Oracle 9*i*. Podręcznik administratora baz danych

Autorzy: Kevin Looney, Marlene Theriault
Tłumaczenie: Bartłomiej Garbacz, Sławomir Dzieniszewski
ISBN: 83-7361-063-4
Tytuł oryginału: Oracle9i DBA Handbook
Format: B5, stron: 1016

Jak sprawić, aby najważniejsze firmowe systemy internetowe i e-biznesowe były wszechstronne, bezpieczne i łatwo dostępne? Wydana przez Helion, a pierwotnie przez wydawnictwo OraclePress książka „Oracle9i. Podręcznik administratora baz danych" odpowiada na te pytania, wyjaśniając, jak przygotować i obsługiwać rozbudowaną i intensywnie wykorzystywaną bazę danych oraz jak wykorzystać w pełni nowe narzędzia i możliwości, których dostarcza Oracle9i.

Kevin Looney i Marie Thieriault, dwoje znakomitych ekspertów w dziedzinie Oracle, opisują tutaj podstawy działania systemu i dostarczają licznych, zaczerpniętych z życia przykładów oraz prezentują wiele użytecznych technik ułatwiających obsługę systemu Oracle. Książką ta jest niezbędną pozycją w bibliotece każdego administratora baz danych Oracle.

Wewnątrz między innymi:

- Tworzenie i konfigurowanie bazy danych z wykorzystaniem narzędzia Database Configuration Assistant systemu Oracle9i
- Monitorowanie i strojenie pamięci, wykorzystania plików, transakcji oraz zapytań
- Implementowanie w systemie segmentów wycofania lub automatycznego zarządzania wycofywanymi danymi wprowadzonego w Oracle9i
- Sposoby przenoszenia aplikacji oraz zmieniania otwartych tabel bazy danych
- Diagnozowanie i optymalizacja działania systemu z pomocą pakietu STATSPACK
- Implementowanie jak najlepszych procedur bezpieczeństwa i obserwacja bazy danych
- Automatyzacja procedur tworzenia rezerwowych kopii korzystając z programu RMAN
- Wykorzystanie partycjonowania do radzenia sobie z wielkimi bazami danych
- Rozdzielanie zadań i danych pomiędzy różne serwery działające w sieci korzystając z Oracle Net
- Korzystanie z serwera aplikacji Oracle9iAS, który pozwalają na poprawienie wszechstronności i dostępności bazy danych oraz na łatwiejsze jej rozbudowę

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 10.

Zabezpieczenie i monitorowanie bazy danych

Celem tworzenia i wymuszania procedur zabezpieczenia jest ochrona jednego z najcenniejszych zasobów firmy — danych. Składowanie danych w bazie danych czyni je bardziej użytecznymi i dostępnymi dla całej firmy ale również zwiększa prawdopodobieństwo uzyskiwania do nich nieautoryzowanego dostępu. Takie próby dostępu muszą być wykrywane i należy im zapobiegać.

Baza danych Oracle posiada kilka poziomów zabezpieczeń i zapewnia możliwość monitorowania każdego z nich. W niniejszym rozdziale podano opis wszystkich poziomów zabezpieczeń oraz omówiono proces ich obserwacji. Przedstawiono również metody ustawiania niemożliwych do wykrycia haseł oraz wymuszanie na hasłach unieważnień.

Możliwości zabezpieczenia

Baza danych Oracle udostępnia administratorowi bazy danych kilka poziomów zabezpieczeń:

- zabezpieczenie konta w celu kontroli działania użytkowników;
- zabezpieczenie konta dla obiektów bazy danych;
- zabezpieczenie na poziomie systemu w celu zarządzania uprawnieniami globalnymi.

Każda z tych możliwości zostanie omówiona w następnych podrozdziałach. Podrozdział *Wdrażanie zabezpieczeń* zawiera szczegółowe informacje dotyczące efektywnego wykorzystania dostępnych opcji.

Zabezpieczenie konta

Aby uzyskać dostęp do bazy danych Oracle, konieczne jest uzyskanie dostępu do konta w bazie danych. Ten dostęp może być albo bezpośredni — przez połączenia użytkownika z bazą danych, albo pośredni. Do połączeń pośrednich zalicza się uzyskiwanie dostępu przez wstępnie ustawione autoryzacje wewnątrz powiązań między bazami danych. Każde konto musi posiadać swoje hasło. Konto bazy danych może być związane z kontem systemu operacyjnego.

Hasło jest ustawiane przez użytkownika podczas tworzenia konta użytkownika i może być zmienione po utworzeniu konta. Możliwość zmiany hasła konta przez użytkownika jest ograniczona właściwościami programów narzędziowych, do których użytkownikowi przyznano dostęp. Baza danych przechowuje zaszyfrowaną wersję hasła w tabeli słownika danych. Jeżeli konto jest bezpośrednio związane z kontem systemu operacyjnego, istnieje możliwość pominięcia procesu sprawdzania hasła i zamiast tego poleganie na systemowej metodzie identyfikacji użytkowników.

Od wersji Oracle8 hasło może być ponownie wykorzystane (za pomocą ustawienia historii hasła bazy danych). Można również użyć profili do wymuszenia standardowych parametrów hasła (takich jak minimalna długość). Można również automatycznie blokować hasła, jeżeli występuje wiele kolejnych błędów podczas łączenia się z kontem.

Uprawnienia obiektowe

Dostęp do poszczególnych obiektów bazy danych jest uaktywniany przez *uprawnienia* (ang. *privileges*). Uprawnienia są nadawane za pomocą polecenia **grant** i umożliwiają wykonywanie określonych poleceń w odniesieniu do określonych obiektów bazy danych. Na przykład, jeżeli użytkownik `THUMPER` jest właścicielem tabeli `EMPLOYEE` i wykonuje polecenie

```
grant select on EMPLOYEE to PUBLIC;
```

wtedy wszyscy użytkownicy (`PUBLIC`) będą mogli wybierać rekordy z tabeli `EMPLOYEE` użytkownika `THUMPER`. W celu uproszczenia zarządzania uprawnieniami można tworzyć role, które są nazwanymi grupami uprawnień. W przypadku aplikacji, z których korzysta wielu użytkowników, zastosowanie mechanizmu ról w dużym stopniu zmniejsza liczbę nadawanych uprawnień. Role mogą być zabezpieczone hasłem, mogą być dynamicznie aktywowane i dezaktywowane, co umożliwia uzyskanie dodatkowej warstwy zabezpieczenia.

Uprawnienia i role systemowe

Istnieje możliwość wykorzystania roli do zarządzania dostępnymi dla użytkowników poleceniami systemowymi. Do tych poleceń zalicza się: **create table** oraz **alter index**. Operacje dotyczące każdego typu obiektu bazy danych są autoryzowane za pomocą osobnych uprawnień. Na przykład użytkownikowi można przyznać prawo `CREATE TABLE` ale nie prawo `CREATE TYPE`. Tworzone role systemowe można dostosowywać do wymagań użytkowników, co pozwala na przyznawanie dokładnie takich uprawnień, jakich użytkownicy ci potrzebują. Unika się w ten sposób przyznawania nadmiernych uprawnień

w bazie danych. Jak wspomniano w rozdziale 5., role CONNECT oraz RESOURCE zapewniają podstawowe uprawnienia systemowe wymagane, odpowiednio, przez końcowych użytkowników i programistów.

Użytkownicy o przypisanej roli RESOURCE posiadają tym samym prawo systemowe UNLIMITED TABLESPACE, umożliwiające im tworzenie obiektów w dowolnym miejscu bazy danych. Z powodu tego dodatkowego prawa należy ograniczać przyznawanie roli RESOURCE do środowisk związanych z programowaniem oraz testowaniem aplikacji.

Wdrażanie zabezpieczeń

W bazie danych Oracle można zastosować następujące mechanizmy zabezpieczające: role, profile oraz bezpośrednie przyznawanie uprawnień. Pakiet *Oracle Enterprise Manager* udostępnia program narzędziowy *Security Manager* (*Menedżer zabezpieczeń*), co umożliwia zarządzanie kontami użytkowników, rolami, uprawnieniami i profilami. W następnych podrozdziałach omówiono sposoby wykorzystywania wszystkich tych możliwości. Uwzględniono także kilka nieudokumentowanych opcji dodatkowych.

Punkt wyjścia: zabezpieczenie systemu operacyjnego

Uzyskanie dostępu do bazy danych jest możliwe dopiero po uzyskaniu pośredniego lub bezpośredniego dostępu do serwera, na którym jest uruchomiona baza danych. Pierwszym krokiem zabezpieczania bazy danych jest zabezpieczenie platformy i sieci, w której rezyduje ta platforma. Po dokonaniu tej czynności należy rozważyć sposób zabezpieczenia systemu operacyjnego.

Pewne pliki bazy danych Oracle nie muszą być bezpośrednio udostępniane jej użytkownikom. Na przykład, pliki danych oraz pliki czynnego dziennika powtórzeń są zapisywane i odczytywane tylko przez procesy drugoplanowe Oracle. Zatem jedynie administratorzy baz danych, którzy tworzą i usuwają te pliki, wymagają do nich bezpośredniego dostępu na poziomie systemu operacyjnego. Należy pamiętać o zabezpieczeniu tych plików, jak również o zabezpieczeniu wynikowych plików eksportu i innych plików kopii zapasowych.

Dane mogą być kopiowane do innych baz danych — albo jako część schematu replikacji, albo podczas zapełniania projektowanej bazy danych. Zatem zabezpieczenie danych jest możliwe pod warunkiem zabezpieczenia każdej bazy danych, w której rezydują dane. Należy także pamiętać o kopiach zapasowych każdej z baz danych. Jeżeli istnieje możliwość uzyskania dostępu osób niepożądanych do taśm archiwizowania zawierających kopie danych, cały wdrożony w bazie danych system zabezpieczeń staje się bezwartościowy. Nie wolno dopuszczać do możliwości zaistnienia nieuprawnionego dostępu do którejkolwiek kopii chronionych danych.

Tworzenie użytkowników

Celem tworzenia użytkowników jest zakładanie bezpiecznych, użytecznych kont o adekwatnych uprawnieniach i odpowiednich ustawieniach domyślnych. Do tworzenia nowych kont bazy danych można używać polecenia `create user`. Bezpośrednio po utworzeniu konto nie posiada żadnych możliwości i jego użytkownik nie jest się w stanie nawet zalogować.

Następuje to dopiero po przyznaniu uprawnień. Wszystkie konieczne ustawienia tworzonego konta użytkownika mogą być określone za pomocą pojedynczego polecenia **create user**. Do tych ustawień zaliczają się wartości wszystkich parametrów wykazanych w tabeli 10.1.

Tabela 10.1. *Parametry polecenia create user*

Parametr	Zastosowanie
username	Nazwa schematu.
password	Hasło dla konta. Może być bezpośrednio związane z nazwą konta systemu operacyjnego hosta lub identyfikowane przez sieciową usługę identyfikacyjną. Dla identyfikacji bazującej na hoście należy użyć klauzuli `identified externally`. Dla identyfikacji bazującej na sieci należy użyć klauzuli `identified globally as`.
default tablespace	Domyślna przestrzeń tabel przeznaczona do składowania obiektów utworzonych w danym schemacie. To ustawienie nie daje użytkownikowi praw do tworzenia obiektów, tylko ustawia wartość domyślną parametru wykorzystywanego w razie przyznania użytkownikowi takich uprawnień.
temporary tablespace	Przestrzeń tabel przeznaczona do składowania tymczasowych segmentów wykorzystywanych podczas operacji sortowania w ramach transakcji.
quota [on tablespace]	Umożliwia użytkownikowi składowanie obiektów w określonej przestrzeni tabel aż do całkowitego rozmiaru określonego jako limit.
profile	Przydziela użytkownikowi profil. Jeżeli nie określono żadnego profilu, wtedy jest stosowany profil domyślny. Profile umożliwiają ograniczenie wykorzystania zasobów systemowych oraz wymuszają stosowanie reguł zarządzania hasłem.
password expire	Unieważnia hasło.
account	Ustawia konto jako zablokowane bądź odblokowane.
default role[s]	Ustawia domyślne role, które mają być uaktywnione dla użytkownika.

Nie można ustawić domyślnych ról podczas tworzenia użytkownika:

```
create user FRED identified by FRED default role ABC;
ORA-01954: DEAFAULT ROLE clause not valid for CREATE USER
```

Poniżej przedstawiono przykładowe polecenie **create user**. Tworzony jest tu użytkownik `THUMPER` z hasłem `R3BB#T`, z domyślną przestrzenią tabel `USERS`, z tymczasową przestrzenią tabel `TEMP` bez limitów wykorzystania przestrzeni oraz z profilem domyślnym.

```
create user THUMPER
identified by R3BB#T
default tablespace USERS
temporary tablespace TEMP;
```

Nie określono tu żadnego profilu, zatem zastosowany będzie profil domyślny bazy danych. Jest to rzeczywisty profil o nazwie `DEFAULT`. Jego początkowe ustawienia dla wszystkich limitów korzystania z zasobów są ustawione na `UNLIMITED` (*nieograniczony*). Szczegółowe informacje dotyczące profili znajdują się w podrozdziale *Profile użytkownika* w dalszej części tego rozdziału.

Ze względu na to, że nie określono żadnych limitów, użytkownik nie może tworzyć obiektów w bazie danych. Przyznanie limitu zasobów przeprowadza się za pomocą parametru **quota** polecenia **create user** lub polecenia **alter user**, jak pokazano poniżej. W tym przykładzie użytkownikowi THUMPER przyznano limit 100 MB w przestrzeni tabel USERS.

```
alter user THUMPER
quota 100M on USERS;
```

Po wykonaniu tego polecenia użytkownik THUMPER może tworzyć segmenty o łącznej wielkości do 100 MB w przestrzeni tabel USERS.

> Użytkownicy nie potrzebują limitów przestrzeni w przestrzeni tabel TEMP, aby dla swoich zapytań tworzyć tymczasowe segmenty w tej przestrzeni tabel.

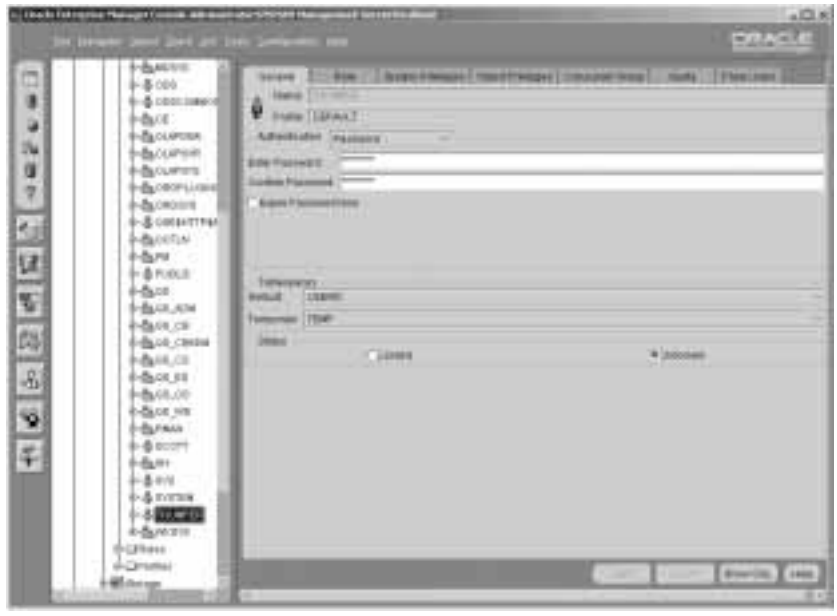
Z wyjątkiem parametru username (*nazwa użytkownika*) wszystkie parametry w poleceniu **create user** mogą być zmienione za pomocą polecenia **alter user**.

