UNIVERZITET U BEOGRADU
FAKULTET ORGANIZACIONIH NAUKA

Administracija baza podataka

ADMINISTRACIJA ORACLE SUBP-a

Tema: **Transparent Data Encrypton**

(Kriptovanje na nivou kolone tabele i tablespace-a)

Mentor: Prof. dr Nenad Aničić

Student: Vojislav Ristivojević 2016/3079

Beograd, 2017.

Sadržaj

Uvod

Administracija baze podataka podrazumeva upravljanje i kontrolu nad fizičkim dizajnom i implementacijom konkretnog sistema u jednom preduzeću, uključujući kontrolu bezbednosti i integriteta podataka, praćenje performansi sistema i eventualne izmene i reorganizacije samog sistema.[4] Svaka organizacija koja koristi neki SUBP zahteva kontinuirano i stručno bavljenje svim aspektima implementacije, korišćenja i održavanja baze, a ova uloga je najčešće poverena jednoj ili grupi osoba kojima je administracija baze jedina dužnost u kompaniji.[11]

Segment administratorskih aktivnosti kojim ćemo se baviti u ovom radu obuhvata pre svega pitanja bezbednosti podataka, odnosno zaštitu poverljivosti i privatnosti istih. Poverljivost podataka (*confidentiality*) podrazumeva potrebu da kritični podaci za jednu organizaciju ostanu dostupni samo određenim korisnicima, dok privatnost podataka (*privacy*) označava neophodnost čuvanja informacija o pojedincima. Ugrožavanje bilo kojeg od ovih principa može imati teške finansijske i pravne posledice po kompaniju zaduženu za bezbednost informacija.[4]

Prikazaćemo kako je korišćenjem transparentne enkripcije podataka u Oracle SUBP-u moguće predupredити pokušaje zaobilaženja internih bezbednosnih mehanizama putem direktnog pristupa fajlovima u kojima se čuvaju podaci, bez obzira da li su trenutno u upotrebi ili su pohranjeni u obliku rezervnih kopija.

U prvom poglavlju ovog rada biće reči o osnovnim administratorskim zaduženjima, sa akcentom na sprovođenje istih u Oracle SUBP okruženju, te će ukratko biti opisan podmodel jednog poslovnog sistema koji će biti korišćen u daljem radu.

U drugom delu rada prikazana je i opisana funkcionalnost transparentne enkripcije podataka na nivou kolone i tabelarnog prostora (*Transparent data encryprion*), tehnologije koju pomenuti SUBP koristi radi zaštite podataka na nivou fizičkog medija.

1. Osnovne administratorske aktivnosti

Oracle SUBP je veoma robustan i pouzdan sistem, ali istovremeno složen i sveobuhvatan. Paralelno sa razvojem konkretnog sistema, menja se i definicija zaduženja administratora baze podataka. Održavanje baze podataka operativnom i efikasnom obuhvata zadatke koji se ne mogu obaviti samo jednom, npr. prilikom inicijalne konfiguracije, već različite aktivnosti zahtevaju određenu dinamiku koja zavisi od konkretnih karakteristika datog sistema.[15]

Iako je moguće deo ovih aktivnosti automatizovati ili bar zabeležiti u listi tipičnih zadataka, svakodnevni problemi i novi zahtevi korisnika zahtevaju da se lista aktivnosti stalno ažurira i prilagođava situaciji.[15]

1.1. Rad sa logičkom/fizičkom strukturom baze podataka

Na početku rada sa novom bazom podataka, odnosno šemom (*schema*), kreiraćemo novog korisnika *voja_3079* i dodeliti mu potrebne sistemske i objektne privilegije za dalji rad.

```
CREATE USER voja_3079 IDENTIFIED BY student;
```

Uloge su sredstvo pomoću kojeg administratori baze podataka mogu da dodeljuju mogu da dodeljuju razne objektne i sistemske privilegije jednom ili grupi korisnika.[15] Postoji nekoliko unapred definisanih uloga koje se često koriste prilikom dodavanja novih korisnika (npr. CONNECT, RESOURCE itd.), a koje u sebi sadrže privilegije kao što su CREATE SESSION, CREATE TABLE, CREATE TRIGGER, CREATE TYPE itd. Iako nije uobičajeno da se sva ova ovlašćenja daju korisnicima baze, za potrebe našeg projekta ćemo ih iskoristiti.

```
GRANT CONNECT, RESOURCE TO voja_3079;
```

Pre nego što korisnik pristupi kreiranju tabela u okviru svoje šeme, od strane administratora će biti kreiran poseban logički prostor (*tablespace*) a korisniku dodeljene adekvatne privilegije za korišćenje istog.

```
CREATE SMALLFILE TABLESPACE tbs_3079
DATAFILE 'c:\app\oradata\orcl\tbs_3079.dbf' SIZE 50m
AUTOEXTEND ON NEXT 1024k;
ALTER USER voja_3079 DEFAULT TABLESPACE tbs_3079;
```

U kreiranoj šemi, korisnik će napraviti tabele potrebne za dalji rad, odnosno funkcionisanje aplikacije. Baza koja će biti korišćena u ovom radu deo je projektnog zadatka na predmetu “Baze podataka 2”, a odnosni se na prikaz podmodela “Prodaja” poslovnog procesa firme “VWG Inženjering iz Beograda”.

```
CREATE TABLE "KUPAC"
   ("SIFRAKUPCA" NUMBER(5,0) NOT NULL,
  "IMEKUPCA" VARCHAR2(30 BYTE) NOT NULL,
  "ADRESA" VARCHAR2(50),
  "PIB"    NUMBER(8),
  "BRTEL"  NUMBER(8),
  "EMAIL"  VARCHAR2(30));

CREATE TABLE "CENA"
   ("DATUM" TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP NOT NULL,
  "SIFRAARTIKLA" NUMBER(5,0) NOT NULL,
  "CENA" NUMBER(5,0),
  "STOPAPDV" NUMBER(5,0),
  "NAZIVARTIKLA" VARCHAR2(30 BYTE));

CREATE TABLE "PONUDA"
   ("BRPONUDE" NUMBER(5,0) NOT NULL,
  "DATUM" TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP,
  "SIFRAKUPCA" NUMBER(5,0),
  "IMEKUPCA" VARCHAR2(30 BYTE),
  "UKUPNI_IZNOS" NUMBER DEFAULT 0);

CREATE TABLE "STAVKAPONUDE"
   ("RBR" NUMBER(5,0) NOT NULL,
  "BRPONUDE" NUMBER(5,0) NOT NULL,
  "KOLICINA" NUMBER(5,0),
  "SIFRAARTIKLA" NUMBER(5,0));
```

Na tabelu ARTIKAL primenićemo drugačiju organizaciju strukture, odnosno izvesti vertikalno particionisanje kako bismo je podelili na dve tabele ARTIKAL i ARTIKAL_DETALJI. Nad tim tabelama ćemo kreirati pogled i adekvatne trigere kako bi bile moguće sve potrebne DML operacije.

```
CREATE TABLE "ARTIKAL"
   ("SIFRAARTIKLA" NUMBER(5,0) NOT NULL,
  "NAZIVARTIKLA" VARCHAR2(30 BYTE) NOT NULL);

CREATE TABLE "ARTIKAL_DETALJI"
```

```
   ("JEDMERE" VARCHAR2(30 BYTE),
  "OPISARTIKLA" VARCHAR2(30 BYTE),
  "SIFRAARTIKLA" NUMBER NOT NULL,
  "TRENUTNA_CENA" NUMBER);
```

Pogled ARTIKAL_VIEW:

```
CREATE OR REPLACE FORCE VIEW "ARTIKAL_VIEW"
("SIFRAARTIKLA", "NAZIVARTIKLA", "JEDMERE", "OPISARTIKLA",
"TRENUTNA_CENA")
AS
select
a.sifraartikla, a.nazivartikla, ad.jedmere, ad.opisartikla,
ad.TRENUTNA_CENA
from ARTIKAL a, ARTIKAL_DETALJI ad
where ad.SIFRAARTIKLA = a.SIFRAARTIKLA;
```

