, či
  -- táto platba nepresiahne denný limit.
  ELSIF sys_context('bank_context2', 'user_type') = 'CLIENT' THEN
    RETURN l_predicate || ' AND (trans_type = ''EBANK_PAY'' AND
      amount <= (SELECT day_limit
                 FROM   account acc
                 WHERE  acc.account_no = sys_context(''bank_context2'',
                                                     ''account_no'')) -
                (SELECT nvl(SUM(amount), 0)
                 FROM   TRANSACTION
                 WHERE  account_from = sys_context(''bank_context2'',
                                                   ''account_no'')
                 AND    created > trunc(SYSDATE)
                 AND    status <> ''CANCELED''
                 AND    trans_type = ''EBANK_PAY''))';

  -- Ak spúšťa dopyt terminál umožnime mu len paltby typy CARD_ATM a CARD_PAY
  -- a overíme, či transakcia nepresiahla denny limit.
  ELSIF sys_context('bank_context2', 'user_type') = 'TERMINAL' THEN

    RETURN l_predicate || ' AND ((trans_type = ''CARD_ATM'' AND
      amount <= (SELECT atm_limit
                 FROM   card
                 WHERE  account_no = sys_context(''bank_context2'',
                                                 ''account_no'')
                 AND    valid = 1) -
                (SELECT nvl(SUM(amount), 0)
                 FROM   TRANSACTION
                 WHERE  account_from = sys_context(''bank_context2'',
                                                   ''account_no'')
                 AND    created > trunc(SYSDATE)
                 AND    status <> ''CANCELED''
```

```
                AND    trans_type = ''CARD_ATM'')) OR
      (trans_type = ''CARD_PAY'' AND
      amount <= (SELECT pay_limit
                FROM   card
                WHERE  account_no = sys_context(''bank_context2'',
                                                ''account_no'')
                AND    valid = 1) -
               (SELECT nvl(SUM(amount), 0)
                FROM   TRANSACTION
                WHERE  account_from = sys_context(''bank_context2'',
                                                  ''account_no'')
                AND    created > trunc(SYSDATE)
                AND    status <> ''CANCELED''
                AND    trans_type = ''CARD_PAY'')))';
  END IF;

  -- Ak spúšťal dopyt niekto iný vrátime predikát, ktorý sa vyhodnotí ako FALSE
  RETURN '0=1';
END transaction_insert_policy;
```

Táto politika je analógiou pohľadu z prílohy 5. Popis v prílohe 5 je možné aplikovať aj na túto politiku.

Poznámka: Niektoré kľúčové slová nie sú zvýraznené, pretože sú súčasťou reťazca.