Program *Security Manager* z pakietu *OEM* umożliwia utworzenie nowego użytkownika lub utworzenie użytkownika o tych samych atrybutach, jakie posiada już istniejący użytkownik. Na rysunku 10.1 przedstawiono wygląd początkowego okna interfejsu programu *Security Manager*, gdzie zaznaczono nazwę użytkownika THUMPER. Podczas tworzenia użytkownika za pomocą programu narzędziowego *OEM* można przydzielać role, uprawnienia systemowe, uprawnienia obiektowe oraz limity. Początkowe okno programu *Security Manager*, pokazane na rysunku 10.1, służy do uaktywniania hasła identyfikującego użytkownika. Dzięki narzędziom *OEM* można określić role i uprawnienia systemowe a także rozmiar przestrzeni przysługującej użytkownikowi. Za pomocą odpowiedniego oznaczenia hasła, co pokazano na rysunku 10.1, można określić rodzaj danego konta użytkownika. Dane konto może być globalne — służące do zarządzania odległą bazą danych. Okno to umożliwia również określenie, że dane konto ma być identyfikowane zewnętrznie (na poziomie systemu operacyjnego). Dostępna jest także opcja umożliwiająca wstępne unieważnienie hasła, dzięki czemu dane konto może być utworzone jako zablokowane lub niezablokowane. Szczegółowe informacje na temat unieważnienia hasła oraz blokowania konta znajdują się w podrozdziale *Zarządzanie hasłem* w dalszej części niniejszego rozdziału.

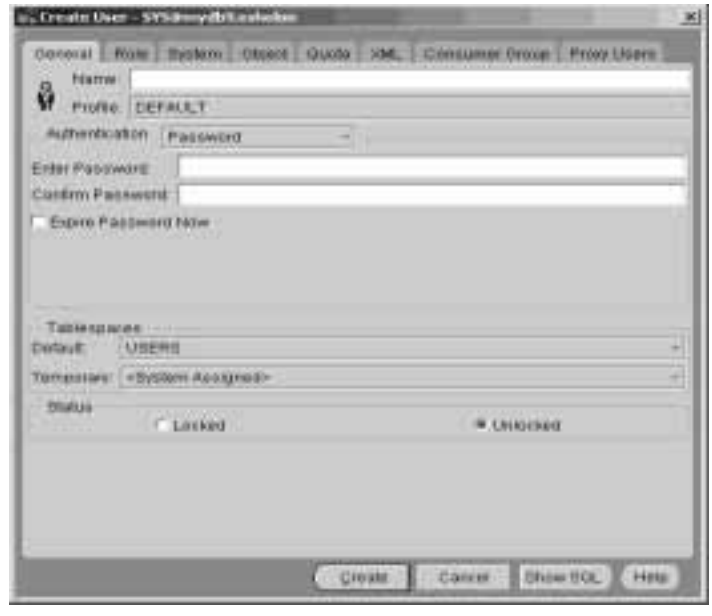
Aby utworzyć nowego użytkownika za pomocą programu *OEM*, należy zaznaczyć w obszarze *Users* (*Użytkownicy*) przycisk *General* (*Ogólne*) i nacisnąć prawy przycisk myszy lub przycisk *Create* (*Utwórz*) z menu *Object* (*Obiekt*). Po wybraniu opcji *Create* (*Utwórz*) lub *Create Like* (*Utwórz podobny do*) uaktywnia się kreator *User Creation Wizard* (*Kreator użytkownika*), co pozwala na szczegółowe określenie atrybutów tworzonego użytkownika (nadanie ról, uprawnień itd.). Domyślnie nowemu użytkownikowi nie jest przydzielana żadna rola. Aby umożliwić mu łączenie się z bazą danych, należy przyznać mu prawo CREATE_SESSION. W ten sposób utworzony użytkownik będzie w stanie połączyć się z bazą danych. Jeśli została określona domyślna wielkość przestrzeni tabel, zostanie ona przyznana nowemu użytkownikowi.

Dla zwykłego użytkownika tymczasowa przestrzeń tabel jest ustawiana jako <System Assigned> (*określona przez system*) ale rozwijana lista pozwala na wybranie odpowiedniej tymczasowej przestrzeni tabel. Na rysunku 10.2 przedstawiono wygląd okna *Create User*

Rysunek 10.1. *Identyfikacja użytkownika za pomocą hasła*



Rysunek 10.2. *Okno Create User, zakładka General*



z oznaczonymi opcjami umożliwiającymi utworzenie nowego użytkownika o właściwościach podobnych do właściwości użytkownika THUMPER, wykorzystywanego w przykładach prezentowanych w tym rozdziale. Domyślną przestrzenią tabel użytkownika THUMPER jest USERS a jego tymczasową przestrzenią tabel jest TEMP. Z powyższego wynika, że nowemu użytkownikowi domyślnie przyznano wszystkie przydziały i uprawnienia, które posiada użytkownik THUMPER. Jedynymi informacjami, które trzeba wprowadzić w celu

utworzenia nowego użytkownika, są nazwa tego użytkownika i hasło. Istnieje również możliwość wprowadzania innych informacji, które będą różniły nowo utworzonego użytkownika od użytkownika THUMPER.

Usuwanie użytkowników

Usuwanie użytkownika z bazy danych przeprowadza się za pomocą polecenia **drop user**. Polecenie **drop user** posiada jeden parametr **cascade**, którego zastosowanie powoduje usunięcie wszystkich obiektów w schemacie użytkownika przed usunięciem tego użytkownika. Jeżeli użytkownik posiada obiekty, konieczne jest określenie parametru **cascade** w celu usunięcia użytkownika. Przykładowe polecenie **drop user** pokazano niżej:

```
drop user THUMPER cascade;
```

Wszystkie perspektywy, synonimy, procedury, funkcje lub pakiety odwołujące się do obiektów w schemacie usuniętego użytkownika są oznaczane jako INVALID. Jeśli później zostanie dodany użytkownik o identycznej nazwie, nie uzyska żadnych związanych z poprzednikiem. Struktury te pozostaną niedostępne nawet w przypadku późniejszego utworzenia innego użytkownika o tej samej nazwie. Program narzędziowy *Security Manager* pakietu *OEM* umożliwia usuwanie użytkowników. Przed ostatecznym usunięciem użytkownika program ten wyświetla okno z żądaniem potwierdzenia.

Uprawnienia systemowe

Role systemowe mogą służyć do ustalania, kto ma prawo do wykonywania poleceń systemowych, służących do zarządzania bazą danych. Można utworzyć odpowiadające rzeczywistym potrzebom role systemowe, albo też zastosować gotowe role, dostarczone wraz z bazą danych. Spis dostępnych uprawnień, które mogą być przyznane przez role systemowe, podano w dodatku A, w opisie polecenia GRANT.

Klauzula **with grant option** polecenia **grant** umożliwia przekazywanie uprawnień innym użytkownikom.

W tabeli 10.2 wyszczególniono 15 ról systemowych, dostarczanych wraz z bazą danych Oracle. Zastosowanie tych ról umożliwia ograniczenie uprawnień systemowych przyznawanym rolom zarządzania bazy. Oprócz ról pokazanych w tabeli 10.2 baza danych może uwzględniać role generowane przez opcję *Advanced Queuing Option* (AQ_USER_ROLE, GLOBAL_AQ__USER_ROLE oraz AQ_ADMINISTRATOR_ROLE), skrypty Java (JAVAUSERPRIV, JAVAIDPRIV, JAVADEBUGPRIV, JAVA_ADMIN i JAVA_DEPLOY), *Oracle Context Management* (CTXAPP) oraz role generowane za pomocą modułu *Intelligent Agents* pakietu *OEM* (rola OEM_MANAGER).

> Oprócz uprawnień wykazanych w tabeli 10.2 użytkownicy ról DBA oraz RESOURCE otrzymują również prawo UNLIMITED TABLESPACE.

Rola CONNECT jest zwykle przyznawana użytkownikom końcowym. Rola ta daje pewne możliwości tworzenia obiektów (włącznie z prawem CREATE TABLE), jednak nie nadaje użytkownikowi żadnego limitu (ang. *quota*) dla przestrzeni tabel. Zatem przed przydzieleniem limitów przestrzeni tabel użytkownicy roli CONNECT nie mogą tworzyć tabel.

Tabela 10.2. *Role systemowe dostarczone w Oracle9i*

Nazwa roli	Uprawnienia przyznane roli
CONNECT	ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SYNONYM, CREATE TABLE, CREATE VIEW
RESOURCE	CREATE CLUSTER, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER
DBA	**Wszystkie uprawnienia systemowe** WITH ADMIN OPTION
EXP_FULL_DATABASE	SELECT ANY TABLE, BACKUP ANY TABLE, **instrukcje** INSERT, DELETE **oraz** UPDATE **na tabelach** SYS.INCVID, SYS.INCFIL **oraz** SYS.INCEXP
WM_ADMIN_ROLE	**Wszystkie uprawnienia narzędzia** *Worspace Manger* **wraz z opcją** GRANT OPTION
IMP_FULL_DATABASE	BECOME USER
DELETE_CATALOG_ROLE	**prawo** DELETE **na wszystkich pakietach słownika**
EXECUTE_CATALOG_ROLE	**prawo** EXECUTE **na wszystkich pakietach słownika**
SELECT_CATALOG_ROLE	**prawo** SELECT **na wszystkich tabelach i perspektywach katalogowych**
CREATE_TYPE	CREATE TYPE, EXECUTE, EXECUTE ANY TYPE, ADMIN OPTION, GRANT OPTION
RECOVERY_CATALOG_OWNER	DROP ROLE, CREATE ROLE, CREATE TRIGGER, CREATE PROCEDURE
OLAP_DBA	ALTER ANY DIMENSION, ALTER ANY TABLE, ANALYZE ANY, CREATE ANY DIMENSION, CREATE ANY INDEX, CREATE ANY TABLE, CREATE ANY VIEW, DROP ANY DIMENSION, DROP ANY TABLE, DROP ANY VIEW, LOCK ANY TABLE, SELECT ANY DICTIONARY, SELECT ANY TABLE
HS_ADMIN_ROLE	HS_EXTERNAL_OBJECT, HS_EXTERNAL_USER
WKADMIN	CREATE ANY DICTIONARY, CREATE CLUSTER, CREATE PROCEDURE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE, DROP ANY DIRECTORY
WKUSER	CREATE ANY DICTIONARY, CREATE CLUSTER, CREATE PROCEDURE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE, DROP ANY DIRECTORY

Rola RESOURCE jest przyznawana programistom. Zgodnie z informacjami przedstawionymi w rozdziale 5., rola RESOURCE daje użytkownikom najczęściej stosowane uprawnienia potrzebne do programowania aplikacji. Rola DBA posiada wszystkie 124 uprawnienia dostępne na poziomie systemu z opcją nadawania tych uprawnień innym użytkownikom (GRANT OPTION).

Firma Oracle zaleca tworzenie własnych ról a nie poleganie na trzech opisanych powyżej. Role DBA, CONNECT czy RESOURCE mogą zostać zarzucone w przyszłych wersjach.

Role IMP_FULL_DATABASE oraz EXP_FULL_DATABASE są stosowane, odpowiednio, podczas importowania i eksportowania danych z bazy (patrz rozdział 11.). Te role stanowią część roli DBA. Role te mogą także służyć do przyznania użytkownikom ograniczonych uprawnień zarządzania bazą danych.

Role SELECT_CATALOG_ROLE, EXECUTE_CATALOG_ROLE oraz DELETE_CATALOG_ROLE zostały wprowadzone w wersji Oracle8.

Role SELECT_CATALOG_ROLE oraz EXECUTE_CATALOG_ROLE przyznają użytkownikom uprawnienia do wybierania lub wykonywania eksportowalnych obiektów słownika danych. Warto tu wspomnieć, że nie każdy obiekt bazy danych jest eksportowany podczas pełnego eksportu systemowego. Dokładniejsze informacje na ten temat znajdują się w rozdziale 11. Na przykład, dynamiczne perspektywy wydajności systemu (patrz rozdział 6.) nie są eksportowane. Zatem rola SELECT_CATALOG_ROLE nie daje użytkownikowi możliwości wybierania danych z dynamicznych tabel wydajności V$ROLLSTAT, ale daje mu możliwość wykonywania zapytań na większości danych ze słownika danych. Podobnie rola EXECUTE_CATALOG_ROLE daje użytkownikom możliwość wykonywania procedur i funkcji, które są częścią słownika danych.

Prawo CREATE TYPE jest uaktywniane, jeżeli jest stosowana opcja *Option*. Użytkownicy, którzy mają uaktywnione prawo CREATE TYPE, mogą tworzyć nowe, abstrakcyjne typy danych.

Po udostępnieniu ról i uprawnień systemowych można ponownie sprawdzić proces tworzenia konta. Podobnie jak w przypadku procesu tworzenia kopii zapasowej bazy danych, do utworzenia konta należy posiadać uprawnienia na poziomie *DBA*. Jednak można także określić inne uprawnienia, które umożliwiają tworzenie nowych użytkowników.

Przykładowo, można utworzyć nową rolę systemową o nazwie ACCOUNT_CREATOR. Rola ta umożliwiałaby tylko tworzenie użytkowników bez możliwości wykonywania innych poleceń dostępnych administratorowi. Poniżej przedstawiono przykładowe polecenia, które utworzą taką rolę.

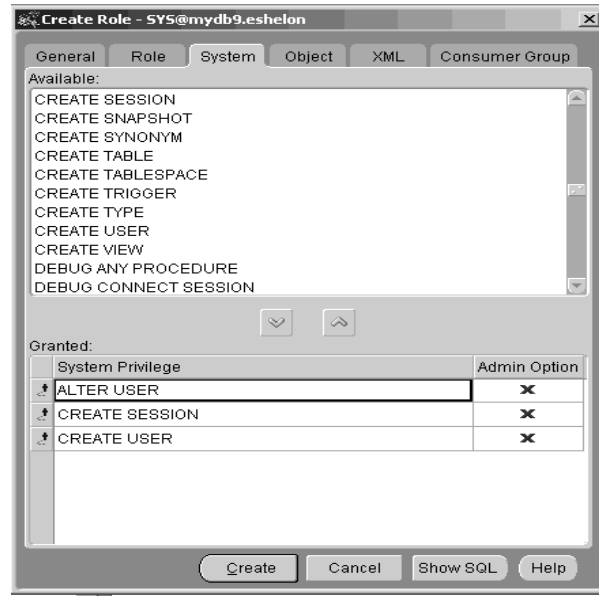
```
create role ACCOUNT_CREATOR;
grant CREATE SESSION, CREATE USER, ALTER USER
   to ACCOUNT_CREATOR;
```

Pierwsze polecenie z powyższego przykładu tworzy rolę o nazwie ACCOUNT_CREATOR, natomiast drugie przyznaje tej roli możliwość zalogowania (CREATE SESSION) oraz tworzenia i zmiany kont (CREATE USER oraz ALTER USER). Przykładowo, rola ACCOUNT_CREATOR może być wykorzystywana przez centralne biuro pomocy, którego zadaniem byłoby koordynowanie tworzenia wszystkich nowych kont w danej aplikacji. Rolę tę można utworzyć za pomocą pakietu *OEM* przez wybranie opcji *Create Role* (*Utwórz rolę*) i wprowadzenie odpowiednich informacji. Na rysunku 10.3 przedstawiono sposób tworzenia roli ACCOUNT_CREATOR za pomocą programu narzędziowego *Security Manager* pakietu *OEM*. Rysunek 10.4 przedstawia sposób przypisania uprawnień do tej roli.

Centralne tworzenie kont jest pomocne w zapewnianiu odpowiednich procedur autoryzacji żądań dostępu do poszczególnych kont. Elastyczność przyznawania uprawnień i ról systemowych umożliwia przydzielenie użytkownikowi (kontynuując przykład, mogłoby to być centralne biuro pomocy) uprawnień pozwalających na tworzenie kont bez zapewnienia temu użytkownikowi możliwości wykonywania zapytań do bazy danych.