Trigeri za ARTIKAL_VIEW:

```
CREATE OR REPLACE TRIGGER "ARTIKAL_DELETE"
instead of delete ON artikal_view
for each row
begin
delete from artikal_detalji where sifraartikla
= :old.sifraartikla;
delete from artikal where sifraartikla = :old.sifraartikla;
end ARTIKAL_DELETE;

CREATE OR REPLACE TRIGGER "ARTIKAL_INSERT"
instead of insert ON artikal_view
for each row
declare x number;
begin
insert into artikal
(NAZIVARTIKLA) values (:NEW.NAZIVARTIKLA);
select sifraartikla
into x
from ARTIKAL
where NAZIVARTIKLA = :NEW.NAZIVARTIKLA;
insert into artikal_detalji
(jedmere, opisartikla, sifraartikla) values
(:NEW.jedmere, :NEW.opisartikla, x);
end artikal_insert;
```

```
CREATE OR REPLACE TRIGGER "ARTIKAL_UPDATE"
instead of update ON artikal_view
for each row
begin
update artikal set nazivartikla=:new.nazivartikla where
sifraartikla = :new.sifraartikla;
end ARTIKAL_UPDATE;
```

Na posebnim lokacijama ćemo smeštati podatke iz tabele, a na posebnim podatke iz indeksa, odnosno osim prethodno kreiranog, napravićemo još jedan tabelarni prostor samo za indekse, pre svega zbog eventualnog čuvanja na bržem disku, što može poboljšati brzinu pristupa.

```
CREATE TABLESPACE TBS_ZA_INDEXE
DATAFILE 'c:\app\oradata\orcl\tbs_za_indexe.dbf'
SIZE 32m
AUTOEXTEND ON NEXT 32m;
```

1.2. Rad sa objektima

Kreiraćemo ograničenja *primary key*, *foreign key*, *not nul* i *check*, tamo gde poslovna logika to zahteva. Takođe, neophodno je kreirati sekvence za automatsko povećavanje vrednosti u pojedinim kolonama, pre svega onim koje se koriste za numerisanje odnosno dodeljivanje identifikatora. Ukoliko opcija automatskog kreiranja sekvence koju nudi Oracle nije dovoljna za konkretni slučaj, neophodno je ručno definisati kod za numeraciju korišćenjem PL/SQL jezika i postaviti ga u odgovarajući triger.

Kreiraćemo *primary key* ograničenja:

```
ALTER TABLE ARTIKAL ADD CONSTRAINT "ARTIKAL_PK"
PRIMARY KEY ("SIFRAARTIKLA") ENABLE;

ALTER TABLE CENA ADD CONSTRAINT "CENA_PK"
PRIMARY KEY ("DATUM", "SIFRAARTIKLA") ENABLE;

ALTER TABLE PONUDA ADD CONSTRAINT "PONUDA_PK"
PRIMARY KEY ("BRPONUDE") ENABLE;

ALTER TABLE STAVKAPONUDE ADD CONSTRAINT "STAVKAPONUDE_PK"
```

```
PRIMARY KEY ("RBR", "BRPONUDE") ENABLE;

ALTER TABLE KUPAC ADD CONSTRAINT "KUPAC_PK"
PRIMARY KEY ("SIFRAKUPCA") ENABLE;
```

Foreign key ograničenja:

```
ALTER TABLE PONUDA ADD CONSTRAINT "PONUDA_FK1"
FOREIGN KEY ("SIFRAKUPCA")
REFERENCES "KUPAC" ("SIFRAKUPCA") ENABLE;

ALTER TABLE STAVKAPONUDE ADD CONSTRAINT "STAVKAPONUDE_FK1"
FOREIGN KEY ("SIFRAARTIKLA")
REFERENCES "ARTIKAL" ("SIFRAARTIKLA") ENABLE;

ALTER TABLE ARTIKAL_DETALJI ADD CONSTRAINT
"ARTIKAL_DETALJI_FK1" FOREIGN KEY ("SIFRAARTIKLA")
REFERENCES "ARTIKAL" ("SIFRAARTIKLA") ON DELETE CASCADE ENABLE;

ALTER TABLE CENA ADD CONSTRAINT "CENA_FK1"
FOREIGN KEY ("SIFRAARTIKLA")
REFERENCES "ARTIKAL" ("SIFRAARTIKLA") ON DELETE CASCADE ENABLE;
```

Automatski kreirane indekse za sve kolone sa *primary key* ograničenjem ćemo premestiti na drugi tabelarni prostor:

```
ALTER INDEX KUPAC_PK REBUILD TABLESPACE "TBS_ZA_INDEXE";

ALTER INDEX ARTIKAL_PK REBUILD TABLESPACE "TBS_ZA_INDEXE";

ALTER INDEX CENA_PK REBUILD TABLESPACE "TBS_ZA_INDEXE";

ALTER INDEX PONUDA_PK REBUILD TABLESPACE "TBS_ZA_INDEXE";

ALTER INDEX STAVKAPONUDE_PK REBUILD TABLESPACE "TBS_ZA_INDEXE";
```

Primer CHECK ograničenja za kolonu PIB u tabeli KUPAC kojom se zahteva osmocifreni broj:

```
ALTER TABLE KUPAC
ADD CONSTRAINT KUPAC_CHK1 CHECK
(PIB>=10000000 AND PIB<=99999999)
ENABLE;
```

Primer sekvence za tabelu PONUDA:

```
CREATE SEQUENCE  "VOJA_3079"."PONUDA_SEQ"  MINVALUE 1
NOMAXVALUE INCREMENT BY 1 START WITH 1 CACHE 20 NOORDER
NOCYCLE ;
```

Triger koji koristi sekvencu PONUDA_SEQ za automatsko uvećanje vrednosti kolone BRPONUDE:

```
CREATE OR REPLACE TRIGGER PONUDA_TRG
BEFORE INSERT ON PONUDA
FOR EACH ROW
BEGIN
  <<COLUMN_SEQUENCES>>
  BEGIN
    IF INSERTING AND :NEW.BRPONUDE IS NULL THEN
      SELECT PONUDA_SEQ.NEXTVAL INTO :NEW.BRPONUDE FROM
SYS.DUAL;
    END IF;
  END COLUMN_SEQUENCES;
END;
```

Triger koji automatski numeriše vrednosti stavke ponude u tabeli STAVAKAPONUDE, bez korišćenja sekvence:

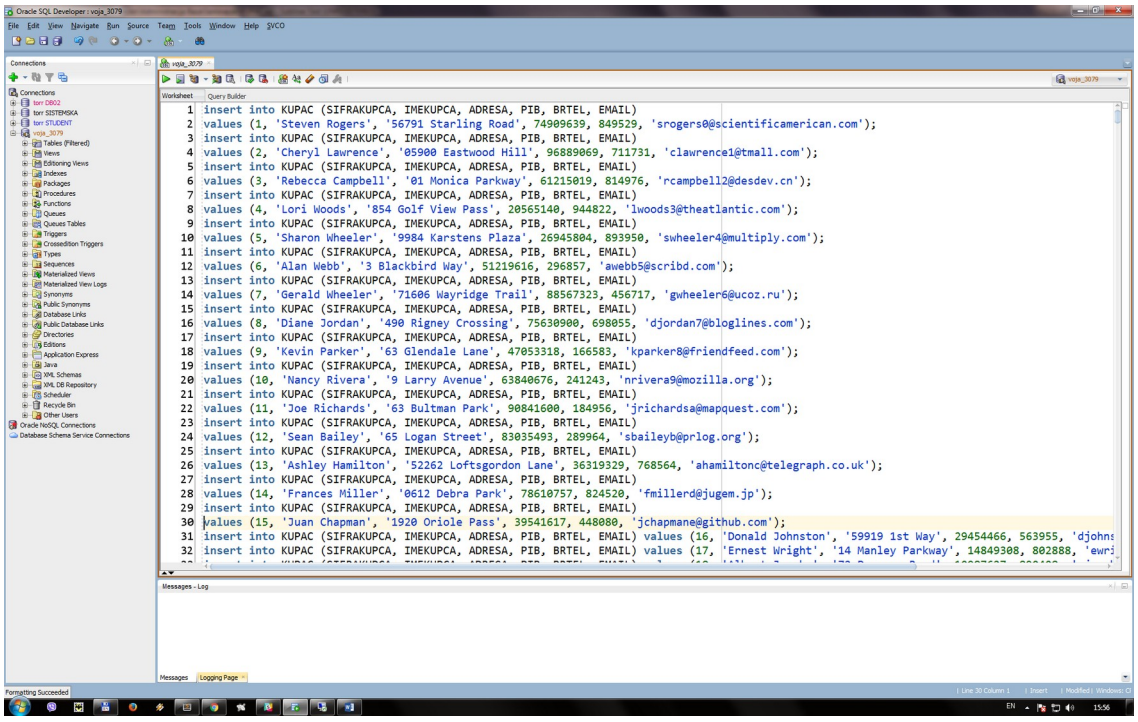
```
CREATE OR REPLACE TRIGGER "STAVKAPONUDE_TRG3"
BEFORE INSERT ON STAVKAPONUDE
FOR EACH ROW
DECLARE tmp_rbr NUMBER;

  BEGIN
    IF INSERTING AND :NEW.RBR IS NULL THEN
      SELECT COUNT(RBR) INTO tmp_rbr FROM STAVKAPONUDE WHERE
BRPONUDE = :NEW.BRPONUDE;
      tmp_rbr := tmp_rbr +1;
      :NEW.RBR := tmp_rbr;
    END IF;
  END;
```

Ukoliko tokom korišćenja baze, odnosno u slučaju promene zahteva korisnika dođe do potrebe za izmenom pojedinih delova sistema, iste je moguće izvršiti predviđenim DML operacijama. Nakon sto je kreirana, struktura tabele se može menjati dodavanjem kolona, izmenom tipa kolone, preimenovanjem, dodelom podrazumevanih vrednosti itd.

```
ALTER TABLE ARTIKAL_DETALJI
MODIFY (OPISARTIKLA VARCHAR2(50 BYTE) );
```

```
ALTER TABLE ARTIKAL_DETALJI
MODIFY (JEDMERE DEFAULT 'kom' );

ALTER TABLE PONUDA DROP COLUMN "IMEKUPCA";
```

Za potrebe testiranja pojedinih objekata baze, npr. uskladištenih procedura, odnosno provere same aplikacije, neophodno je popuniti tabele podacima. Ovo se može izvršiti ručnim unosom, odnosno korišćenjem posebne skripte ukoliko je potrebna veća količina podataka. Za potrebe našeg projekta, iskoristićemo javno dostupni generator *Mockaroo* (https://www.mockaroo.com), a ovako dobijeni .sql fajl ćemo uvesti u bazu koristeći SQL Developer.



Slika 1 - Automatski generisani podaci

1.3. Plan izvršavanja

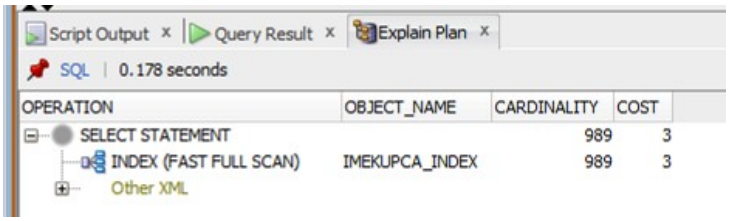
Indeksi su jedan od najvažnijih alata koji se koriste za poboljšanje performansi SUBP-a. Njihove različite vrste mogu se prilagoditi potrebama, odnosno konkretnim upitima koji se od strane korisnika i aplikacija upućuju ka bazi.[14]

Radi ilustracije primera plana izvršavanja, napravićemo indeks nad kolonom IMEKUPCA koja nema *pk*, *not null* ni *unique* ograničenje.

```
CREATE INDEX IMEKUPCA_INDEX ON KUPAC (IMEKUPCA)
TABLESPACE TBS_ZA_INDEXE;
```

Isti indeks se može koristiti na različite načine: *full index scan*, *full table scan*, *range scan*, a zavisno od samog upita, sto je moguće videti iz plana izvršavanja.

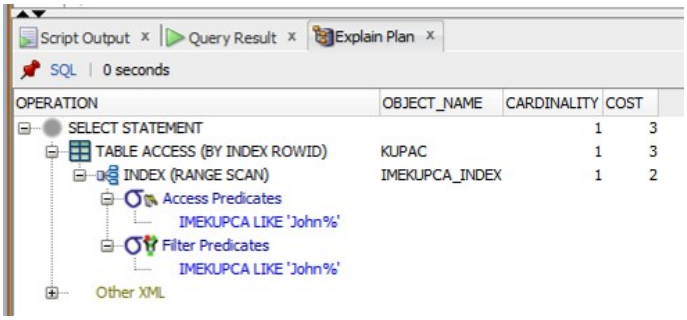
```
SELECT imekupca FROM kupac;
```



Slika 2 – Full index scan

Indeks IMEKUPCA_INDEX će biti iskorišćen u *full index scan* modu, jer se traže svi redovi kolone nad kojom je indeks napravljen. Optimizer bira ovakav način pretrage kada utvrdi da će biti neophodno vratiti većinu redova koji postoje u bazi, a informacije koje čine rezultat se nalaze u samom indeksu. S obzirom da je indeks najčešće manji od cele tabele, optimizer će zaključiti da je pretraga celog indeksa ipak efikasnija od pretrage cele tabele.[8]

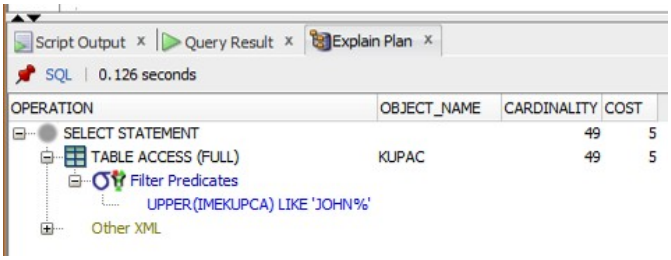
```
SELECT PIB FROM kupac
WHERE imekupca LIKE 'John%';
```



Slika 3 – Range scan index

Indeks IMEKUPCA_INDEX će biti iskorišćen u *range scan* modu, jer se traže samo određeni redovi kolone nad kojom je indeks napravljen. Ovaj slučej se javlja kada optimizer proceni da je efikasno koristiti indeks da bi se vratilo više redova koje je upit zahtevao. Ovo je jedan od najčešće korišćenih načina pristupa indeksu.[8]

```
SELECT PIB FROM kupac WHERE UPPER(imekupca) LIKE 'JOHN%';
```



Slika 4 – Full table scan

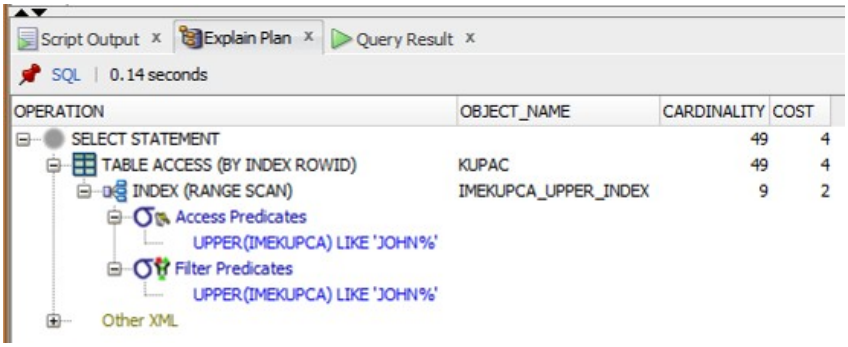
Indeks IMEKUPCA_INDEX neće biti iskorišćen, već će pretraga biti izvršena upotrebom *full table scan* pristupa. Bez obzira što je kolona indeksirana, optimizer neće iskoristiti indeks jer je pretraga zapravo izvršena nad rezultatom funkcije UPPER(imekupca). S obzirom da ne postoji funkcionalni indeks predviđen za ovakvu pretragu, biće izvršena direktna pretraga cele tabele.[8]

Da bi prethodna pretraga koristila indeksiranje, kreiraćemo funkcionalni indeks prilagođen konkretnoj pretrazi koju ćemo vršiti.

```
CREATE INDEX IMEKUPCA_UPPER_INDEX ON KUPAC (UPPER(IMEKUPCA))
TABLESPACE TBS_ZA_INDEXE;
```

Kada ponovimo prethodnu pretragu, videćemo da će novokreirani indeks biti iskorišćen u *range scan* modu.

```
SELECT PIB FROM kupac WHERE UPPER(imekupca) LIKE 'JOHN%';
```