Możliwość tworzenia roli ACCOUNT_CREATOR jest szczególnie użyteczna podczas wdrażania oprogramowania pakietowego. Liczni, niezależni od siebie producenci aplikacji pakietowych założyli, że użytkownicy tych aplikacji będą posiadali pełne uprawnienia administratora bazy danych, kiedy faktycznie są potrzebne jedynie możliwości wykonywania poleceń **create user** oraz **alter user**. Utworzenie roli ACCOUNT_CREATOR pozwoli na ograniczenie schematu pakietu uprawnień właściciela w pozostałej części bazy danych.

Rysunek 10.3. *Tworzenie roli ACCOUNT_CREATOR*



Rysunek 10.4. *Przypisanie uprawnień systemowych roli ACCOUNT_CREATOR*

Role określone jako domyślne są uaktywniane po każdym zalogowaniu. Za pomocą klauzuli **default role** polecenia **alter user** można zmieniać domyślną rolę użytkownika. Można również określić, że dany użytkownik nie posiada żadnych ról uaktywnianych domyślnie.

```
alter user THUMPER default role NONE;
```

Można określić role do uaktywnienia.

```
alter user THUMPER default role CONNECT;
```

Można również określić role, które nie powinny być uaktywniane po rozpoczęciu sesji.

```
alter user THUMPER default role all except ACCOUNT_CREATOR;
```

Określenie danej roli jako domyślnej za pomocą polecenia **alter user** nie powiedzie się, jeżeli rola ta nie została już wcześniej przyznana użytkownikowi. Przykładowo, jeżeli dany użytkownik nie posiada roli CONNECT na poziomie systemu, wtedy próba ustawienia dla użytkownika tej roli jako domyślnej zakończy się wystąpieniem następującego komunikatu o błędzie:

```
ORA-01919: role 'CONNECT' does not exists
```

Jeżeli określona rola jest specyficzną rolą bazy danych, która nie została przyznana użytkownikowi, polecenie **alter user** nie powiedzie się i wystąpi następujący komunikat o błędzie:

```
ORA-01955: DEFAULT ROLE 'ACCOUNT_CREATOR' not granted to user
```

Zatem przed ustalaniem domyślnych ról użytkowników jest konieczne przyznanie tych ról. W przypadku zastosowania klauzuli **default role all** wszystkie role użytkownika są uaktywniane po rozpoczęciu sesji użytkownika. Jeżeli planowane jest dynamiczne aktywowanie i dezaktywowanie ról w różnych częściach aplikacji (za pomocą poleceń **set role**), wtedy należy kontrolować, które role są uaktywniane domyślnie.

Parametr MAX_ENABLED_ROLES pliku *init.ora* ogranicza liczbę ról, jakie każdy użytkownik może jednocześnie posiadać jako aktywne. Dla systemu Oracle9i wartością domyślną jest 20.

Przy tworzeniu roli jest ona uaktywniana domyślnie. Jeżeli jest tworzonych wiele ról, wtedy można przekroczyć ustawienie MAX_ENABLED_ROLES, nawet jeżeli nie jest się użytkownikiem tych ról.

Profile użytkownika

Profile użytkownika mogą służyć do określania limitów ilości zasobów systemu i bazy danych dostępnych dla użytkownika oraz do zarządzania ograniczeniami hasła. Jeżeli w bazie danych nie określono żadnych profili, wtedy jest wykorzystany profil domyślny określający brak ograniczeń dostępu do zasobów dla wszystkich użytkowników.

W tabeli 10.3 wyszczególniono zasoby, do których można ograniczyć dostęp za pomocą profili.

Ustawienia PASSWORD_REUSE_MAX oraz PASSWORD_REUSE_TIME wzajemnie się wykluczają. Jeżeli jeden z tych parametrów jest ustawiony na określoną wartość, wartość drugiego musi być ustawiona jako UNLIMITED.

Tabela 10.3. *Zasoby, które można ograniczać przez odpowiednie ustawianie profili*

Zasób	Opis
SESSION_PER_USER	Liczba współbieżnych sesji otwieranych przez użytkownika w instancji.
CPU_PER_SESSION	Czas pracy procesora poświęcony jednej sesji, wyrażony w setnych częściach sekundy.
CPU_PER_CALL	Czas pracy procesora wykorzystany do przetwarzania jednego kroku instrukcji SQL: analizy składniowej, wykonywania instrukcji lub pobierania danych, wyrażony w setnych sekundy.
CONNECT_TIME	Czas w minutach, w ciągu którego sesja może realizować połączenie z bazą danych.
IDLE_TIME	Czas w minutach, w ciągu którego niewykorzystywana sesja może realizować połączenie z bazą danych.
LOGICAL_READS_PER_SESSION	Liczba bloków bazy danych odczytywanych w czasie sesji.
LOGICAL_READS_PER_CALL	Liczba bloków bazy danych odczytywanych w czasie kroku przetwarzania instrukcji SQL: analizy składniowej, wykonywania lub pobierania.
PRIVATE_SGA	Wielkość przestrzeni prywatnej, jaką sesja może przydzielić w dzielonym obszarze SQL (ang. *Shared SQL Pool*) globalnego obszaru systemu SGA (dla serwera wielowątkowego MTS).
COMPOSITE_LIMIT	Limit złożony, bazujący na poprzednich limitach.
FAILED_LOGIN_ATTEMPTS	Liczba kolejnych nieudanych prób zalogowania, po przekroczeniu której nastąpi zablokowanie konta.
PASSWORD_LIFE_TIME	Czas ważności hasła wyrażony jako liczba dni.
PASSWORD_REUSE_TIME	Czas, po którym można ponownie wykorzystać hasło, wyrażony w dniach.
PASSWORD_REUSE_MAX	Liczba określająca, ile razy trzeba zmienić hasło przed ponownym wykorzystaniem już używanego hasła.
PASSWORD_LOCK_TIME	Czas zablokowania hasła po przekroczeniu ustawienia FAILED_LOGIN_ ATTEMPTS, wyrażony w dniach.
PASSWORD_GRACE_TIME	Okres „tymczasowego okresu ważności", kiedy hasło może być w dalszym ciągu zmienione, po osiągnięciu jego PASSWORD_LIFE_TIME. Parametr ten wyrażany jest jako liczba dni.
PASSWORD_VERIFY_FUNCTION	Nazwa funkcji wykorzystanej do oceny złożoności hasła. Baza danych Oracle dostarcza jedną taką funkcję. Istnieje możliwość jej edytowania.

Jak wynika z tabeli 10.3, istnieje możliwość ograniczenia dostępu do pewnej liczby zasobów. Warto podkreślić, że wszystkie te ograniczenia są reakcyjne — żadna akcja nie występuje przed przekroczeniem limitu zasobu. Zatem profile nie mogą wspomagać zapobiegania wykorzystywania przez niekontrolowane zapytania dużych ilości zasobów systemowych przed wyczerpaniem określonego limitu. Dopiero, gdy ten limit zostanie osiągnięty, wykonywanie instrukcji SQL będzie zatrzymane.

Profile są tworzone za pomocą polecenia **create profile**. Polecenie **alter profile**, które przedstawiono w poniższym przykładzie, służy do modyfikowania istniejących profili. Tutaj następuje zmiana profilu DEFAULT bazy danych w celu ustawienia maksymalnego czasu jałowego, tj. kiedy sesja jest nieaktywna, (ang. *idle time*) na 1 godzinę:

```
alter profile DEFAULT
idle_time 60;
```

Program narzędziowy *Security Manager* pakietu *OEM* daje możliwość tworzenia i zarządzania profilami poprzez graficzny interfejs użytkownika. Na rysunku 10.5 przedstawiono wygląd okna z domyślnymi ustawieniami profilu, gdzie wykazano przydzielone zasoby profilu.

Rysunek 10.5. *Ustawienia profilu DEFAULT*

Stosowanie profili umożliwia również zarządzanie złożonością hasła i jego czasem istnienia. Zagadnienia te opisano w następnym podrozdziale.

Zarządzanie hasłem

Profile mogą służyć do zarządzania unieważnianiem, ponownym wykorzystaniem i złożonością haseł. Na przykład, można ograniczyć czas istnienia hasła i zablokować konto, którego hasło jest za stare. Możliwe jest również wymuszenie przynajmniej średniej złożoności haseł oraz ich blokowanie przy powtórzonych, nieudanych próbach zalogowania.

Na przykład, jeżeli parametr `FAILED_LOGIN_ATTEMPTS` profilu użytkownika jest ustawiony na wartość `5`, wtedy dozwolonych jest pięć kolejnych nieudanych prób zalogowania. Szósta próba zakończona niepowodzeniem spowoduje zablokowanie konta.

Jeżeli podczas piątej próby zostanie podane prawidłowe hasło, wtedy licznik nieudanych prób zalogowania zostaje wyzerowany. Umożliwia to przeprowadzenie kolejnych 5 nieudanych prób zalogowania przed zablokowaniem konta.

Poniżej przedstawiono przykładowe polecenia prowadzące do utworzenia profilu `LIMITED_PROFILE` dla użytkownika `JANE`:

```
create profile LIMITED_PROFILE limit
FAILED_LOGIN_ATTEMPTS 5;
```

```
create user JANE identified by EYRE
profile LIMITED_PROFILE;

grant CREATE SESSION to JANE;
```

W razie wystąpienia pięciu kolejnych nieudanych połączeń z kontem JANE system Oracle automatycznie zablokuje to konto. Jeśli następnie zostanie podane prawidłowe hasło dla konta JANE, nastąpi wyświetlenie komunikatu o błędzie:

```
Connect jane/eyre
ERROR: ORA-2800: the account is locked
```

Aby odblokować konto, należy zastosować klauzulę **account unlock** polecenia **alter user** wydanego z poziomu konta DBA —administratora bazy danych. Poniżej przedstawiono odpowiedni przykład:

```
alter user JANE account unlock;
```

Po odblokowaniu konta połączenia z kontem JANE są ponownie dozwolone. Istnieje również możliwość ręcznego zablokowania konta za pomocą klauzuli **account lock** polecenia **alter user**.

```
alter user JANE account lock;
```

Jeśli konto zostanie zablokowane w związku z powtórzonymi nieudanymi próbami połączenia, jego odblokowanie nastąpi automatycznie po przekroczeniu wartości ustawienia profilu konta PASSWORD_LOCK_TIME. Na przykład, jeżeli parametr PASSWORD_LOCK_TIME jest ustawiony na 1, wtedy konto JANE z poprzedniego przykładu pozostanie zablokowane przez jeden dzień, po czym nastąpi jego odblokowanie.

Można również ustalić maksymalny czas ważności hasła za pomocą ustawienia PASSWORD_LIFE_TIME w ramach profilu. Na przykład, można wymusić na użytkownikach o profilu LIMITED_PROFILE zmianę haseł co 30 dni.

```
alter profile LIMITED_PROFILE limit
PASSWORD_LIFE_TIME 30;
```

W powyższym przykładzie zastosowano polecenie **alter profile** w celu modyfikacji profilu LIMITED_PROFILE. Wartość PASSWORD_LIFE_TIME jest ustawiona na 30, zatem unieważnienie hasła każdego konta korzystającego z tego profilu nastąpi po 30 dniach. Jeżeli hasło zostało unieważnione, należy je zmienić podczas następnego zalogowania. Konieczne jest dokonanie tego w czasie określonym w profilu jako tymczasowy okres ważności. Parametr tymczasowego okresu ważności nazywa się PASSWORD_GRACE_TIME. Jeżeli hasło nie zostanie zmienione w ciągu tymczasowego okresu ważności, konto zostanie unieważnione.

> Jeżeli zastosowano parametr PASSWORD_LIFE_TIME, należy poinstruować użytkowników o sposobach łatwej zmiany haseł.

Istotna jest różnica pomiędzy unieważnieniem a zablokowaniem konta. Z treści tego podrozdziału wynika, że zablokowane konto może zostać z upływem czasu automatycznie odblokowane. Konto unieważnione wymaga osobistej interwencji ze strony administratora bazy danych w celu jego ponownego uaktywnienia.

W razie wykorzystywania możliwości unieważniania haseł, konta wykorzystywane przez aplikację powinny mieć inne ustawienia profilu. W przeciwnym razie istnieje możliwość zablokowania tych kont, co spowoduje unieruchomienie aplikacji.

W celu ponownego uaktywnienia unieważnionego konta należy wykonać polecenia **alter user**. Stosowny przykład znajduje się poniżej. Najpierw hasło użytkownika JANE zostało unieważnione ręcznie przez administratora bazy danych:

```
alter user jane password expire;

User altered.
```

Następnie użytkownik JANE próbuje połączyć się ze swoim kontem. Kiedy podaje hasło, natychmiast jest proszony o wpisanie nowego hasła.

```
Connect jane/eyre
ERROR: ORA-28001: the account has expired

Changing password for jane
Old password:
New password:
Retype new password:
Password changed
Connected.
SQL>
```

Istnieje również możliwość zmuszenia użytkowników do zmiany haseł podczas pierwszego uzyskania dostępu do swoich kont. Przeprowadza się to za pomocą klauzuli **password expire** polecenia **create user**. Jednak polecenie **create user** nie pozwala na ustawienie daty unieważnienia nowego hasła ustawianego przez użytkownika. W tym celu konieczne jest zastosowanie parametru profilu PASSWORD_LIFE_TIME, co przedstawiono powyżej w odpowiednich przykładach.

Przeglądanie daty unieważnienia hasła jakiegokolwiek konta jest możliwe po wykonaniu zapytania na kolumnie Expire_Date perspektywy słownika danych DBA_USERS. Użytkownicy chcący sprawdzić datę unieważnienia haseł dla swoich kont mogą wykonać zapytanie na kolumnie Expire_Date perspektywy słownika danych USER_USERS. Mogą też tego dokonać za pomocą programu *SQL*Plus* albo za pomocą zapytań formułowanych przy pomocy aplikacji klienta.

Uniemożliwianie ponownego zastosowania hasła

Aby uniemożliwić ponowne zastosowanie hasła, można wykorzystać jeden z dwóch parametrów profilu: PASSWORD_REUSE_MAX lub PASSWORD_REUSE_TIME. Trzeba tylko pamiętać, że te dwa parametry wzajemnie się wykluczają. Jeżeli ustawiona zostanie wartość jednego z nich, drugi musi być ustawiony na wartość UNLIMITED.

Parametr PASSWORD_REUSE_TIME określa liczbę dni, po upływie których unieważnione hasło może być zastosowane ponownie. Na przykład, po ustawieniu parametru PASSWORD_REUSE_TIME na wartość 60 dane hasło nie może być zastosowane ponownie przez czas równy 60 dniom.

Parametr PASSWORD_REUSE_MAX określa liczbę zmian hasła, po których możliwe jest zastosowanie tego samego hasła. W przypadku próby ponownego ustawienia danego hasła przed osiągnięciem wartości granicznej system Oracle odrzuci tę próbę zmiany hasła.

Poniżej przedstawiono sposób ustawienia parametru PASSWORD_REUSE_MAX dla profilu LIMITED_PROFILE, którego przykładowy sposób tworzenia pokazano wcześniej w tym rozdziale.

```
alter profile LIMITED_PROFILE limit
PASSWORD_REUSE_MAX 3
PASSWORD_REUSE_TIME UNLIMITED;
```

Jeżeli teraz użytkownik JANE podejmie próbę ustawienia poprzednio wykorzystywanego hasła, ta próba zmiany hasła zakończy się niepowodzeniem. Poniżej znajduje się odpowiedni przykład. Użytkownik JANE ustawił hasło za pomocą polecenia:

```
alter user JANE identified by AUSTEN;
```

Po pewnym czasie nastąpiła jego ponowna zmiana:

```
alter user JANE identified by EYRE;
```

Podczas następnej zmiany hasła użytkownik JANE usiłuje ponownie wykorzystać poprzednio stosowane hasło i próba ta kończy się niepowodzeniem.

```
alter user JANE identified by AUSTEN;
alter user JANE identified by AUSTEN
*
ERROR at line 1:
ORA-28007: the password cannot be reused
```

Zatem użytkownik nie może ponownie ustawić swojego poprzedniego hasła i musi zastosować nowe.