Slika 5 - Range scan index

1.4. Sigurnost

Pošto je baza dizajnirana i implementirana, programerima i korisnicima je neophodno omogućiti da pristupaju i menjaju podatke. Da bi se sprečile neovlašćeni uvidi i izmene, potrebno je omogućiti da samo određeni korisnici imaju pravo pristupa. Održavanje adekvatnih prava pristupa je jedno od osnovih zaduženja administratora. Ovo se najčešće postiže korišćenjem internih mogućnosti SUBP-a u obliku GRANT i REVOKE komandi, kao i putem grupno dodeljenih privilegija.[11]

Iz sistemske konekcije ćemo napraviti novog korisnika, *voja_2016_3079* i dodeliti mu minimalne privilegije potrebne za rad sa bazom (CONNECT i RESOURCE uloge).

```
CREATE USER voja_2016_3079 IDENTIFIED BY student;
```

Zatim ćemo kreirati još jednog korisnika *test_user*, kome će početni korisnik dodeliti određene privilegije nad pojedinim objektima iz svoje šeme.

```
CREATE USER test_user IDENTIFIED BY student;
```

Na primeru ćemo prikazati prava pristupa korisnika *test_user* objektima *voja_2016_3079*:
Prvo će korisnik *voja_2016_3079* kreirati tabelu i popuniti je test podacima:

```
  CREATE TABLE "ZAPOSLENI"
   (    "RBR" VARCHAR2(20 BYTE) NOT NULL ENABLE,
   "IME" VARCHAR2(20 BYTE) NOT NULL ENABLE,
   "PREZIME" VARCHAR2(20 BYTE) NOT NULL ENABLE,
   "STATUS" NUMBER,
    CONSTRAINT "TABLE1_PK" PRIMARY KEY ("RBR") ENABLE,
    CONSTRAINT "TABLE1_CHK1" CHECK (status in (1,2))
ENABLE  ) ;
```

Zatim će isti korisnik dodeliti SELECT privilegiju na celoj tabeli ZAPOSLENI korisniku *test_user*, dok će mu UPDATE operacija biti dozvoljene isključivo na koloni STATUS. Operacije INSERT i DELETE podrazumevano neće biti dodeljene.

```
GRANT SELECT ON ZAPOSLENI TO TEST_USER;
GRANT UPDATE (STATUS) ON ZAPOSLENI TO TEST_USER;
```

Kao što je očekivano, *test_user* će moći da pročita podatke korisnika *voja_2016_3079*:

```
SELECT * from "VOJA_2016_3079"."ZAPOSLENI";

UPDATE  "VOJA_2016_3079"."ZAPOSLENI"
SET STATUS = '2'
WHERE RBR = '10';
```

A bilo koji drugi pokušaj izmene će biti propraćen greškom:

```
SQL Error: ORA-01031: insufficient privileges
```

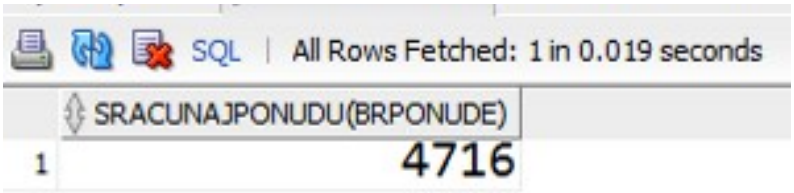
1.5. Napredne funkcije

Korisnički definisane agregatne funkcije služe za implementaciju funkcionalnosti koje nisu već dostupne u SQL-u. Jednom kreirane, mogu se koristiti u naredbama kao i sve ostale predefinisane funkcije. Kreiraćemo korisnički definisanu agregatnu funkciju koja sračunava ukupnu vrednost ponude na osnovu pojedinačnih stavki.

```
--1. Kreiramo Object type
create or replace type UkupnaVrednost as object (
  br_pon number(10),

  static function ODCIAggregateInitialize(ctx IN OUT
UkupnaVrednost)
        return number,

  member function ODCIAggregateIterate(self IN OUT
UkupnaVrednost,
                                       value IN number)
        return number,

  member function ODCIAggregateTerminate(self IN OUT
UkupnaVrednost,
                                        returnValue OUT number,
                                        flags IN number)
        return number,

  member function ODCIAggregateMerge(self IN OUT
UkupnaVrednost,
                                     ctx2 IN UkupnaVrednost)
        return number
);


-----------------------------------------------------------
---
```

```
--2. Implementiramo BODY tipa

create or replace type body UkupnaVrednost is

  static function ODCIAggregateInitialize(ctx IN OUT
UkupnaVrednost)
        return number is
        begin
          ctx:= UkupnaVrednost(null);
          return odciconst.Success;
        end;

  member function ODCIAggregateIterate(self IN OUT
UkupnaVrednost,
                                       value IN number)
        return number is
        begin
          self.br_pon := value;
            select sum(c.cena*sp.kolicina) into self.br_pon
            from STAVKAPONUDE sp join cena c on
            (c.sifraartikla=sp.sifraartikla)
            where BRPONUDE = value;
          return odciconst.Success;
        end;

  member function ODCIAggregateTerminate(self IN OUT
UkupnaVrednost,
                                        returnValue OUT number,
                                        flags IN number)
        return number is
        begin
          returnvalue := self.br_pon;
          return odciconst.Success;
        end;

  member function ODCIAggregateMerge(self IN OUT
UkupnaVrednost,
                                     ctx2 IN UkupnaVrednost)
        return number is
        begin
          self.br_pon := self.br_pon + ctx2.br_pon ;
          return odciconst.Success;
```

```
        end;
end;


--3. Kreiramo funkciju
CREATE OR REPLACE FUNCTION SracunajPonudu (input number)
RETURN number
PARALLEL_ENABLE AGGREGATE USING UkupnaVrednost;

----------------------------------------
-- TESTIRANJE

select  SracunajPonudu(BRPONUDE) from STAVKAPONUDE where
BRPONUDE = 1;
```



Slika 6 – Rezultat KD agregatne funkcije

1.6. Backup i oporavak

Administrator baze mora biti spreman da povrati podatke u slučaju bilo kakvog problema, koji mogu varirati od proste greške u nekoj aplikaciji do elementarne nepogode koja u potpunosti onesposobi deo sistema. Ipak, većina grešaka, oko 80%, koje dovode do gubitka podataka su posledica greške u softveru ili greške korisnika, dok su hardverski otkazi danas daleko ređi. U svakom slučaju, administrator mora biti spreman da u svakom trenutku osposobi bazu u najkraćem mogućem roku. [11]

Odgovoran odnos prema administraciji baze podataka podrazumeva prihvatanje činjenice da u svakom trenutku može doci do otkaza sistema. Sam pad sistema nije kritični deo, već je od najveće važnosti imati mogućnost oporavka. Osim kvalitetnih i ažurnih rezervnih kopija, takođe je neophodno imati adekvatan plan oporavka.

Jednom napravljen plan oporavka nije dovoljan, već je neophodno konstantno ga inovirati u skladu sa zahtevima korisnika, odnosno aplikacije, ali i drugim spoljnim faktorima. To mogu biti promena

tolerancije kompanije na mogućnost gubitka podataka, dostupnost prostora za rezervne kopije, kao i dozvoljeno vreme nedostupnosti podataka, odnosno vreme potrebno za oporavak. Takođe, neophodno je konstantno testiranje svrsishodnosti plana i uvežbavanje njegovog sprovođenja u uslovima najbližim realnom otkazu sistema.[2]

Radi demonstracije oporavka baze podataka, obrisaćemo tabelu KUPAC i rekonstruisati istu koristeći postojeću rezervnu kopiju. Prethodno smo se uverili da je baza postavljena u *Archive Log Mode*:

```
SQL> archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            USE_DB_RECOVERY_FILE_DEST
Oldest online log sequence     75
Next log sequence to archive   77
Current log sequence           77
```

Kao dobru praksu, prvo smo uradili backup cele baze podataka:

```
RMAN> BACKUP DATABASE PLUS ARCHIVELOG;
```

Brišemo tabelu KUPAC:

```
DROP TABLE KUPAC;
```

A možemo je vratiti u pređašnje stanje komandom FLASHBACK:

```
FLASHBACK TABLE KUPAC TO BEFORE DROP;
```

2. Transparent Data Encrypton - Teorijske osnove

Jedan od glavnih problema u oblasti bezbednosti baza podataka je činjenica da pored administratora i programera, pristup potencijalno osetljivim podacima imaju i osobe koje su odgovorne za kreiranje rezervnih kopija, kao i one koje su prisutne na lokaciji gde se ove kopije čuvaju.[2] Transparentna enkripcija podataka (*Transparent Data Encryption – TDE*) predstavlja napredan metod zaštite osetljivih, odnosno poverljivih podataka i to na takav način da nije neophodna nikakva dodatna izmena u kodu same aplikacije kreirane nad bazom. Uticaj enkripcije na performanse baze je minimalan i iznosi oko 5% gubitka na brzini pristupa, dok je po pitanju veličine fajlova zanemarljiv. [12]

Transparentna enkripcija podataka je sistem kontrole pristupa zasnovan na sistemu ključeva koji se čuvaju na eksternom modulu. Master ključ kojim se posredno šifruju objekti baze ne čuva se u samoj bazi, a njegova upotreba je neprimetna za same korisnike.[3]

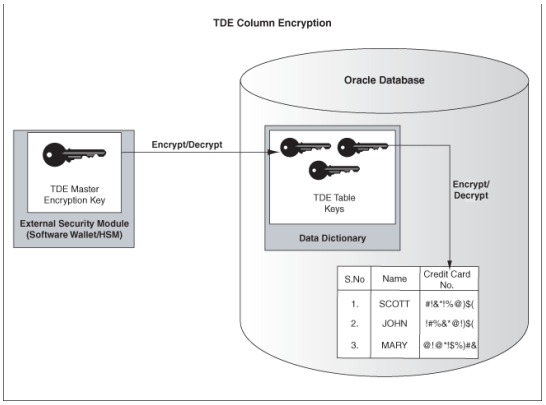
Ova opcija je uvedena u verziji 10gR2 Oracle SUBP-a, a u aktuelnom obliku postoji od verzije 11gR1. Pri implementaciji kriptografskih protokola, vođeno je računa o svim relevantnim standardima i propisima u oblasti informacione bezbednosti, odnosno čuvanja podataka.[13] Rad sa enkripcijom u Oracle sistemu je moguć iz konzole, grafičkog interfejsa i korišćenjem pomoćnih programa koji dolaze uz softverski paket.

Oracle je od verzije 11g uveo brojne značajne novine koje se tiču bezbednosti baze, a tiču se upotrebe jačih algoritama enkripcije lozinke, strožih pravila za izbor lozinke, novih mogućnosti za beleženje potencijalno rizičnih događaja, pokušaja neovlašćenih pristupa itd. Koncept enkripcije na nivou tabelarnog prostora nastoji da reši probleme vezane za tzv. podatke u mirovanju (*data at rest*) istovremeno uzimajući u obzir performanse i izvodljivost. Da bi ovo bilo moguće, došlo je do značajnih izmena u internom formatu beleženja određenih tipova podataka, pre svega tzv. velikih objekata (BLOB i CLOB), kako bi i njihova upotreba bila na jednakom nivou performansi kao i pre uvođenja enkripcije.[2] Osim navedenih, podržana je enkripcija sledećih tipova podataka: BINARY_FLOAT, NUMBER, BINARY_DOUBLE, NVARCHAR2, CHAR, RAW, DATE, TIMESTAMP, NCHAR, VARCHAR2.

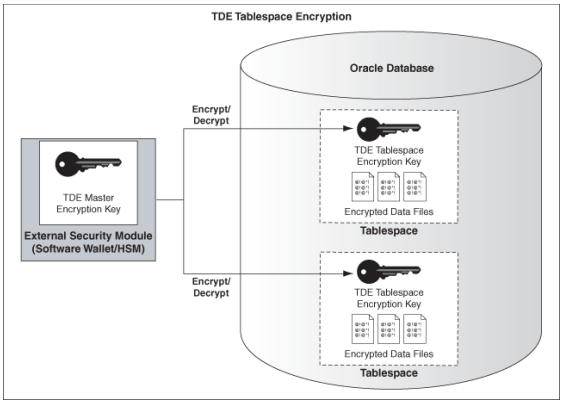
Za samu enkripciju podataka koristi se AES (*Advanced Encryption Standard*) algoritam sa tri moguće dužine ključa: 256, 192, i 128 bitova. Hardversko ubrzanje (akceleracija) kriptografskih

operacija podržana je na pojedinim Intel i Solaris arhitekturama, a odnosi se isključivo na šifrovanje tabelarnih prostora.[13]

Glavni, odnosno master ključ se koristi za enkripciju sekundarnih ključeva kojima se vrši neposredno šifrovanje kolona i tabelarnih prostora. Ključevi za enkripciju kolona se u šifrovanom obliku čuvaju u rečniku podataka, a ključ za tabelarni prostor u zaglavlju (*header*) samog prostora kao i u zaglavljima pojedinačnih fajlova operativnog sistema koji ga čine.



Slika 7 - Pozicija ključa pri šifrovanju kolone



Slika 8 - Pozicija ključa pri šifrovanju tablespace-a

Takođe, šifrovanje je transparentno što znači da nije potrebno rukovati nikakvim dodatnim tabelama ili okidačima (*trigger*) radi pristupa zaštićenim podacima. Prilikom unošenja podatka u neku šifrovanu kolonu, sistem vrši automatsku enkripciju, a prilikom čitanja automatsku dekripciju podataka. Osim na regularne, transparentna enkripcija se može primeniti i na eksterne tabele.[1] Isti mehanizam zasnovan na enkripcijskim ključevima koristi se i prilikom šifrovanja tabelarnog prostora. Sami ključevi se čuvaju u tzv. novčaniku (*wallet*) izvan same baze podataka.[2]

Oracle novčanik (*wallet*) predstavlja svojevrstan kontejner za skladištenje akreditiva (*credentials*) autentičnosti i potpisivanja.[2] U njemu se čuva glavni (master) ključ iz kojeg se izvode ostali ključevi za šifrovanje pojedinih objekata u sistemu. Novčanik je zaseban fajl koji ima naziv *ewallet.p12* a njegovu lokaciju je moguće podesiti u konfiguracionom fajlu *sqlnet.ora* a za rad sa istim je neophodna privilegija ALTER SYSTEM.

Podrazumevani sistemski novčanik se može kreirati isključivo SQL komandama odnosno korišćenjem *Enterprise Manager*-a, dok softverski paket *Oracle Wallet Manager* ne podržava tu opciju. Za većinu situacija implementacija Oracle novčanika u vidu fajla predstavlja prihvatljivo

rešenje, dok kompanije koje imaju strože bezbednosne zahteve, dodatnu zaštitu podataka mogu ostvariti primenom hardverskih rešenja. Uvođenjem hardverskih bezbednosnih modula (*Hardware Security Module - HSM*) izbegava se nužno prisustvo dešifrovanog master ključa u memoriji sistema budući da se sve funkcije vezane za šifrovanje izvršavaju u samom fizički odvojenom modulu. Time se eliminiše veoma značajan vektor napada, budući da su ključevi pohranjeni na bezbednom mestu, a čak ni fizička kompromitacija samog modula ne omogućava napadaču da do njih dođe. Oracle u potpunosti podržava upotrebu HSM modula, a posle instalacije i konfiguracije, njihovo korišćenje se sa strane krajnjeg korisnika ne razlikuje od običnog Oracle novčanika.[2]

U slučajevima kada je neophodno obezbediti da baza može da se restartuje i bez prisustva administratora, postoji mogućnost automatskog otvaranja novčanika. Ovim putem se uklanja inače neophodan korak unošenja lozinke za otvaranje novčanika, pa su ključevi za enkripciju uvek dostupni bazi kroz fajl *cwallet.sso*.[13]

Pravljenje rezervne kopije putem RMAN aplikacije neće sačuvati novčanik u bilo kom obliku zajedno sa kopijama ostalih fajlova, te je potrebno ručno napraviti duplikat istog i čuvati ga na bezbednoj lokaciji, podrazumeva se ne istoj gde su kopije enkriptovanih fajlova.[6]

Mogućnost šifrovanja čitavog tabelarnog prostora eliminiše potrebu za detaljnom analizom aplikacija i samih tabela kako bi se utvrdile one koje sadrže potencijalno osetljive informacije. Šifrovanje se takođe može izvršiti premeštanjem postojećih objekata u novokreirani šifrovani tabelarni prostor. Izuzetak su tabele sa BFILE kolonom i eksterne tabele, s obzirom da se one čuvaju van datog tabelarnog prostora.