Historia haseł jest umieszczana w tabeli o nazwie USER_HISTORY$ w schemacie SYS. W tej tabeli system Oracle składuje identyfikator użytkownika userid, zaszyfrowaną wartość hasła oraz datę i znacznik czasu utworzenia hasła. Po przekroczeniu wartości parametru PASSWORD_REUSE_TIME lub jeśli liczba zmian hasła przekroczy wartość parametru PASSWORD_REUSE_MAX, stare rekordy haseł są usuwane z tabeli SYS.USER_HISTORY$. Pozwala to systemowi Oracle na rozpoznawaniu niedozwolonych haseł i odrzucanie ich.

Ze względu na to, że historie haseł są składowane w tabeli będącej w posiadaniu użytkownika SYS, dane te znajdują się w przestrzeni tabel SYSTEM. Dlatego utrzymywanie bardzo dużej liczby nieważnych haseł dla bardzo licznych użytkowników, którzy często zmieniają swoje hasła, może spowodować sytuację, w której wymagania przestrzeni tabeli historii haseł (SYS.USER_HISTORY$) mogą wpływać na wymagania przestrzeni tabel SYSTEM.

Ustawienie złożoności haseł

Istnieje możliwość wymuszenia pewnych standardów złożoności haseł stosowanych przez użytkowników. Na przykład do takich wymogów można zaliczyć minimalną liczbę znaków, niedopuszczenie prostych wyrazów lub obecność przynajmniej jednej cyfry lub

znaku interpunkcyjnego w haśle. Parametr PASSWORD_VERIFY_FUNCTION poleceń **create profile** oraz **alter profile** określa nazwę funkcji, która będzie oceniać hasła. Wtedy w razie próby ustawienia hasła, które nie spełnia danych kryteriów, hasło to nie zostanie zaakceptowane. Przykładowo, wyrazy austen oraz eyre mogłyby być odrzucone jako hasła, ponieważ nie zawierają żadnych numerycznych wartości.

Uproszczenie procesu wymuszania złożoności haseł jest możliwe dzięki funkcji zwanej VERIFY_FUNCTION. Domyślnie ta funkcja *nie* jest tworzona. Utworzenie funkcji VERIFY_FUNCTION zachodzi tylko po uruchomieniu skryptu *utlpwdmg.sql* znajdującego się w podkatalogu */rdbms/admin* katalogu podstawowego oprogramowania Oracle. Poniższy listing przedstawia skróconą wersję tego pliku. Dalej omówiono te części pliku, które wyróżniono pogrubieniem czcionki.

```
Rem utlpwdmg.sql
Rem
Rem  Copyright (c) Oracle Corporation 1996. Wszelkie prawa
Rem  zastrzeżone.
Rem
Rem    NAZWA
Rem      utlpwdmg.sql - skrypt dla limitów zasobów domyślnego hasła
Rem
Rem    OPIS
Rem      To jest skrypt do uaktywnienia opcji zarządzania hasłem
Rem      przez ustawienie limitów zasobów domyślnego hasła.
Rem
Rem    UWAGI
Rem      Ten plik zawiera funkcję minimalnego sprawdzania złożoności
Rem      hasła. To jest raczej próbna funkcja, którą użytkownik może
Rem      użyć, aby opracować funkcję dla rzeczywistych sprawdzeń
Rem      złożoności, które chciałby wykonać dla nowego
Rem      hasła.
Rem
Rem    asurpur     12/12/96 - Zmiana nazwy funkcji_weryfikacji_hasła
Rem    password_verify_function
-- Ten skrypt ustawia parametry zasobów domyślnego hasła.
-- Aby uaktywnić właściwości hasła, należy ten skrypt uruchomić.
-- Domyślne parametry zasobów mogą być zmienione według potrzeb.
-- Dostarczona jest również funkcja sprawdzenia złożoności domyślnego
-- hasła. Ta funkcja wykonuje minimalne sprawdzenia złożoności, takie
-- jak minimalną długość hasła, czy hasło nie jest takie samo jak
-- nazwa użytkownika itd. Użytkownik może tę funkcję wzbogacić według
-- swoich potrzeb. Ta funkcja musi być utworzona w schemacie SYS.
-- Przed uruchomieniem tego skryptu należy zestawić połączenie za pomocą
-- polecenia: sys/<hasło> as sysdba.

CREATE OR REPLACE FUNCTION verify_function
(username varchar2,
  password varchar2,
  old_password varchar2)
  RETURN boolean IS
   n boolean;
   m integer;
   differ integer;
   isdigit boolean;
   ischar  boolean;
   ispunct boolean;
```

```
   digitarray varchar2(20);
   punctarray varchar2(25);
   chararray varchar2(52);

BEGIN
   digitarray:= '0123456789';
   chararray:= 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
   punctarray:='!"#$%&()''*+,-/:;<=>?_';

   -- Sprawdź, czy hasło jest takie samo jak nazwa użytkownika
   IF password = username THEN
     raise_application_error(-20001, 'Hasło takie samo jak nazwa
                             użytkownika');
   END IF;

   -- Sprawdź występowanie minimalnej długości hasła
   IF length(password) < 4 THEN
      raise_application_error(-20002,'Długość hasła mniejsza niż 4');
   END IF;

   -- Sprawdź, czy hasło nie jest za proste. Można podać słownik
   -- wyrazów za pomocą którego moduł sprawdzający nie pozwoli na
   -- zastosowanie słów, które są zbyt proste na hasło.
IF password IN ('witam','hasło','oracle','komputer',abcd') THEN
      raise_application_error(-20002, 'Hasło za proste');
   END IF;

   -- Sprawdź, czy hasło zawiera przynajmniej jedną literę, jedną
   -- cyfrę i jeden znak interpunkcyjny.
   -- 1. Sprawdź występowanie cyfry
   isdigit:=FALSE;
   m := length(password);
   FOR i IN 1..10 LOOP
      FOR j IN 1..m LOOP
         IF substr(password,j,1) = substr(digitarray,i,1) THEN
            isdigit:=TRUE;
             GOTO findchar;
         END IF;
      END LOOP;
   END LOOP;
   IF isdigit = FALSE THEN
      raise_application_error(-20003, 'Hasło powinno zawierać przynajmniej jedną
      ➥literę, jedną cyfrę i jeden znak interpunkcyjny');
   END IF;
   -- 2. Sprawdź występowanie litery
   <<findchar>>
   ischar:=FALSE;
   FOR i IN 1..length(chararray) LOOP
      FOR j IN 1..m LOOP
         IF substr(password,j,1) = substr(chararray,i,1) THEN
            ischar:=TRUE;
             GOTO findpunct;
         END IF;
      END LOOP;
   END LOOP;
   IF ischar = FALSE THEN
      raise_application_error(-20003, 'Hasło powinno zawierać przynajmniej jedną
      ➥literę, jedną cyfrę i jeden znak interpunkcyjny');
```

```
   END IF;
   -- 3. Sprawdź występowanie znaku interpunkcyjnego
   <<findpunct>>
   ispunct:=FALSE;
   FOR i IN 1..length(punctarray) LOOP
      FOR j IN 1..m LOOP
         IF substr(password,j,1) = substr(punctarray,i,1) THEN
            ispunct:=TRUE;
             GOTO endsearch;
         END IF;
      END LOOP;
   END LOOP;
   IF ispunct = FALSE THEN
      raise_application_error(-20003, 'Hasło powinno zawierać przynajmniej jedną
      ➥literę, jedną cyfrę i jeden znak interpunkcyjny');
   END IF;

   <<endsearch>>
   -- Sprawdź, czy hasło różni się od poprzedniego hasła przynajmniej
   -- 3 literami
   IF old_password = '' THEN
      raise_application_error(-20004, 'Stare hasło ma wartość
                              zerową');
   END IF;
   -- Wszystko jest w porządku; return TRUE ;
   RETURN(TRUE);
   differ := length(old_password) - length(password);

   IF abs(differ) < 3 THEN
      IF length(password) < length(old_password) THEN
         m := length(password);
      ELSE
         m := length(old_password);
      END IF;
      differ := abs(differ);
      FOR i IN 1..m LOOP
          IF substr(password,i,1) != substr(old_password,i,1) THEN
             differ := differ + 1;
          END IF;
      END LOOP;
      IF differ < 3 THEN
          raise_application_error(-20004, 'Hasło powinno różnić się
                                  przynajmniej 3 znakami');
      END IF;
   END IF;
   -- Wszystko jest w porządku; return TRUE ;
   RETURN(TRUE);
END;
/

-- Ten skrypt zmienia parametry domyślne dla opcji zarządzania hasłami
-- Password Management. To znaczy, że wszyscy użytkownicy w systemie
-- będą mieli uaktywnioną opcję zarządzania hasłami Password
-- Management i będą mieli ustawione podane poniżej wartości, dopóki
-- dla użytkownika nie zostanie utworzony i przydzielony
-- inny profil z parametrami o wartościach innych lub UNLIMITED.
```

```
ALTER PROFILE DEFAULT LIMIT
PASSWORD_LIFE_TIME 60
PASSWORD_GRACE_TIME 10
PASSWORD_REUSE_TIME 1800
PASSWORD_REUSE_MAX UNLIMITED
FAILED_LOGIN_ATTEMPTS 3
PASSWORD_LOCK_TIME 1/1440
PASSWORD_VERIFY_FUNCTION verify_function;
```

Funkcję tę należy utworzyć w schemacie SYS.

Pierwsze trzy warunki `if` tej funkcji służą do sprawdzenia, czy hasło jest takie samo jak nazwa użytkownika, czy hasło składa się z mniej niż czterech znaków i czy hasło występuje w zbiorze określonych wyrazów (niedostateczna złożoność hasła). Warunki te można modyfikować, można także dodawać własne ustawienia. Na przykład, firmowe wytyczne zabezpieczenia aplikacji mogą wymagać stosowania haseł składających się przynajmniej z sześciu znaków. W celu wprowadzenia takiego warunku wystarczy zmodyfikować odpowiednią część pliku *ultpwdmg.sql* przed jego uruchomieniem.

Następną, dłuższą częścią tej funkcji jest trójetapowe sprawdzanie zawartości ciągu znaków hasła. Zaakceptowanie proponowanego hasła jest możliwe, jeśli zawiera ono przynajmniej jedną literę, jedną cyfrę oraz jeden znak interpunkcyjny. Także te ustawienia mogą być edytowane, podobnie jak w przypadku wyżej opisanych warunków. Na przykład, jeśli stosowanie w hasłach znaków interpunkcyjnych nie jest konieczne, tę część kontroli propozycji hasła można po prostu pominąć.

Kolejny fragment wyżej przedstawionej funkcji służy do porównywania starego hasła z proponowanym nowym hasłem na zasadzie przyrównania znaku po znaku. Jeżeli wyrażenia te nie różnią się na przynajmniej trzech miejscach, wtedy nowe hasło jest odrzucane.

Weryfikacja dokonywana przez funkcję VERIFIY_FUNCTION nie jest tak sprawna, jak pełniące tę samą rolę samą funkcje działające na poziomie systemu operacyjnego. Jeśli istnieje potrzeba wykorzystywania bardziej złożonego hasła i lepszego systemu jego sprawdzania, należy zmodyfikować procedurę VERIFY_FUNCTION postępując wedle wskazówek zaprezentowanych wcześniej.

Ostatnie polecenie w powyższym skrypcie nie jest już częścią funkcji. Jest to polecenie zmiany profilu **`alter profile`** zmieniające domyślny profil DEFAULT. Zmiana profilu DEFAULT zachodzi dla każdego użytkownika bazy danych, który używa profilu DEFAULT. Polecenie pokazane powyżej powoduje ustawienie następujących limitów: 60 dni czasu istnienia hasła, 10 dni tymczasowego okresu ważności, 1800 dni bez możliwości ponownego zastosowania hasła i blokowanie konta po trzech nieudanych próbach logowania z automatycznym odblokowaniem konta po jednej minucie (1/1440 dnia). Oczywiście, powyższe parametry mogą się różnić od parametrów wymaganych w danej aplikacji. Najważniejsze jest ostatnie ustawienie PASSWORD_VERIFY_FUNCTION. Określa ono, że funkcją weryfikującą jest funkcja utworzona przez skrypt *utlpwdmg.sql* VERIFY_FUNCTION.

Ta funkcja będzie stosowana tylko w przypadku użytkowników korzystających z podanego profilu.

Należy zwrócić uwagę, że funkcja weryfikująca VERIFY_FUNCTION nie daje dostępu do bazy danych ani nie modyfikuje żadnych wartości bazy danych. Także ewentualne modyfikacje tej funkcji nie powinny wymagać dostępu do bazy danych lub zmian bazy danych.

Można zmieniać domyślny profil w celu zastosowania funkcji VERIFY_FUNCTION bez zmiany parametrów unieważnienia hasła.

```
alter profile DEFAULT limit
PASSWORD_VERIFY_FUNCTION VERIFY_FUNCTION;
```

W razie zmiany profilu domyślnego DEFAULT należy zapewnić wszystkim użytkownikom tego profilu możliwość korzystania z niego. Na przykład, jeśli użytkownicy SYS oraz SYSTEM wykorzystują profil DEFAULT, zarządzanie ich hasłami odbywać by się mogło według określonych wyżej ustawień. W pewnych warunkach mogłoby to być niekorzystne. W celu uproszczenia zarządzania profilami można utworzyć nowy profil i przydzielić go użytkownikom, którzy nie są administratorami bazy danych oraz użytkownikom nieposiadającym dostępu do żadnej aplikacji. Wadą tego rozwiązania jest konieczność pamiętania o przydzielaniu odpowiedniego profilu wszystkim nowym użytkownikom. Standaryzowanie procedur zarządzania ustawieniami dotyczącymi użytkowników znacznie zwiększa szanse powodzenia wdrożenia tego procesu.

Nazwą funkcji weryfikacji haseł nie musi być VERIFY_FUNCTION. Z powyższego listingu wynika, że nazwa funkcji jest podawana jako parametr w poleceniu **alter profile**. Nazwa VERIFY_FUNCTION może być zastosowana prawie do każdej innej funkcji. Warto stosować takie nazwy funkcji, aby wskazywały one ich znaczenie dla bazy danych. Na przykład opisywana wyżej funkcja może nosić nazwę WERYFIKACJA_HASLA_ORACLE. Takie postępowanie zwiększa prawdopodobieństwo, że inny administrator bazy danych nie będzie miał problemów z określeniem funkcji pełnionej przez dany program w aplikacji.

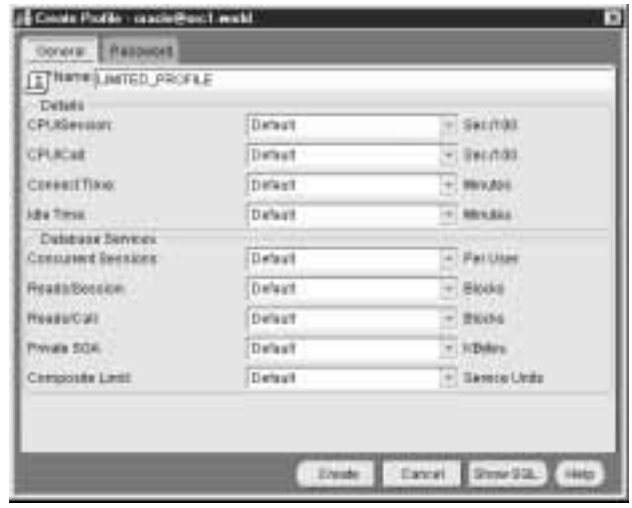
Program narzędziowy *Security Manager* pakietu *OEM* umożliwia tworzenie profili. Może on posłużyć do łatwego definiowania limitów zarówno ogólnych zasobów kontroli profilu, jak i zasobów hasła. Na rysunku 10.6 pokazano okno z ogólnymi ustawieniami profilu, a na rysunku 10.7 — okno z ustawieniami dotyczącymi hasła.