Postoji nekoliko ograničenja prilikom šifrovanja tabelarnih prostora. Naime, nije moguće šifrovati već postojeće nego je to moguće samo prilikom njihovog kreiranja. Ukoliko je potrebno šifrovati već postojeće podatke, potrebno je kreirati novi prostor i premestiti ih. Takođe, privremeni (*temporary*) i *undo* tabelarni prostori ne mogu biti šifrovani dok prenosivi (*transportable*) moraju imati isti format upisa na disk (*endian* format).[2]

Kao dodatna mera bezbednosti uvedeno je sigurno brisanje odnosno tzv. seckanje rezervnih kopija (*backup shredding*), pomoću koga se mogu uništavati enkripcijski ključevi šifrovanih rezervnih kopija, čime se one čine potpuno neupotrebljivim i nedostupnim.[2]

Treba imati u vidu da samom enkripcijom određene kolone ne bivaju uništeni prethodni podaci sačuvani u *plaintext* obliku. Oracle sistem premešta podatke u okviru tabelarnog prostora poštujući određenu preprogramiranu logiku koja ne podrazumeva nikakav poseban odnos prema podacima koji su sada šifrovani, a čije prethodne verzije ostaju u samom fajlu dok ne budu prepisane nekim novim podacima.

Da bi se ovo izbeglo, najbolje prakse nalažu da se, čak i kada se enkriptuje samo jedna kolona jedne tabele, uvek kreira novi tabelarni prostor na koji će se tabela naknadno prebaciti. Takođe, kada se prethodni tabelarni prostor obriše (DROP TABLESPACE), preporučljivo je ne koristiti opciju AND DATAFILES, već same fajlove operativnog sistema obrisati koristeći neki od alata za bezbedno uklanjanje tj. seckanje (*shredding*) poverljivih podataka.[6]

2.1. Primena

Na početku rada sa TDE, u fajlu *sqlnet.ora* navešćemo putanju na kojoj će se čuvati sam *wallet* fajl.

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE = (METHOD = FILE)
 (METHOD_DATA =
 (DIRECTORY = c:\app\admin\orcl\wallet\)))
```

Zatim je neophodno kreirati *wallet* fajl u kome će se čuvati master ključ, a koji će biti zaštićen odgovarajućom lozinkom.

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY student1;
```

Posle svakog restartovanja instance, neophodno je otvoriti *wallet* da bi rad sa enkriptovanim tabelarnim prostorom bio moguć. Ovo se postiže komandom:

```
ALTER SYSTEM SET WALLET OPEN IDENTIFIED BY student1;
```

U suprotnom, svaki pokušaj pristupa šifrovanim podacima rezultira greškom:

```
ORA-28365: wallet is not open
```

Lozinku možemo promeniti aplikacijom orapki (Oracle PKI Tool) koja dolazi uz Oracle paket.
Inače, promena lozinke ne podrazumeva promenu master ključa koji se u novčaniku nalazi.

```
orapki wallet change_pwd -wallet c:\app\admin\orcl\wallet\
ewallet.p12 –oldpwd student1 -newpwd student2
```

Detalje novčanika možemo pogledati koristeći isti alat:

```
C:\app\oradata\orcl>orapki wallet display -wallet c:\app\admin\
orcl\wallet\ewallet.p12

Oracle PKI Tool : Version 11.2.0.1.0 - Production
Copyright (c) 2004, 2009, Oracle and/or its affiliates. All
rights reserved.
Requested Certificates:
Subject:        CN=oracle
User Certificates:
Oracle Secret Store entries:
ORACLE.SECURITY.DB.ENCRYPTION.AQqaaEfaHU/Avw+5i+
+3dBgAAAAAAAAAAAAAAAAAAAAAAAAAA
ORACLE.SECURITY.DB.ENCRYPTION.AR0dcnLRa09Rv+sDOgbByXEAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
ORACLE.SECURITY.DB.ENCRYPTION.MASTERKEY
ORACLE.SECURITY.TS.ENCRYPTION.BZNJRjDLNauhStemoL881TwCAwAAAAAA
AAAAAAAAAAAAAAAAAA
Trusted Certificates:
```

A status i putanju iz pogleda V$ENCRYPTION_WALLET:

```
SELECT * FROM V$ENCRYPTION_WALLET;
```

	WRL_TYPE	WRL_PARAMETER	STATUS
1	file	c:\app\admin\orcl\wallet	OPEN

Slika 9 – Pogled V$ENCRYPTION_WALLET

Da bismo novčanik prebacili u režim automatskog otvaranja, neophodno je kreirati *cwallet.sso* fajl koristeći postojeći *ewallet.p12* kako bi master ključ bio sačuvan u adekvatnom formatu. Ovaj fajl je vezan za konkretan sistem na kome je kreiran, te nije moguće importovati ga na drugi server. Ovakav novčanik je šifrovan kombinacijom imena servera i korisničkog imena korisnika koji ga je kreirao, te ga je moguće relativno lako dešifrovati ukoliko su ovi parametri poznati.[7] Takođe, za naknadno otvaranje ovako sačuvanog novčanika nije neophodna lozinka i sistem će to raditi automatski. Pomenuti fajl kreiramo komandom:

```
orapki wallet create -wallet c:\app\admin\orcl\wallet\ -
auto_login
```

A ukoliko je potrebno, naknadno ga možemo zatvoriti, bez navođenja lozinke, putem komande:

```
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
```

U narednom koraku ćemo kreirati novi tabelarni prostor:

```
CREATE SMALLFILE TABLESPACE tbs_3079_enc
DATAFILE 'c:\app\oradata\orcl\tbs_3079_enc.dbf' SIZE 50m
AUTOEXTEND ON NEXT 10m
ENCRYPTION USING 'AES256'
DEFAULT STORAGE (ENCRYPT);
```

I uveriti se da je zaista enkriptovan:

```
SELECT TABLESPACE_NAME, ENCRYPTED FROM DBA_TABLESPACES;
```

	TABLESPACE_NAME	ENCRYPTED
1	SYSTEM	NO
2	SYSAUX	NO
3	UNDOTBS1	NO
4	TEMP	NO
5	USERS	NO
6	EXAMPLE	NO
7	TBS_3079	NO
8	TBS_ZA_INDEXE	NO
9	TBS_3079_ENC	YES

Slika 10 – Pogled DBA_TABLESPACES

Kreiranje tabele na šifrovanom tabelarnom prostoru se ni po čemu ne razlikuje od uobičajenog, uz uslov da je administrator baze pri startovanju iste uneo lozinku za otvaranje novčanika. Tabelu ćemo kao i u prethodnim koracima, popuniti nasumičnim test podacima.

```
CREATE TABLE "KUPAC_ENC"
   ("SIFRAKUPCA" NUMBER(5,0) NOT NULL,
  "IMEKUPCA" VARCHAR2(30 BYTE) NOT NULL,
  "ADRESA" VARCHAR2(50),
  "PIB"    NUMBER(8),
  "BRTEL"  NUMBER(8),
  "EMAIL"  VARCHAR2(30)
   ) tablespace tbs_3079_enc;
```

Da su podaci zaista enkriptovani, možemo se uveriti ukoliko pokušamo direktan pristup fajlovima operativnog sistema koji sadrže tabelarne prostore. Iz običnog fajla vrlo lako možemo pročitati većinu informacija:

```
C:\app\oradata\orcl>strings TBS_3079.DBF
…
Sara Hughes
9546 Butterfield Pass
KNd
```

```
d8Z
shughes33@unc.edu,
Janice Murray
38425 Rockefeller Parkway
jmurray32@weibo.com,
…
```

Dok iz enkriptovanog, to nije moguće:

```
C:\app\oradata\orcl>strings TBS_3079_ENC.DBF
…
VF8\
M>9
ud?
.97
Y,O
4D@?
nLsS
}O_5
…
```