Dodatkowe opcje zarządzania hasłami są opisane w podrozdziale *Szyfrowanie haseł zwiększa możliwości kontroli*.

Wiązanie kont bazy danych z kontami hosta

Użytkownicy uzyskują dostęp do bazy danych po wprowadzeniu poprawnej nazwy użytkownika i hasła. Pewne możliwości systemu operacyjnego pozwalają na zapewnienie dodatkowego poziomu identyfikacji użytkownika.

Można połączyć konto bazy danych z kontem systemu operacyjnego na tym samym serwerze. Dwie nazwy tych kont mogłyby różnić się tylko prefiksem nazwy konta bazy danych. Prefiks ma wartość domyślną OPS$ ale może być ustawiony na inną wartość za pomocą parametru OS_AUTHENT_PREFIX pliku bazy danych *init.ora*. Ustawienie jego wartości na pusty łańcuch spowoduje brak prefiksu.



Rysunek 10.6. *Ogólne ustawienia profilu LIMITED_PROFILE*

Rysunek 10.7. *Ustawienia hasła profilu LIMITED_PROFILE*

W razie zmiany wartości parametru `OS_AUTHENT_PREFIX` **na wartość różną od** `OPS$`**, konta bazy danych mogą być stosowane albo jako konta zalogowania automatycznego, albo jako konta z dostępem za pomocą nazwy użytkownika i hasła. Nie jest możliwe wykorzystanie obydwóch tych metod naraz. Jeżeli prefiks identyfikacji ma wartość** `OPS$`**, wtedy można uzyskać dostęp do konta zarówno przez zalogowanie automatyczne, jak i za pomocą kombinacji nazwy użytkownika i hasła. W przypadku większości instalacji stosuje się prefiks** `OPS$`.

Poniżej rozważono przykładowe konto systemu operacyjnego o nazwie `FARMER`. Odpowiadającą mu nazwą konta bazy danych dla tego użytkownika jest `OPS$FARMER`. Po zalogowaniu do swojego konta systemu operacyjnego użytkownik `FARMER` może uzyskać dostęp do konta `OPS$FARMER` bez podawania hasła, jak pokazano niżej.

```
> sqlplus /
```

Prawy ukośnik zastępuje kombinację nazwy użytkownika i hasła, która normalnie byłaby wymagana w celu uzyskania dostępu do konta.

Możliwość zalogowania automatycznego nie jest dostępna na wszystkich platformach. Wprowadzenie `sqlplus /` w wierszu poleceń systemu *NT DOS* zwróci komunikat o błędzie *ORA-01017*.

Konta mogą być tworzone wraz z hasłami. Kontynuując powyższy przykład, konto `OPS$FARMER` może być utworzone po wydaniu następującego polecenia:

```
create user OPS$FARMER
identified by HASLO
default tablespace USERS
temporary tablespace TEMP;
```

Nawet jeżeli hasło nie jest używane, jest ono już określone. Dzięki temu jest możliwe uzyskanie dostępu do konta `OPS$FARMER` bazy danych z innego systemu operacyjnego. Warunkiem jest znajomość hasła konta bazy danych. Poniżej przedstawiono przykładowe polecenie połączenia z kontem `OPS$FARMER` z konta innego systemu operacyjnego:

```
> sqlplus ops$farmer/haslo
```

Występują dwa sposoby ominięcia powyższego problemu. Pierwszym z nich jest utworzenie konta bez określonego hasła za pomocą klauzuli **`identified externally`**. Polecenia potrzebne do tego celu pokazano poniżej. Dzięki temu nie zachodzi potrzeba podawania jawnego hasła konta przy utrzymaniu połączenia pomiędzy nazwą konta hosta a nazwą konta bazy danych.

```
create user OPS$FARMER
identified externally
default tablespace USERS
temporary tablespace TEMP;
```

Zastosowanie klauzuli **`identified externally`** powoduje sprawdzanie poprawności konta systemu operacyjnego wykorzystanego do uzyskania dostępu do bazy danych. Nazwa konta systemu operacyjnego oraz nazwa konta bazy danych muszą być identyczne (oprócz prefiksu nazwy konta bazy danych).

Drugą metodą jest utworzenie konta z niemożliwym hasłem. Metoda ta, opisana w podrozdziale *Ustawianie niemożliwych haseł* w dalszej części niniejszego rozdziału, uniemożliwia użytkownikowi zalogowanie się do konta bazy danych inaczej, niż przez konto systemu operacyjnego związane z kontem bazy danych.

Istnieją sytuacje, gdy nie należy zezwalać użytkownikom na posiadanie konta `OPS$` z hasłem możliwym do wykorzystania. W razie potrzeby umożliwienia użytkownikowi logowania się zarówno bezpośrednio z systemu operacyjnego, jak i z odległego konta przez program *Oracle Net*, zastosowanie konta z hasłem dostępu zdalnego może być korzystne. Przykładowo, jeżeli programista jest połączony z bazą danych na poziomie systemu operacyjnego, podczas testowania skryptu może zaistnieć sytuacja, w której zachodzi jawne wyświetlenie jego hasła. Z oczywistych względów jest to niekorzystne. Problem ten rozwiązuje możliwość zalogowania automatycznego dzięki prefiksowi `OPS$`. Jeżeli ten

programista łączy się za pomocą dostępu zdalnego (parametr REMOTE_OS_AUTHENT w pliku parametrów bazy danych nie jest ustawiony na wartość TRUE), w celu uzyskania dostępu do bazy danych musi podać hasło.

Wykorzystanie pliku haseł do identyfikacji

W większości przypadków użytkownicy posiadający uprawnienia *DBA* (administratora bazy danych) mogą być identyfikowani przez system operacyjny. Na przykład, w systemach UNIX członek grupy *DBA* w pliku */etc/group* może połączyć się z bazą danych używając opcji **connect internal** w bazie danych Oracle8 lub Oracle8i lub opcji **connect / as sysdba** w bazie danych Oracle9i. Opcja **connect internal** nie jest już dostępna w systemie Oracle9i Jeżeli użytkownicy posiadający uprawnienia *DBA* nie mogą być identyfikowani przez system operacyjny, konieczne jest utworzenie i przechowywanie pliku haseł.

Aby utworzyć plik haseł, należy postępować według poniższych wskazówek:

1. Utworzyć plik haseł wykorzystując program narzędziowy *ORAPWD*.

   ```
   ORAPWD FILE=nazwa_pliku PASSWORD=haslo ENTRIES=maks_liczba_uzytk
   ```

   Program *ORAPWD* jest programem narzędziowym generującym plik haseł. W czasie wykonywania programu *ORAPWD* należy określić nazwę tworzonego pliku haseł razem z hasłem dostępu do SYS i INTERNAL. Parametr ENTRIES określa liczbę wpisów w pliku haseł. Trzeba pamiętać, że po utworzeniu pliku nie będzie można go już rozszerzyć, zatem należy ustawić dużą wartość parametru ENTRIES. Przekroczenie limitu wpisów pliku haseł spowoduje wystąpienie komunikatu o błędzie ORA-1996. Podczas ponownego tworzenia pliku haseł konieczne jest powtórne przyznanie uprawnień SYSDBA oraz SYSOPER.

2. Ustawić parametr inicjalizacji REMOTE_LOGIN_PASSWORDFILE na wartość EXCLUSIVE w pliku *init.ora*. Następnie należy zamknąć i ponownie uruchomić bazę danych, co pozwoli na uaktywnienie zmienionego parametru.

3. Przyznać uprawnienia SYSOPER oraz SYSDBA każdemu użytkownikowi, który ma zarządzać bazą danych. Odpowiednie polecenia przedstawiono w poniższych przykładach. Prawo SYSDBA daje użytkownikowi uprawnienia administratora bazy danych. Prawo SYSOPER pozwala użytkownikowi na wykonywanie operacji wspomagających działanie bazy danych. W celu przyznania użytkownikowi prawa SYSOPER lub SYSDBA konieczne jest połączenie wewnętrzne. Użytkownicy uprawnieni powinni teraz pomyślnie łączyć się z bazą danych za pomocą polecenia podobnego do pokazanego poniżej:

   ```
   connect george/mch11@PROD.eshelon AS SYSDBA
   ```

   Polecenie **revoke** może posłużyć do odebrania użytkownikowi uprawnień systemowych SYSOPER oraz SYSDBA. Odpowiednie polecenie przedstawiono poniżej:

   ```
   revoke SYSDBA from George;
   ```

   Przeglądanie użytkowników posiadających uprawnienia systemowe SYSOPER oraz SYSDBA jest możliwe po wykonaniu zapytania na tabeli V$PWFILE_USERS. Tabela V$PWFILE_USERS posiada wartość TRUE w kolumnie SysDBA, jeżeli użytkownikowi przyznano prawo SYSDBA oraz wartość TRUE w kolumnie SysOper, jeżeli użytkownikowi przyznano prawo SYSOPER.

Ochrona za pomocą haseł

Zarówno konta, jak i role mogą być chronione za pomocą haseł ustawianych w chwili ich tworzenia. Hasła mogą być modyfikowane za pomocą poleceń **alter user** oraz **alter role**.

Początkowe hasło dla konta jest ustawiane podczas wykonywania polecenia **create user**, jak pokazano na poniższym listingu. W tym przykładzie jest tworzone konto THUMPER z początkowym hasłem R3BB#T:

```
create user THUMPER
identified by R3BB#T;
```

Hasła kont powinny być zmieniane za pomocą polecenia **alter user**. Przykładowe polecenie **alter user** pokazano poniżej:

```
alter user THUMPER identified by NOWEHASŁO;
```

W celu zmiany hasła można wykorzystywać polecenie *SQL*Plus* **password**. Polecenie **password** powoduje, że użytkownik jest monitowany o podanie starego hasła, nowego hasła oraz o weryfikację nowego hasła. Wprowadzane wartości hasła nie są wyświetlane na ekranie.

Użytkownik może zmienić swoje własne hasło w programie *SQL*Plus* za pomocą polecenia **password**, czego przykład przedstawiono poniżej:

```
password
```

W celu dokonania zmiany hasła innego użytkownika po poleceniu **password** należy podać nazwę użytkownika.

```
password JANE
```

W odpowiedzi na prośbę systemu należy wprowadzić nowe hasło użytkownika JANE i dokonać jego weryfikacji. Polecenie **password** jest bardzo użyteczne dla użytkowników końcowych, ponieważ znacznie upraszcza sposób zmiany haseł. Jeśli zastosowanie polecenia **password** jest niemożliwe, wtedy trzeba posłużyć się następującym poleceniem:

```
alter user NAZWAUZYTKOWNIKA identified by NOWEHASLO;
```

> Posługiwanie się poleceniem **alter user** w celu zmiany hasła nie pozwala na wykorzystanie w pełni funkcji weryfikacji hasła przedstawionej wyżej. Firma Oracle zaleca stosowanie polecenia **password** w celu zmiany hasła.

Hasła dla ról są ustawiane w czasie tworzenia ról za pomocą polecenia **create role**. Nie jest konieczne ustawienie hasła dla roli. W razie ustawienia hasła dla roli trzeba określić, kiedy użytkownik ma ją aktywować. Poniżej przedstawiono przykładowe polecenie powodujące utworzenie roli.

```
create role ACCOUNT_CREATOR identified by HELPDESK_ONLY;
```

W celu zmiany hasła związanego z rolą można użyć polecenia **alter role**. Podobnie jak hasła, role mogą być również identyfikowane zewnętrznie, co wymusza powiązanie pomiędzy nazwą konta hosta a nazwą roli. W odróżnieniu od kont użytkowników, możli-

we jest posiadanie roli bez haseł (domyślnie). Usuwanie hasła z roli przeprowadza się za pomocą klauzuli **not identified**, jak pokazano w poniższym przykładzie:

```
alter role ACCOUNT_CREATOR not identified;
```

Po wykonaniu tego polecenia rola ACCOUNT_CREATOR nie będzie chroniona hasłem.

Role mogą być związane z uprawnieniami systemu operacyjnego. Jeżeli taka możliwość jest dostępna w systemie operacyjnym, wtedy można ją wywołać za pomocą klauzuli **identified externally** polecenia **alter role**. Po uaktywnieniu roli system Oracle sprawdza ustawienia systemu operacyjnego w celu weryfikacji dostępu. Sposób modyfikacji roli w celu wykorzystania tej możliwości zabezpieczenia przedstawiono w poniższym przykładzie:

```
alter role MANAGER identified externally;
```

W systemie VMS proces weryfikacji wykorzystuje identyfikatory uprawnień systemu operacyjnego. W większości systemów UNIX proces weryfikacji przebiega za pomocą pliku */etc/group*. Aby zastosować go w każdym systemie operacyjnym, parametr uruchomienia bazy danych OS_ROLES w pliku *init.ora* musi być ustawiony na wartość TRUE.

Kolejny przykład procesu weryfikacji dotyczy instancji bazy danych o nazwie local w systemie UNIX. Plik serwera */etc/group* może zawierać następujący wpis:

```
ora_local_manager_d:NONE:1:dora
```

Ten wpis przyznaje rolę MANAGER kontu nazwanemu Dora. Sufiks _d oznacza domyślne przyznawanie tej roli po wystąpieniu zalogowania na koncie Dora. Sufiks _a spowodowałby uaktywnienie tej roli z klauzulą **with admin option**. Sufiks _ad oznacza, że rola z klauzulą with admin option jest domyślną rolą użytkownika. W celu przyznania danej roli większej liczbie użytkowników należy dołączyć do pliku *etc/group* dodatkowe nazwy tych użytkowników. Poniżej przedstawiono stosowny przykład:

```
ora_local_manager_d:NONE:1:dora,judy
```

Zastosowanie tej opcji spowoduje uaktywnianie wszystkich ról w bazie przez system operacyjny.

Uprawnienia obiektowe

Uprawnienia obiektowe (ang. *object-level privileges*) dają użytkownikowi dostęp do danych, których nie jest posiadaczem. Zarządzanie uprawnieniami może być znacznie łatwiejsze dzięki stosowaniu ról. Możliwe jest również bezpośrednie nadawanie uprawnień. Zastosowanie ich jest konieczne w niektórych okolicznościach.

Uprawnienia tworzy się za pomocą polecenia **grant** i są one rejestrowane w słowniku danych. Użytkownikom można przyznać dostęp do tabel, perspektyw, sekwencji, jak również do ich synonimów a także możliwość wykonywania procedur, funkcji, pakietów i typów.

Uprawnienia, które mogą być przyznane w odniesieniu do różnych obiektów wyszczególniono w tabeli 10.4.

Tabela 10.4. *Dostępne uprawnienia obiektowe*

Prawo	Przyznane możliwości
ALTER	Pozwala na zmianę właściwości obiektu
DEBUG*	Pozwala na włączenie usuwania błędów z programu Java (przyznawane wraz z rolą JAVADEBUGPRIV).
DELETE	Pozwala na usuwanie wierszy z obiektu.
DEQUEUE*	Pozwala na usuwanie wiadomości z kolejki (używane wraz z pakietem DBMS_AQ).
ENQUEUE*	Pozwala na dodawanie wiadomości do kolejki (używane wraz z pakietem DBMS_AQ).
EXECUTE	Pozwala na uruchamianie funkcji, pakietu, procedury, biblioteki lub typu.
INDEX	Pozwala na tworzenie indeksu na tabeli.
INSERT	Pozwala na wstawianie wierszy do obiektu. To prawo może być przyznane dla określonych kolumn obiektu.
ON COMMIT REFRESH	Pozwala na tworzenie odświeżanych na żądanie perspektyw materializowanych dla określonej tabeli.
QUERY REWRITE	Pozwala na utworzenie perspektywy materializowanej w celu przepisania zapytania z wykorzystaniem określonej tabeli.
READ	Pozwala na uzyskanie dostępu do katalogu.
REFERENCE	Pozwala na tworzenie kluczy obcych, które odwołują się do tabeli.
SELECT	Pozwala na wykonywanie zapytań na obiekcie.
UNDER	Pozwala na tworzenie podperspektywy lub podtypu danej perspektywy lub typu.
UPDATE	Pozwala na modyfikowanie wierszy w obiekcie. To prawo może być przyznane dla określonych kolumn obiektu.
WRITE	Pozwala na zapisanie pliku w katalogu.

W tabeli 10.4 niektóre przywileje oznaczone gwiazdką (*). Są one dostępne jedynie wraz z pakietem, który wymieniono w ich opisie.

Klauzula **with grant option** pozwala użytkownikowi na przekazywanie uprawnień do obiektów bazodanowych. Odpowiedni przykład znajduje się poniżej, na listingu programu **SQL*Plus**. W tym przypadku użytkownik o nazwie THUMPER przyznaje użytkownikowi MCGREGOR prawo dostępu do tabeli o nazwie EMPLOYEE. Przyznawanymi uprawnieniami są SELECT i prawo częściowej modyfikacji UPDATE, dodano również klauzulę przekazywania uprawnień **with grant option**. Następnie użytkownik MCGREGOR przyznaje te uprawnienia użytkownikowi o nazwie JFISHER.

```
grant select, update (Employee_Name, Address)
on EMPLOYEE to MCGREGOR
with grant option;

connect MCGREGOR/FARMER
grant select on THUMPER.EMPLOYEE to JFISHER;
```

Przyznawanie uprawnień wraz z opcją *PUBLIC* powoduje udostępnienie tych uprawnień wszystkim użytkownikom bazy danych.

Jeżeli EMPLOYEE jest tabelą partycjonowaną, nie można przyznać dostępu z uprawnieniem SELECT dotyczącym tylko jednej partycji tej tabeli. Jednak można utworzyć perspektywę, która wybiera dane tylko z jednej partycji a następnie przyznawać innym użytkownikom dostęp z prawem SELECT do tej perspektywy. Co prawda, perspektywa taka stanie się dodatkowym obiektem wymagającym zarządzania, ale jest to dodatkowy sposób zabezpieczenia danych na poziomie partycji.

Zarządzanie uprawnieniami może szybko stać się zadaniem czasochłonnym. Każdemu użytkownikowi trzeba przyznać odpowiednie uprawnienia dla każdego obiektu w aplikacji bazy danych. Przykładowo, w przypadku małej aplikacji, w skład której wchodzi 20 tabel i 30 użytkowników, zachodzi potrzeba zarządzania co najmniej 600 uprawnieniami (20 tabel razy 30 użytkowników).

Po wprowadzeniu ról zarządzanie uprawnieniami stało się dużo łatwiejsze. Role można określić jako grupy uprawnień. W ten sposób role są przyznawane użytkownikom, co bardzo usprawnia proces zarządzania uprawnieniami.

Poniższy listing przedstawia przykład wykorzystania ról. W tym przypadku tworzone są dwie role. Pierwszej roli APPLICATION_USER przyznawane jest prawo systemowe CREATE SESSION. Użytkownik, któremu ta rola została przyznana, może zalogować się do bazy danych. Drugiej roli, którą jest DATA_ENTRY_CLERK, są przyznawane uprawnienia do wykonywania pewnych operacji na tabelach.

```
create role APPLICATION_USER;
grant CREATE SESSION to APPLICATION_USER;

create role DATA_ENTRY_CLERK;
grant select, insert on THUMPER.EMPLOYEE to DATA_ENTRY_CLERK;
grant select, insert on THUMPER.TIME_CARDS to DATA_ENTRY_CLERK;
grant select, insert on THUMPER.DEPARTMENT to DATA_ENTRY_CLERK;
```

Poszczególne role mogą być przyznawane innym rolom. Na przykład można przyznać rolę APPLICATION_USER roli DATA_ENTRY_CLERK, co przedstawiono w poniższym przykładzie:

```
grant APPLICATION_USER to DATA_ENTRY_CLERK;
```

Także taka rola może być przyznana użytkownikowi. Istnieje możliwość jej dynamicznej aktywacji lub dezaktywacji podczas sesji użytkownika za pomocą polecenia **set role**.

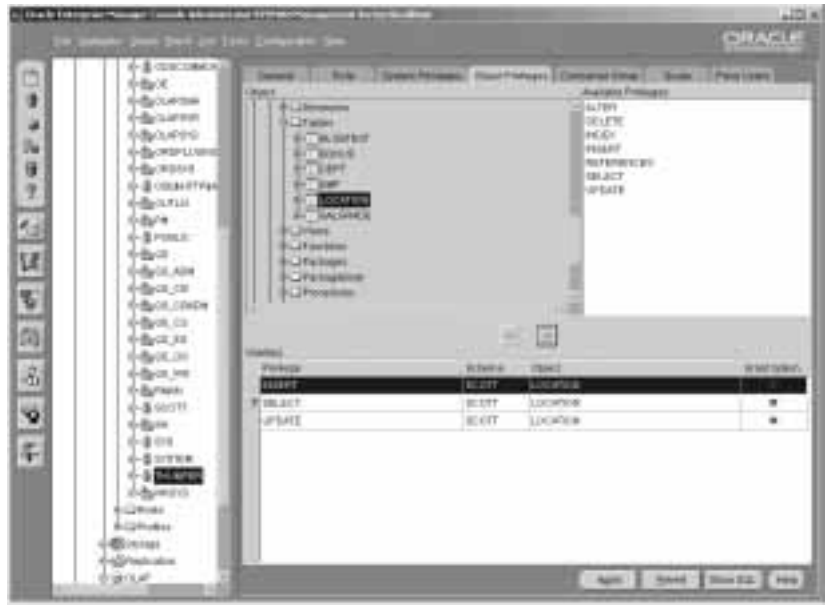
```
grant DATA_ENTRY_CLERK to MCGREGOR;
```

Program narzędziowy *Security Manager* pakietu *OEM* umożliwia dokonanie wyboru określonego użytkownika lub roli w celu przyznania im uprawnień obiektowych. Następnie za pomocą okna *Object Privileges* (*Uprawnienia obiektowe*) wybiera się uprawnienia obiektowe. Na przykładzie z rysunku 10.8 przedstawiono sposób przyznania użytkownikowi THUMPER uprawnień SELECT, INSERT oraz UPDATE dotyczących tabeli LOCATION.

Role i uprawnienia systemowe (takie jak CREATE TABLE) mogą być przyznawane użytkownikom z możliwością dalszego przekazywania ich innym użytkownikom. W przypadku roli w tym celu wykorzystywana jest klauzula **with admin option**. Poniższej przedstawiono przykładowy sposób przekazywania uprzednio utworzonej roli DATA_ENTRY_CLERK użytkownikowi BPOTTER, wraz z prawem do zarządzania tą rolą.

```
grant DATA_ENTRY_CLERK to BPOTTER with admin option;
```

Rysunek 10.8. *Przyznanie użytkownikowi uprawnień obiektowych*



Dzięki temu prawu użytkownik BPOTTER może przyznawać tę rolę (za pomocą polecenia **grant**) lub też odbierać ją (stosując polecenie **revoke**) innym użytkownikom. Może on również tę rolę usunąć.

Użytkownicy, którzy poprzez role mają uprawnienia do dokonywania pewnych czynności na tabeli, nie mogą tworzyć perspektyw lub procedur na bazie tej tabeli. Jest to potrzebne ograniczenie, ponieważ przyznania dokonane przez rolę są ważne tylko wtedy, gdy użytkownik jest zalogowany i gdy rola jest aktywna. Tworzenie perspektyw na tabelach przez użytkowników, którzy nie są ich właścicielami, wymaga bezpośredniego przyznania uprawnień do takich tabelach.

Dynamiczna natura ról jest bardzo przydatna do ograniczania uprawnień użytkowników. Jeżeli uaktywnienie roli (za pomocą polecenia **set role**) następuje w czasie, gdy użytkownik uruchamia aplikację, a jej dezaktywowanie następuje po opuszczeniu aplikacji, to użytkownik może korzystać z uprawnień tej roli tylko wtedy, gdy aplikacja jest używana.

Na przykład, po zalogowaniu użytkownika MCGREGOR do aplikacji może nastąpić wykonanie poniższego polecenia.

```
set role DATA_ENTRY_CLERK;
```

Kiedy ten użytkownik opuszcza aplikację, polecenie:

```
set role NONE;
```

dezaktywuje każde prawo, które było przyznane przez tę rolę.

Polecenie **revoke** służy do odbierania uprawnień i ról użytkownikom. Można odbierać albo niektóre uprawnienia użytkownika (przez ich jawne wypisanie), albo wszystkie uprawnienia użytkownika (za pomocą słowa kluczowego **all**). Poniżej przedstawiono

przykład polecenia odbierającego określone uprawnienie (DELETE) na tabeli EMPLOYEE jednemu użytkownikowi. Uprawnienia drugiego użytkownika są odbierane całkowicie.

```
revoke delete on EMPLOYEE from PETER;
revoke all on EMPLOYEE from MCGREGOR;
```

Kolejny przykład przedstawia polecenie odbierające rolę ACCOUNT_CREATOR z konta użytkownika o nazwie HELPDESK:

```
revoke ACCOUNT_CREATOR from HELPDESK;
```

Konta użytkowników mogą być całkowicie usuwane za pomocą polecenia:

```
drop user NAZWAKONTA cascade;
```

W razie zastosowania parametru cascade kasowanie uprawnień usuwanych kont nie jest wymagane. Polecenie **revoke** jest wykorzystywane w pokazany wyżej sposób głównie podczas zmiany statusu użytkowników lub kiedy aplikacje są przenoszone z jednego środowiska (np. test akceptacyjny) do drugiego (zastosowanie produkcyjne).

Istnieje ważna różnica pomiędzy odebraniem uprawnień przyznanych z klauzulą przyznawania **with grant option** a tymi, które przyznano z klauzulą zarządzania **with admin option**. Rozważono przykładową sytuację, w której użytkownik THUMPER przyznaje użytkownikowi MCGREGOR prawo dostępu do tabeli EMPLOYEE wraz z klauzulą **with grant option**:

```
grant SELECT on EMPLOYEE to MCGREGOR with grant option;
```

Użytkownik MCGREGOR może teraz przekazać to prawo użytkownikowi BPOTTER, również z klauzulą **with grant option**:

```
grant SELECT on THUMPER.EMPLOYEE to BPOTTER with grant option;
```

Jeżeli użytkownik THUMPER odbierze wcześniej przyznane uprawnienia użytkownikowi MCGREGOR:

```
revoke SELECT on EMPLOYEE from MCGREGOR;
```

wtedy użytkownik BPOTTER, który otrzymał prawo dostępu do tabeli EMPLOYEE od użytkownika MCGREGOR, nie może już posiadać uprawnień do tabeli EMPLOYEE, ponieważ użytkownik MCGREGOR także nie ma już dostępu do tej tabeli.

Odbieranie uprawnień przyznanych za pomocą klauzuli **with admin option** funkcjonuje inaczej. Kontynuując powyższy przykład, jeżeli użytkownik MCGREGOR otrzyma prawo systemowe z opcją **with admin option**, wtedy może przekazać takie prawo użytkownikowi BPOTTER. Jeżeli nastąpi odebranie prawa systemowego użytkownikowi MCGREGOR, wtedy użytkownik BPOTTER je zachowa.

Wykazy uprawnień

Informacje na temat przyznanych uprawnień są składowane w słowniku danych. Dostęp do tych danych można uzyskać za pomocą perspektyw słownika danych.

Perspektywy słownika danych wyszczególnione w tabeli 10.5 służą do wykazywania uprawnień, które przyznano w bazie danych. Dostępne są również perspektywy na poziomie użytkownika.

Tabela 10.5. *Perspektywy słownika danych związane z uprawnieniami*

Perspektywa słownika danych	Zawartość
DBA_ROLES	Nazwy ról i ich status hasła.
DBA_ROLE_PRIVS	Użytkownicy, którym przyznano role.
DBA_SYS_PRIVS	Użytkownicy, którym przyznano uprawnienia systemowe.
DBA_TAB_PRIVS	Użytkownicy, którym przyznano uprawnienia na tabelach.
DBA_COL_PRIVS	Użytkownicy, którym przyznano uprawnienia na kolumnach.
ROLE_ROLE_PRIVS	Role, które przyznano innym rolom.
ROLE_SYS_PRIVS	Uprawnienia systemowe, które przyznano rolom.
ROLE_TAB_PRIVS	Uprawnienia tabeli, które przyznano rolom.

Na przykład, istnieje możliwość otrzymania informacji dotyczących uprawnień systemowych przyznanych poszczególnym rolom.

```
select
      Role,             /*Nazwa roli*/
      Privilege,        /*Prawo systemowe*/
      Admin_Option      /*Czy została przyznana opcja admin?*/
 from ROLE_SYS_PRIVS;
```

Aby otrzymać listę przyznanych przywilejów dotyczących tabel, konieczne jest szukanie dwóch typów uprawnień: uprawnień przyznanych użytkownikom bezpośrednio oraz przyznanych za pomocą ról.

W celu przeglądania uprawnień przyznanych bezpośrednio, należy wykonać zapytanie na perspektywie DBA_TAB_PRIVS. Poniżej przedstawiono odpowiedni przykład.

```
select
      Grantee,          /*Odbiorca przyznania*/
      Owner,            /*Właściciel obiektu*/
      Table_Name,       /*Nazwa obiektu*/
      Grantor,          /*Użytkownik, który dokonał przyznania*/
      Privilege,        /*Przyznane prawo*/
      Grantable         /*Czy została przyznana opcja admin?*/
 from DBA_TAB_PRIVS;
```

Aby przeglądać uprawnienia tabel przyznawane przez rolę, należy odnaleźć rekordy danego użytkownika w perspektywie DBA_ROLE_PRIVS i porównać je z uprawnieniami tabel roli. Są one wykazane w perspektywie ROLE_TAB_PRIVS.

```
select
      DBA_ROLE_PRIVS.Grantee,       /*Odbiorca przyznania*/
      ROLE_TAB_PRIVS.Owner,         /*Właściciel obiektu*/
      ROLE_TAB_PRIVS.Table_Name,    /*Nazwa obiektu*/
      ROLE_TAB_PRIVS.Privilege,     /*Przyznane prawo*/
      ROLE_TAB_PRIVS.Grantable      /*Czy została przyznana opcja admin?*/
 from DBA_ROLE_PRIVS, ROLE_TAB_PRIVS
where DBA_ROLE_PRIVS.Granted_Role = ROLE_TAB_PRIVS.Role
  and DBA_ROLE_PRIVS.Grantee = 'nazwa użytkownika';
```

Takie zapytanie wyszuka uprawnienia tabel przyznane roli dla danego użytkownika. Aby przeglądać limity profili występujące w bieżącej sesji, można zastosować zapytanie na perspektywie USER_RESOURCE_LIMITS. Jej kolumnami są:

- Resource_Name — nazwa zasobu (np. SESSION_PER_USER);
- Limit — określony limit dotyczący danego zasobu.

Perspektywa USER_PASSWORD_LIMITS opisuje parametry profilu hasła dla użytkownika. Posiada ona te same kolumny, co perspektywa USER_RESOURCE_LIMITS.