Prilikom enkripcije pojedinačne kolone, neophodno je dodati opciju ENCRYPT prilikom kreiranja tabele.

```
CREATE TABLE "KUPAC_ENC_KOL"
   ("SIFRAKUPCA" NUMBER(5,0) NOT NULL,
  "IMEKUPCA" VARCHAR2(30 BYTE) NOT NULL,
  "KRED_KARTICA" NUMBER(16) ENCRYPT
   ) tablespace tbs_3079;

SELECT * FROM USER_ENCRYPTED_COLUMNS;
```

Primetićemo da je izabran podrazumevani algoritam „AES-192“:

	TABLE_NAME	COLUMN_NAME	ENCRYPTION_ALG	SALT	INTEGRITY_ALG
1	KUPAC_ENC_KOL	KRED_KARTICA	AES 192 bits key	YES	SHA-1

Slika 8 - Lista šifrovanih kolona

Kada pokušamo direktan pristup podacima na nivou fajla operativnog sistema, moći ćemo da pročitamo podatke iz ostalih, ali ne iz šifrovane kolone. Podatak unet u šifrovanu kolonu KRED_KARTICA je broj od 16 cifara, a listanje stringova i filtriranje rezultata po tom kriterijumu komandom grep nije dalo nijedan takav broj.

```
C:\app\oradata\orcl>strings TBS_3079.DBF | grep -x '.\{16,16\}'
```

```
…
Justine Wallace
19883 Duke Hill
BEING_PROCESSED
BEING_PROCESSED
MGMT_TASK_QTABL
X$KZSRO@SEL$236
ultrices posuer
aquam fringilla
…
```

Sam pristup podacima od strane legitimnih korisnika je moguć, kao i u prethodnom slučaju, pod uslovom da je administrator baze pri startovanju iste uneo lozinku za otvaranje novčanika, odnosno ako je novčanik automatski otvoren. Ukoliko to nije slučaj, biće moguće pristupiti samo kolonama date tabele koje nisu šifrovane.

Takođe je moguće šifrovati već postojeću kolonu neke tabele:

```
ALTER TABLE KUPAC_ENC_KOL MODIFY (IMEKUPCA ENCRYPT);
SELECT * FROM USER_ENCRYPTED_COLUMNS;
```

ript Output | Query Result

SQL | All Rows Fetched: 2 in 0.002 seconds

	TABLE_NAME	COLUMN_NAME	ENCRYPTION_ALG	SALT	INTEGRITY_ALG
1	KUPAC_ENC_KOL	IMEKUPCA	AES 256 bits key	YES	SHA-1
2	KUPAC_ENC_KOL	KRED_KARTICA	AES 256 bits key	YES	SHA-1

Slika 9 - Lista šifrovanih kolona nakon izmene

S obzirom da ćemo u demonstraciji premeštanja postojećih tabela raditi sa fajlovima samog operativnog sistema, pre prebacivanja podataka sa jednog *tablespace*-a na drugi, eksplicitno ćemo zahtevati da Oracle izvrši upis na disk podataka iz svih završenih transakcija, a koje su eventualno ostale u *buffer cache* memoriji.

```
ALTER SYSTEM CHECKPOINT;
```

Nakon toga, prebacićemo tabelu KUPAC na šifrovani tabelarni prostor:

```
ALTER TABLE KUPAC MOVE TABLESPACE tbs_3079_enc;
SELECT TABLE_NAME, TABLESPACE_NAME FROM USER_TABLES;
```

Script Output | Query Result

SQL | All Rows Fetched: 8 in 0.054 seconds

	TABLE_NAME	TABLESPACE_NAME
1	CENA	TBS_3079
2	PONUDA	TBS_3079
3	STAVKAPONUDE	TBS_3079
4	ARTIKAL	TBS_3079
5	ARTIKAL_DETALJI	TBS_3079
6	KUPAC_ENC	TBS_3079_ENC
7	KUPAC_ENC_KOL	TBS_3079
8	KUPAC	TBS_3079_ENC

Slika 10 – Lista tabelarnih prostora

Kao što je u uvodnom delu napomenuto, premeštanje tabele sa jednog *tablespace*-a na drugi oslobađa prostor, ali ne briše podatke fizički u samim fajlovima operativnog sistema. To znači da će oni biti prisutni sve dok ne budu prepisani novim podacima. U nedostatku bolje sistemske opcije, kreiraćemo tabelu FILLER_DATA na *tablespace*-u u čijem *datafile*-u želimo da prebrišemo podatke i popuniti je sa pet miliona redova slučajnih brojeva.

```
CREATE TABLE FILLER_DATA (
BROJ NUMBER(10)
) TABLESPACE tbs_3079;

INSERT INTO FILLER_DATA
SELECT DBMS_RANDOM.VALUE(1000000000,9999999999)
FROM (SELECT LEVEL FROM DUAL CONNECT BY LEVEL <= 5000000);
```

Nakon ove operacije, ustanovljenom metodom listanja stringova potvrđujemo da se ne može doći do podataka o kupcima neposrednim uvidom u sistemske fajlove, odnosno da je većina istih prepisana slučajnim brojevima. Naravno, podaci u *tablespace*-u *tbs_za_indexe*, u kome se nalazi indeks na koloni IMEKUPCA ostaju neenkriptovani, a što treba imati u vidu u realnom scenariju. Po završetku rada, obrisaćemo tabelu FILLER_DATA koristeći opciju PURGE jer nam neće biti potreban eventualni oporavak iste.

```
DROP TABLE filler_data PURGE;
```

Podrazumevani Algoritam je AES-192, a kao i sam ključ kojim je kolona šifrovana moguće je izmeniti ga komandom *rekey*:

```
ALTER TABLE VOJA_3079.KUPAC_ENC_KOL REKEY USING 'AES128';
ALTER TABLE VOJA_3079.KUPAC_ENC_KOL REKEY;
SELECT * FROM USER_ENCRYPTED_COLUMNS;
```

	TABLE_NAME	COLUMN_NAME	ENCRYPTION_ALG	SALT	INTEGRITY_ALG
1	KUPAC_ENC_KOL	IMEKUPCA	AES 128 bits key	YES	SHA-1
2	KUPAC_ENC_KOL	KRED_KARTICA	AES 128 bits key	YES	SHA-1

Slika 11 – Detalji šifrovanih kolona

Inače, podrazumevani režim rada sa kriptozaštitom kolona koristi *salt*, odnosno nasumični niz karaktera (string) koji se dodaje ispred *plantext*-a kako bi se sprečili određeni kriptoanalitički napadi (*dictionary, pattern matching, rainbow table*). Ukoliko je potrebno, ova opcija se može isključiti:

```
ALTER TABLE VOJA_3079.KUPAC_ENC_KOL MODIFY
(KRED_KARTICA ENCRYPT NO SALT );
SELECT * FROM USER_ENCRYPTED_COLUMNS;
```

SQL | All Rows Fetched: 2 in 0.003 seconds

	TABLE_NAME	COLUMN_NAME	ENCRYPTION_ALG	SALT	INTEGRITY_ALG
1	KUPAC_ENC_KOL	IMEKUPCA	AES 128 bits key	YES	SHA-1
2	KUPAC_ENC_KOL	KRED_KARTICA	AES 128 bits key	NO	SHA-1

Slika 12 – Isključivanje salt opcije

Izvoz (*export*) podataka putem *datapump* alata je takođe moguće zaštititi putem transparentne enkripcije. Neophodno je da novčanik bude otvoren i da podesimo parametar ENCRYPTION_MODE=TRANSPARENT. Takođe, baza u koju importujemo podatke mora imati isti novčanik odnosno enkripcijski ključ.

```
CREATE DIRECTORY SIFROVANI_DUMP AS 'C:\APP\DPUMP\';

expdp
dumpfile=sifrovani_dump:voja_3079_sifrovani_dump_transparent.dm
p
logfile=sifrovani_dump:voja_3079_sifrovani_dump_transparent.log
schemas=voja_3079 encryption=all encryption_mode=transparent
```

Analizom fajla VOJA_3079_SIFROVANI_DUMP_TRANSPARENT.DMP utvrdili smo da nema podataka u *plaintext*-u.