Nie istnieje żadna wersja *DBA* perspektywy USER_PASSWORD_LIMITS. Jest ona ściśle ograniczona do bieżącej sesji użytkownika. Aby zobaczyć koszt związany z każdym dostępnym zasobem, należy wykonać zapytanie na perspektywie RESOURCE_COST. Administratorzy baz danych mogą uzyskiwać dostęp do perspektywy DBA_PROFILES, co umożliwia przeglądanie limitów zasobów dla wszystkich profili. Kolumna Resource_Type perspektywy DBA_PROFILES wskazuje, czy profil zasobu jest profilem PASSWORD czy KERNEL.

Istnieją jeszcze dwie perspektywy składające się z pojedynczej kolumny, które służą do przeglądania uaktywnionych dla bieżącej sesji uprawnień i ról. Są to:

- SESSION_PRIVS — kolumna Privilege wykazuje wszystkie uprawnienia systemowe dostępne w sesji, niezależnie od tego, czy są one przyznane bezpośrednio, czy przez role;
- SESSION_ROLES — kolumna Role wykazuje wszystkie role, które są bieżąco aktywne w sesji;
- SESSION_PRIVS oraz SESSION_ROLES są dostępne dla wszystkich użytkowników.

Ograniczanie dostępnych poleceń za pomocą tabel Product User Profile

Program *SQL*Plus* zapewnia dodatkowy poziom zabezpieczenia — poszczególne polecenia mogą być dezaktywowane dla określonych użytkowników. W ten sposób użytkownicy z uprawnieniem UPDATE dotyczącym danej tabeli mogą nie mieć możliwości łączenia się z poziomu wiersza poleceń programu *SQL*Plus* w celu modyfikowania zawartości tej tabeli w niekontrolowany sposób. W ten sposób administratorzy baz danych uzyskują metodę uniemożliwiania użytkownikom uzyskiwania dostępu do systemu operacyjnego z programu *SQL*Plus* (za pomocą polecenia **host**). Jest to użyteczne ograniczenie, szczególnie jeśli aplikacja daje możliwość uzyskania dostępu do programu *SQL*Plus* i dostęp użytkowników do systemu operacyjnego jest niepożądany.

Oprócz odebrania użytkownikom możliwości stosowania polecenia **host** z programu *SQL*Plus*, można również odebrać im możliwość wykorzystywania polecenia **connect**. Eliminacja dostępu do tych dwóch poleceń zmusi użytkowników do posługiwania się swoimi kontami. Jeśli zastosowano omawiane zabezpieczenia, wyniki wydania powyższych poleceń wyglądają następująco:

```
SQL>host
Invalid command: host
SQL> connect system/manager
Invalid command: connect
```

W każdym przypadku wyświetlany jest komunikat *invalid command* (*nieprawidłowe polecenie*). Użytkownik może korzystać z aplikacji jedynie z poziomu swojego konta.

W celu uzyskania takiego poziomu zabezpieczenia trzeba utworzyć tabele *Product User Profile*. Skrypt służący do ich tworzenia nosi nazwę *pupbld.sql* i znajduje się w podkatalogu */sqlplus/admin* katalogu podstawowego oprogramowania Oracle. Ten skrypt tworzy kilka tabel i perspektyw i powinien być uruchamiany z konta SYSTEM. Począwszy od Oracle8i skrypt ten uruchamiany jest automatycznie w momencie tworzenia bazy danych.

Dla *SQL*Plus* uzyskanie dostępu do najważniejszej tabeli jest możliwy za pomocą synonimu PRODUCT_USER_PROFILE. Kolumny o kluczowym dla celów zabezpieczeń znaczeniu wykazano w tabeli 10.6. Aby utworzyć zabezpieczenie o pożądanym poziomie, należy wprowadzić rekordy do tej tabeli.

Tabela 10.6. *Kolumny w tabeli PRODUCT_USER_PROFILE*

Nazwa kolumny	Opis
Product	Ustawić na *SQL*Plus*. Nazwa musi być zapisana w odpowiedni sposób, z uwzględnieniem małych i dużych liter.
Userid	Nazwy użytkowników, których polecenia są dezaktywowane, mają być zapisane dużymi literami.
Attribute	Nazwa dezaktywowanego polecenia musi być zapisana dużymi literami. Dezaktywowanie polecenia SET w programie *SQL*Plus* dezaktywuje również polecenia set role oraz set transaction.
Char_Value	Ustawić na DISABLED zapisując to słowo dużymi literami.

W celu dezaktywacji ról można również użyć tabel Product User Profiles. W tym celu należy dokonać ustawienia kolumny Attribute na ROLES i umieścić nazwę roli w kolumnie Char_Value. Dezaktywacja ról jest zwykle wykonywana wraz z dezaktywacją polecenia **set** (patrz tabela 10.6).

Zabezpieczenie hasła podczas logowania

Podczas łączenia się z komputera klienta do serwera bazy danych lub z jednej bazy danych do drugiej przez powiązanie baz danych, system Oracle domyślnie wysyła wprowadzone hasło w formie niezaszyfrowanej. W systemie Oracle można wymusić szyfrowanie wartości hasła przed jego wysłaniem. Aby uaktywnić szyfrowanie hasła, należy ustawić następujące parametry:

- dla komputerów klienckich parametr ORA_ENCRYPT_LOGIN w pliku *sqlnet.ora* na wartość TRUE;
- dla serwerów parametr DBLINK_ENCRYPT_LOGIN w pliku *init.ora* na wartość TRUE.

Po ustawieniu tych parametrów należy wyłączyć i ponownie uruchomić bazę danych. Od tego momentu hasła będą wysyłane od klienta do serwera w formie zaszyfrowanej.

Szyfrowanie haseł zwiększa możliwości kontroli

Znajomość sposobu szyfrowania i ustawiania hasła przez bazę danych umożliwia administratorom baz danych wykonywanie pewnych zadań, które bez tej znajomości nie byłyby możliwe do realizacji. Do tych zadań zalicza się ustawianie niemożliwych haseł oraz tymczasowe przejmowanie konta innego użytkownika. Zagadnienia te przedstawiono w kolejnych podrozdziałach.

Składowanie haseł

Po określeniu hasła dla konta użytkownika lub roli *zaszyfrowana* wersja tego hasła jest przechowywana w słowniku danych. Ustawienie tego samego hasła dla dwóch różnych kont spowoduje zaszyfrowanie tych haseł w odmienny sposób. Dla wszystkich haseł zaszyfrowana wartość składa się z 16 znaków i zawiera cyfry oraz duże litery.

Poprawność wprowadzanych haseł podczas weryfikacji użytkownika jest sprawdzana w następujący sposób: wpisane przez użytkownika hasło jest szyfrowane i wygenerowany szyfr jest porównywany z szyfrem ze słownika danych dla tego konta. Jeżeli te dwie wartości sobie odpowiadają, wtedy hasło uznawane jest za poprawne i autoryzacja kończy się powodzeniem.

Ustawianie niemożliwych haseł

Znajomość sposobu zapamiętywania haseł przez bazę danych jest ważna, ponieważ umożliwia wykorzystanie następnych możliwości zabezpieczania konta. Polega to na określeniu szyfru hasła, a nie samego hasła. Można spowodować wygenerowanie szyfru, który nie spełnia reguł formatu dla szyfrowanych haseł. W ten sposób zalogowanie się do takiego konta byłoby niemożliwe, ponieważ żadne hasło nie mogłoby wygenerować tego nieprawidłowego szyfru.

Poniżej rozważono przykład zapytania wybierającego konta i zaszyfrowane hasła. W tym przypadku zapytanie wybiera kolumny `Username` (*nazwa użytkownika*) oraz `Password` (*hasło*) z perspektywy `DBA_USERS`.

```
select
      Username,         /*Nazwa użytkownika*/
      Password          /*Zaszyfrowane hasło*/
from DBA_USERS
where Username in ('MCGREGOR','THUMPER','OPS$FARMER');

USERNAME      PASSWORD
------------------------------
MCGREGOR      1A2DD3CCEE354DFA
THUMPER       F3DE41CBB3AB4452
OPS$FARMER    4FF2FF1CBDE11332
```

Należy zwrócić uwagę, że każde z zaszyfrowanych haseł na powyższym listingu posiada 16 znaków. Samo hasło nie jest składowane w słowniku danych, tylko jego szyfr. Można zatem się zastanawiać, w jaki sposób program *Import* rozpoznaje hasła, skoro w przy-padku wykorzystania opcji importu *Full* (*Pełny*) hasła są importowane w poprawny sposób.

Program *Import* wykonuje polecenia SQL. Podczas importu z opcją *Full* program *Import* wykonuje nieudokumentowaną wersję polecenia **create user**. Importowanie użytkownika MCGREGOR z bazy danych pokazanej na powyższym listingu wygenerowałoby następujące polecenie **create user**:

```
create user MCGREGOR identified by VALUES '1A2DD3CCEE354DFA';
```

A zatem program *Import* stosuje nieudokumentowaną klauzulę **values** wewnątrz klauzuli **identified by** w celu określenia *zaszyfrowanego* hasła użytkownika, które je tworzy.

Nie jest to sytuacja wyjątkowa. Można stosować to samo polecenie do ustawienia szyfru dla każdego konta. Jeżeli ustawiony szyfr nie spełnia reguł szyfrowania (16 znaków, wszystkie występujące litery mają być duże), nie jest możliwe jego dopasowanie podczas identyfikacji użytkownika. Rezultatem zastosowania takiego hasła, którego szyfr jest nieprawidłowy, jest konto dostępne tylko z poprawnego konta systemu operacyjnego na serwerze. Poniżej przedstawiono przykładowy listing, gdzie szyfr ustawiono na wyrażenie 'no way'. Następnie jest wykonywane zapytanie na perspektywie DBA_USERS.

```
alter user OPS$FARMER identified by VALUES 'no way';

select
      Username,         /*Nazwa użytkownika*/
      Password          /*Zaszyfrowane hasło*/
from DBA_USERS
where Username in ('MCGREGOR','THUMPER','OPS$FARMER');

USERNAME      PASSWORD
------------------------------
MCGREGOR      1A2DD3CCEE354DFA
THUMPER       F3DE41CBB3AB4452
OPS$FARMER    no way
```

Teraz uzyskanie dostępu do konta OPS$FARMER jest niemożliwe z wyjątkiem dostępu przez konto FARMER na serwerze. Nawet wtedy jest ono dostępne jedynie przez zalogowanie automatyczne z wykorzystaniem prawego ukośnika /. Hasła niemożliwe są użyteczne w celu blokowania kont nieposiadających w nazwie prefiksu OPS$, do których nie powinno być możliwości bezpośredniego zalogowania. Przykładem takiego konta jest konto SYS.

Przejmowanie konta innego użytkownika

Ze względu na możliwość ustawiania zaszyfrowanych haseł, można tymczasowo przejąć jakiekolwiek konto i następnie ustawić je z powrotem na jego oryginalne hasło bez znajomości hasła konta. Możliwość tymczasowego przejmowania konta innego użytkownika jest bardzo użyteczna podczas testowania aplikacji lub wykrywania i rozwiązywania problemów już podczas produkcyjnego jej wykorzystywania.

Tymczasowe przejmowanie konta innego użytkownika wymaga wykonania następujących czynności:

1. Wykonanie zapytania na perspektywie DBA_USERS w celu określenia aktualnego zaszyfrowanego hasła dla danego konta.
2. Wygenerowanie polecenia **alter user**, które będzie konieczne w celu ponownego ustawienia zaszyfrowanego hasła na odpowiednią wartość.
3. Zachowanie w pliku buforowym polecenia **alter user**.
4. Zmiana hasła użytkownika.
5. Uzyskanie dostępu do konta użytkownika i przeprowadzenie zamierzonych czynności.
6. Uruchomienie pliku zawierającego polecenie **alter user** w celu ponownego ustawienia zaszyfrowanego hasła użytkownika na jego oryginalną wartość.

Ten proces jest wykonywany automatycznie za pomocą skryptu *SQL*Plus*. Skrypt ten generuje polecenia konieczne do ponownego ustawienia konta użytkownika.

```
REM*  become_another_user.sql
REM*
REM*  Ten skrypt generuje polecenia konieczne w celu
REM*  tymczasowego przejęcia konta innego użytkownika.
REM*
REM*  Skrypt musi być uruchomiony z konta DBA.
REM*
REM*  Zmienna wejściowa: nazwa użytkownika przejmowanego konta.
REM*
REM*  Etapy 1, 2 oraz 3: wykonaj zapytanie na perspektywie DBA_USERS.
REM*  Wygeneruj polecenie ALTER USER, które będzie konieczne do
REM*  ponownego ustawienia hasła na jego bieżącą wartość.
REM*
set pagesize 0 feedback off verify off echo off termout off
REM*
REM*  Utwórz plik o nazwie reset.sql do to przechowania wygenerowanych
REM*  poleceń.
REM*
spool reset.sql
REM*
REM*  Wybierz zaszyfrowane hasło z perspektywy DBA_USERS.
REM*
SELECT 'alter user &&1 identified by values '||''''||
password||''''||' profile '||profile||';'
FROM dba_users WHERE username = upper('&&1');

prompt 'host rm -f reset.sql'
prompt 'exit'
spool off
exit
```

W poleceniu **select** stosuje się dwa zestawy czterech apostrofów.

Powyższy skrypt generuje przed zakończeniem skrypt *reset.sql*. Ten plik składa się z trzech wierszy. Pierwszy z nich zawiera polecenie **alter user** z klauzulą **values**, po której następuje zaszyfrowane hasło. Drugi wiersz zawiera polecenie **host**, które usuwa plik *reset.sql* (ponieważ po wykorzystaniu jak podano w punkcie 6., nie będzie on już potrzebny). Trzeci wiersz zawiera polecenie **exit** umożliwiające wyjście z programu *SQL*Plus*. Przykładowy plik *reset.sql* pokazany jest poniżej:

```
alter user MCGREGOR identified by values '1A2DD3CCEE354DFA' profile DEFAULT;
host rm -f reset.sql
exit
```

Polecenie **rm -f** w drugim wierszu powinno być zastąpione przez polecenie usunięcia pliku odpowiednie dla danego systemu operacyjnego.

Następnie można wykonać czynności z punktów 4. oraz 5., które obejmują zmianę hasła użytkownika (za pomocą polecenia **alter user**) i uzyskanie dostępu do konta użytkownika. Te operacje wyszczególniono na poniższym listingu:

```
alter user MCGREGOR identified by MY_TURN;
connect MCGREGOR/MY_TURN
```

Potem nastąpi zalogowanie na konto MCGREGOR. Po zakończeniu testowania należy zalogować się w programie *SQL*Plus* i uruchomić powyższy skrypt *reset.sql*. Polecenie wykonania skryptu *reset.sql* pokazano poniżej:

```
sqlplus system/manager @reset
```

W razie jednoczesnego testowania wielu kont warto niekiedy uwzględniać nazwę użytkownika w nazwie pliku *reset.sql*. Zapobiegnie to sytuacji, w której pierwszy plik *reset.sql* zostałby nadpisany przez późniejszą wersję. Jeżeli opisywane czynności są wykonywane na kontach z prefiksem OPS$, należy zachować szczególną ostrożność, ponieważ znak dolara $ jest znakiem specjalnym w niektórych systemach operacyjnych (takich jak Unix).

Po przeprowadzeniu omawianej procedury konto jest ponownie ustawione na swoją oryginalną zaszyfrowaną wartość hasła i tym samym na oryginalne hasło. Testowanie konta może odbywać się bez potrzeby znajomości jego hasła i bez jego zniszczenia.