Konfigurisanje transparentne enkripcije nam omogućava da postojeće enkripcijske ključeve iskoristimo za kreiranje šifrovanog bekapa, koji će na bezbedan način čuvati sve podatke iz baze, ne samo one koji se već nalaze u šifrovanim kolonima ili tabelarnim prostorima. Da bi se naknadno

izvršio oporavak iz ovako kreiranih rezervnih kopija, neophodno je iskoristiti isti novčanik kojim je kreiran. Inače, na ovaj načine se mogu šifrovati samo rezervne kopije kreirane kao RMAN backup set ali ne i RMAN image kopije.[9]

```
SELECT * FROM  V$RMAN_ENCRYPTION_ALGORITHMS;
```

	ALGORITHM_ID	ALGORITHM_NAME	ALGORITHM_DESCRIPTION	IS_DEFAULT	RESTORE_ONLY
1	1	AES128	AES 128-bit key	YES	NO
2	2	AES192	AES 192-bit key	NO	NO
3	3	AES256	AES 256-bit key	NO	NO

Slika 13 – Podržani algoritmi

```
Rman target /
configure encryption for database on;
BACKUP AS BACKUPSET TABLESPACE TBS_3079;
```

Analizom fajla O1_MF_NNNDF_TAG20170327T201625_DFLOQT7S_.BKP utvrdili smo da nema podataka u *plaintext*-u.

U verziji 11g Oracle sistema, uvedena je opcija onemogućavanja korišćenja kreiranih bekap setova uništavanjem enkripcijskog ključa kojim su kreirani - backup shredding. Takođe, prilikom sigurnog brisanja nije neophodan fizički pristup medijima na kojima se podaci nalaze, a jedino ograničenje je da nije moguće na ovakav način onesposobiti backup koji je zaštićen zasebnom šifrom prilikom kreiranja.[2]

Prvo ćemo se uveriti da je enkripcija bekapa uključena:

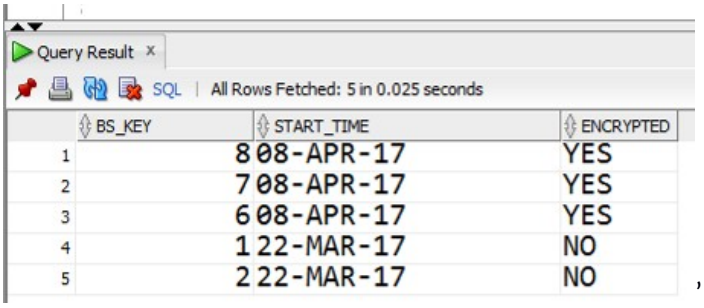
```
RMAN> show all;

RMAN configuration parameters for database with db_unique_name
ORCL are:
...
CONFIGURE ENCRYPTION FOR DATABASE ON;
CONFIGURE ENCRYPTION ALGORITHM 'AES128'; # default
...
```

Ponovićemo komandu:

```
RMAN> BACKUP AS BACKUPSET TABLESPACE tbs_3079;
```

I dobiti fajl O1_MF_NNNDF_TAG20170408T190324_DGL5YWVL_.BKP Pošto ga fizički izmestimo sa podrazumevane putanje, simulirajući premeštanje na udaljenu lokaciju, on će ipak ostati prisutan u tabeli raspoloživih backup setova, ali ga naravno neće biti moguće oporaviti jer je u EXPIRED stanju.

```
select bs_key, start_time, encrypted from v$BACKUP_SET_DETAILS;
```



Slika 17 – Lista backup setova

```
RMAN> CROSSCHECK BACKUP;

using channel ORA_DISK_1
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=C:\APP\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\
2017_03_22\O1_MF_NNNDF_TAG20170322T212210_DF5Q7NKL_.BKP RE
CID=1 STAMP=939331332
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=C:\APP\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\
2017_03_22\O1_MF_NCSNF_TAG20170322T212210_DF5QCNQV_.BKP RE
CID=2 STAMP=939331460
crosschecked backup piece: found to be 'EXPIRED'
backup piece handle=C:\APP\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\
2017_04_08\O1_MF_NNNDF_TAG20170408T190324_DGL5YWVL_.BKP RE
CID=6 STAMP=940791804
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=C:\APP\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\
2017_04_08\O1_MF_NNNDF_TAG20170408T201933_DGLBFOOH_.BKP RE
CID=7 STAMP=940796373
crosschecked backup piece: found to be 'AVAILABLE'
backup piece handle=C:\APP\FLASH_RECOVERY_AREA\ORCL\BACKUPSET\
2017_04_08\O1_MF_NNNDF_TAG20170408T203504_DGLCBRKY_.BKP RE
CID=8 STAMP=940797304
Crosschecked 5 objects
```

Da bi se uz brisanje reference na trenutno nedostupni backup set uspešno izvršilo i brisanje ključa korišćenog pri njegovom kreiranju, pozivamo komandu DELETE sa FORCE opcijom:

```
RMAN> DELETE FORCE NOPROMPT BACKUPSET TAG TAG20170408T190324;
```

Zaključak

U ovom radu prikazali smo neke od osnovnih administratorskih aktivnosti pri kreiranju i održavanju baze podataka. Iako je projektovanje baze dugotrajan, temeljan i dobro promišljen proces, često ažuriranje pa i izmene pojedinih elemenata osnovne strukture sistema su administratorska svakodnevica. Podrazumevani uslovi dostupnosti i funkcionalnosti ne smeju biti narušeni više nego što korporativna politika dopušta, a mogućnost otkaza nekog dela sistema mora biti propraćena postojanjem adekvatnog i ažurnog plana oporavka.

Bezbednost baze podataka je višeslojna kategorija koja se previše često tumači samo sa stanovišta prava pristupa i privilegija postojećih korisnika i aplikacija. Sama materijalna priroda medija na kojima su podaci pohranjeni, te njihova nužna dostupnost pojedinim delovima operativnog sistema, čini mogućim brojne druge vektore napada. Takođe, eventualni pristup rezervnim kopijama podataka od strane neovlašćenih lica je veoma teško predupredi ti poslovnom tj. bezbednosnom politikom kompanije koja sa tim podacima primarno radi.

Transparentna enkripcija podataka na nivou kolone i tabelarnog prostora predstavlja adekvatan odgovor kompanije Oracle na ovaj specifičan problem. Uz minimalan dodatni napor administratora i praktično zanemarljive gubitke performansi, moguće je izbeći većinu pretnji po sigurnost pohranjenih podataka (*data at rest*). Uspešnom kombinacijom prihvaćenih i proverenih kriptografskih rešenja sa intuitivnim načinom implementacije i korišćenja, izbegnuta je uobičajena dilema između bezbednosti i komfora u radu, koja prati mnoge druge softverske pakete.

Reference

[1] Alapati, S. R., *Expert Oracle Database 11g Administration*, Apress, 2012.

[2] Alapati, S. R., Kim, C., *Oracle Database 11g: od početnika do profesionalca*, Kompjuter biblioteka, 2009.

[3] Bryla B., Loney K., *Oracle Database 11g DBA Handbook*, McGraw-Hill, 2008.

[4] Connolly, T. M., Begg, C. E., *Database Solutions, A step-by-step guide to building databases*, Pearson Education Limited, 2004.

[5] http://docs.oracle.com/cd/E11882_01/network.112/e40393/asotrans.htm, April, 2017.

[6] http://www.oracle.com/technetwork/database/security/tde-faq-093689.html, April, 2017.

[7] https://github.com/tejado/ssoDecrypt/, April, 2017.

[8] Kuhn, D., Alapati S., Padfield, B., *Expert Indexing in Oracle Database 11g*, Apress, 2012.

[9] Kuhn, D., *RMAN Recipes for Oracle Database 12c*, Apress, 2013.

[10] Lazarević, B., Marjanović, Z., Aničić, N., Babarogić, S., *Baze Podataka*, FON Beograd, 2016.

[11] Mullins, C. S., *Database Administration: The Complete Guide to Practices and Procedures*, Addison Wesley, 2002.

[12] Neagu, A., *Oracle 11g Anti-hacker's Cookbook*, Packt Publishing, 2012.

[13] Oracle White Paper, *Transparent Data Encryption Best Practices*, Oracle, 2012.

[14] Sam Lightstone S., Teorey T., Nadeau T., *Physical Database Design*, Elsevier, 2007.

[15] Vels, E., *Oracle DBA: svakodnevni rad*, Kompjuter biblioteka, 2007.