Pominięcie ograniczeń hasła podczas operacji przejmowania konta użytkownika

Istnieje opcja uniemożliwienia użytkownikom ponowne wykorzystywanie starych haseł (patrz podrozdział *Uniemożliwianie ponownego użycia hasła* w początkowej części tego rozdziału). Uniemożliwienie ponownego stosowania haseł może mieć wpływ na możliwość tymczasowego przejmowania konta użytkownika. Zgodnie z treścią wcześniejszego podrozdziału, standardowy zestaw czynności wykonywanych w celu tymczasowego przejmowania konta innego użytkownika jest następujący:

1. Wykonanie zapytania na perspektywie DBA_USERS w celu określenia aktualnego zaszyfrowanego hasła dla danego konta.

2. Wygenerowanie polecenia `alter user`, które będzie konieczne w celu ponownego ustawienia zaszyfrowanego hasła na odpowiednią wartość.
3. Zachowanie w pliku buforowym polecenia `alter user`.
4. Zmiana hasła użytkownika.
5. Uzyskanie dostępu do konta użytkownika i przeprowadzenie zamierzonych czynności.
6. Uruchomienie pliku zawierającego polecenie `alter user` w celu ponownego ustawienia zaszyfrowanego hasła użytkownika na jego oryginalną wartość.

Jeżeli ustawienia profilu użytkownika uniemożliwiają ponowne wykorzystanie hasła, przeprowadzenie czynności z punktu 6. nie powiedzie się, a zatem nie będzie możliwe ponowne ustawienie hasła użytkownika na jego oryginalną wartość. Konieczne będzie ustawienie nowego hasła dla tego użytkownika — co znacznie zmniejsza efektywność procedury tymczasowego przejmowania konta innego użytkownika.

W celu uniknięcia tej sytuacji należy utworzyć profil, który nie wymusza ograniczeń w ponownym ustawianiu już wykorzystanych haseł. Przed rozpoczęciem procedury tymczasowego przejmowania konta należy przypisać użytkownikowi oryginalny profil. W skrypcie opisywanym w poprzednim podrozdziale ustawienie profilu użytkownika jest przechwycone i zintegrowane w poleceniu `alter user`, które ponownie ustawia hasło użytkownika. Zatem czynności opisywanej procedury są następujące:

1. Sprawdzenie ustawień profilu użytkownika.
2. W razie potrzeby przypisanie użytkownikowi profilu bez limitu historii hasła, bez funkcji weryfikacji hasła oraz z możliwością nieograniczonego ponownego zastosowania hasła. Przykładowe polecenie utworzenia takiego profilu pokazano poniżej:

```
create profile temp_profile limit
password_verify_function null
password_reuse_time      unlimited
password_reuse_max       unlimited
```

3. Wykonanie zapytania na perspektywie `DBA_USERS` w celu określenia aktualnego zaszyfrowanego hasła oraz profilu dla danego konta.
4. Wygenerowanie polecenia `alter user`, które będzie konieczne w celu ponownego ustawienia zaszyfrowanego hasła i profilu na ich oryginalne wartości.
5. Zachowanie w pliku buforowym polecenie `alter user`.
6. Zmiana hasła użytkownika.
7. Uzyskanie dostępu do konta użytkownika i przeprowadzenie zaplanowanych czynności.
8. Uruchomienie pliku zawierającego polecenia `alter user` w celu ponownego ustawienia zaszyfrowanego hasła i profilu użytkownika na ich oryginalne wartości.

Wirtualne Prywatne Bazy Danych

Firma Oracle wprowadziła mechanizm *Wirtualnej Prywatnej Bazy Danych* (ang. *Virtual Private Database*, *VPD*) w wersjiOracle8i, wydanie 3., aby umożliwić szczegółową kontrolę dostępu w połączeniu z zabezpieczeniem aplikacji. Wirtualna prywatna baza danych pozwala na ustanawianie polityki bezpieczeństwa w formie predykatów (klauzul **where**), które są dołączane do każdego zapytania, które użytkownicy wykonują w bazie danych. Organizując dane za pomocą bazy *VPB* wystarczy tylko raz ustalić struktury bezpieczeństwa na poziomie serwera danych. Jako że procedury bezpieczeństwa są przypisane do danych zamiast do aplikacji, role systemu bezpieczeństwa są stosowane zawsze i z każdego punktu. Dzięki temu dane prezentowane przez zapytanie są zawsze takie same. Typ połączenia nie gra tu żadnej roli: czy to będzie aplikacja *SQL*Plus*, czy sterownik *ODBC*, dane mają ten sam wygląd. Wirtualna prywatna baza danych opiera się na kilku mechanizmach, gwarantujących poufność danych dla każdego użytkownika. Aby zabezpieczyć właściwy rozdział danych należy się upewnić, że tabele zostały skonstruowane tak, aby możliwe było limitowanie dostępu do nich, zgodnie z wartościami w jednej lub większej liczbie kolumn.

Częstą praktyką jest definiowanie klucza firmowego identyfikującego daną firmę, zwłaszcza jeśli z tych samych danych korzysta kilka instytucji na raz. Należy wtedy skojarzyć użytkowników z kluczami a następnie ustalić politykę, która pozwala użytkownikom na uzyskiwanie dostępu do poszczególnych, odpowiednich dla nich wierszy. W takim przypadku, kiedy zostanie utworzona perspektywa, zostanie zastosowana klauzula **where** ograniczająca użytkownikowi możliwości podglądu i manipulowania tylko do jego danych. W ten sposób automatycznie implementowana jest precyzyjna kontrola dostępu.

Dla przykładu zakłada się, że jest dokonywany wybór z tabeli `PATIENTS` (*Pacjenci*) z wykorzystaniem poniższej perspektywy:

```
select * from PATIENTS;
```

Jeśli tabela `PATIENTS` uwzględnia prowadzenie polityki bezpieczeństwa, co ogranicza możliwości pacjentów do przeglądania jedynie danych dotyczących siebie, wtedy perspektywa zostanie automatycznie przepisana w następujący sposób:

```
select *
  From  PATIENTS
where PATIENTS_ID =  sys_context ('PATIENTS_CONTEXT','ALL_PATIENTS_ID')
/
```

W tym przykładzie klauzula **where** została automatycznie zastosowana do perspektywy, jako że stosowana polityka bezpieczeństwa sprawia, że użytkownicy mogą mieć dostęp jedynie do swoich danych, nie ważne jakich informacji szukają lub jak konstruują pytanie. Identyfikator pacjenta jest pobierany z definiowanego przez użytkownika kontekstu aplikacji `PATIENT_CONTEXT`. Funkcja systemowa `sys_context` zwraca wartość atrybutu `ALL_PATIENTS_ID` dla kontekstu `PATIENT_CONTEXT`.

Wewnątrz samej bazy *VPD* trzeba nadal przyznawać użytkownikom odpowiednie prawa dla każdej z tabel, ale nie trzeba tworzyć osobnych perspektyw ani procedur, które

miałyby ograniczyć dostęp do danych należących do innych użytkowników. Tak więc używając bazy *VPD* nie trzeba się już dłużej martwić o użytkowników uzyskujących dostęp do danych poprzez *SQL*Plus*, którzy uzyskali różne uprawnienia za pośrednictwem aplikacji.

Tworzenie bazy VPD

Tworzenie wirtualnej prywatnej bazy systemu Oracle znacznie się różni od instalacji i konfiguracji większości narzędzi Oracle. Baza *VPD* nie jest aplikacją samą w sobie ale raczej jest instalowana wraz z bazą danych. Aby dokonać instalacji wirtualnej prywatnej bazy danych, należy podjąć następujące kroki:

1. Określić obiekty bazy danych i ich relacje.
2. Zdefiniować politykę bezpieczeństwa.
3. Utworzyć kontekst aplikacji.
4. Utworzyć pakiet, który ustawi kontekst.
5. Utworzyć funkcje polityki bezpieczeństwa.
6. Połączyć funkcje polityki z tabelami i perspektywami.

Istnieje kilka różnych sposobów, aby wykonać te czynności. Poniżej zaprezentowano metodę, która jest uważana za najwydajniejszą, najłatwiejszą do dalszego rozbudowywania, najpewniejszą i najlepiej zabezpieczoną.

Określenie obiektów bazy danych oraz ich wzajemnych relacji

Aby rozpocząć tworzenie bazy *VPD*, należy określić jakie obiekty będą wchodziły w jej zakres, jakie będą miały względem siebie relacje i wedle jakich kluczy będą tworzone procedury bezpieczeństwa. Na przykład w przychodni lekarskiej pacjenci będą mogli obejrzeć i skopiować swoje karty, ale nie będą w stanie zrobić tego z cudzymi.

Najważniejszą funkcją będzie utrzymanie poziomu zabezpieczeń i prywatności danych. Najważniejszym kluczem będzie identyfikator pacjenta, który determinuje sposób jego uzyskiwania dostępu do danych. W omawianym przykładzie (który jest ogromnie uproszczony) zakłada się istnienie jedynie dwóch tabel, w których będą przechowywane dane: `PATIENT_PERSONAL_INFORMATION` oraz `MEDICAL_INFORMATION`. `PATIENT_PERSONAL_INFORMATION` zawiera informacje o pacjencie: imię, nazwisko, adres, numer telefonu, podczas gdy tabela `MEDICAL_INFORMATION`, jak sugeruje nazwa, zawiera dane medyczne. Identyfikator `PATIENT_ID` jest używany do tego, aby połączyć dane przechowywane w obu tych tabelach.

Oto wynik połączenia obu tych tabel:

```
create table PATIENT_PERSONAL_INFORMATION
(Patient_Id          NUMBER(10) primary key,
 Physician_Id        NUMBER(6) not null,
 Patient_Username  VARCHAR2(10),
 Patient_Name      VARCHAR2(20),
 Patient_Address1  VARCHAR2(20),
 Patient_Address2  VARCHAR2(20),
```

```
 Patient_Phone     NUMBER(10)
);
create table MEDICAL_INFORMATION
(Patient_id          NUMBER(10) primary key,
 Test_Performed      VARCHAR2(30),
 Test_Results        VARCHAR2(50),
 Diagnosis           VARCHAR2(500)
);
```

Definiowanie polityki bezpieczeństwa

Polityka bezpieczeństwa opisywanej, przykładowej sytuacji mówi po prostu, że *pacjenci mogą zapoznawać się tylko i wyłącznie ze swymi własnymi danymi*. Wytyczne innych polityk bezpieczeństwa będą zapewne bardziej skomplikowane, co więcej, może być ich nawet kilka dla każdej jednej tabeli bądź perspektywy. Należy je formułować nadzwyczaj jasno i wyraźnie, ponieważ każde zdanie musi później być przeniesione na kod funkcji PL/SQL, który zostanie dołączony do wyznaczonej tabeli aplikacji lub perspektywy.

Tworzenie kontekstu aplikacji

Kontekst aplikacji jest nazwą zestawu atrybutów i wartości, które można ustawić a następnie powiązać z obecną sesją użytkownika. System Oracle zapewnia domyślny kontekst `USERNV`, który zawiera informacje systemowe o aktualnej sesji, takie jak nazwa użytkownika, host czy nazwa programu. Jeśli trzeba określić dodatkowe atrybuty dla użytkownika, takie jak `PATIENT_ID`, należy się posłużyć kontekstem aplikacji.

Używając prawa `CREATE ANY CONTEXT` można utworzyć kontekst aplikacji. Należy określić szczególną nazwę kontekstu a potem połączyć ją z nazwą pakietu, który implementuje kontekst. Nazwa kontekstu powinna być jedyna w swoim rodzaju i nie może nigdzie się powtarzać w całej bazie danych. Jeśli zdarzy się tak, że plik o takiej nazwie istnieje, system wyświetli informację o błędzie.

Aby utworzyć kontekst, należy posiadać prawo `CREATE ANY CONTEXT`. Poniżej pokazano przykład utworzenia kontekstu nazwanego `MEDICALSEC_CTX`, który będzie częścią pakietu PL/SQL. Kontekst ten zostanie zapisany w schemacie `MEDICAL_DEPT`. Oto składnia tworzenia kontekstu:

```
create context MEDICAL_SEC_CTX using MEDICAL_DEPT.MEDICAL_SEC;
```

Tworzenie pakietu, który ustawi kontekst

Kiedy kontekst zostanie utworzony, należy zbudować pakiet i funkcje, które go ustawią. Poniżej zaprezentowano przykład ustawiania atrybutu kontekstu `PATIENT_ID` za pomocą bieżącej nazwy użytkownika uzyskanej z kontekstu domyślnego, `USERNV`. Ta funkcja wykorzystuje nazwę użytkownika, aby sprawdzić niezbędne atrybuty w tabeli.

```
create or replace package MEDICAL_SEC is
   procedure GET_PATIENT_ID;
end MEDICAL_SEC;
/
create or replace package body MEDICAL_SEC is
   procedure GET_PATIENT_ID
```

```
    is
    PATIENT_ID_VAR NUMBER;
    begin
     select PATIENT_ID
       into PATIENT_ID_VAR from PATIENT_PERSONAL_INFORMATION
      where PATIENT_USERNAME = SYS_CONTEXT('USERENV','SESSION_USER');
     dbms_session.set_context('MEDICAL_SEC_CTX', 'PATIENT_ID', PATIENT_ID_VAR);
    end GET_PATIENT_ID;
end MEDICAL_SEC;
/
```

System Oracle dostarcza zestaw predefiniowanych funkcji w kontekście systemowym SYS_CONTEXT oraz wbudowany kontekst USERNV, który umożliwia zwrócenie nazwy użytkownika wykonującego procedurę. Istnieje także wiele różnych wartości, które można pozyskać z funkcji SYS_CONTEXT. Te, które mają coś wspólnego z zagadnieniami bezpieczeństwa, przedstawiono w tabeli 10.7.

Aby ustawić kontekst dla użytkownika sesji, należy się odwołać do funkcji, która była powiązana z kontekstem, kiedy był on tworzony. Można to zrobić z poziomu samej aplikacji lub poprzez wyzwalacz logowania. Używanie go gwarantuje także, że kontekst będzie zawsze ustawiony, niezależnie od sposobu, w jaki użytkownik zaloguje się do bazy danych. Aby ustawić wyzwalacz logowania, należy użyć wyzwalacza **on logon**, który wprowadzono w Oracle8i, wersja 8.1.5.

Tworzenie funkcji polityki bezpieczeństwa

Następnie należy skorzystać z funkcji PL/SQL, aby zaimplementować politykę bezpieczeństwa. Ta funkcja zostanie skojarzona z tabelą MEDICAL_INFORMATION.

Poniżej zaprezentowano proces, który jeśli funkcja jest już przygotowana, narzuca na zapytanie szczegółowe procedury bezpieczeństwa. Należy zaznaczyć, że w tym przypadku „zapytanie" oznacza każdą formę uzyskiwania dostępu do informacji z tabeli lub perspektywy, włączając w to (ale nie ograniczając się tylko do nich) instrukcje **select**, **insert**, **delete**, **update** oraz instrukcje podzapytań. Od chwili, w której polityka bezpieczeństwa zostanie skojarzona z tabelą lub widokiem, zawsze gdy użytkownik wystosuje zapytanie, system będzie się odwoływał do polityki bezpieczeństwa, która zwróci odpowiednią wartość w formie warunku dostępu do danych lub predykatu.

W praktyce predykat jest klauzulą **where**, która jest dopisywana do instrukcji SQL użytkownika, aby ograniczać wiersze, które zależnie od typu używanej instrukcji, będą zwracane, uaktualniane lub usuwane. Zmodyfikowane zapytanie jest oceniane i optymalizowane podczas analizy instrukcji i może być następnie udostępniane i ponownie używane, aby usprawnić działanie systemu.

Celem omawianego działania jest umożliwienie pacjentom oglądania tylko i wyłącznie ich własnych danych, korzystając z pomocy identyfikatora PATIENT_ID. Oto przykładowa procedura PL/SQL, dzięki której można uzyskać zamierzony efekt:

```
create or replace package MEDICAL_SEC as
 function MEDICAL_ID_SEC return VARCHAR2;
 END MEDICAL_SEC;
/
create or replace package body MEDICAL_SEC as
```

Tabela 10.7. *Parametry SYS_CONTEXT*

Parametr	Zwracana wartość
authentication_data	Dane używane do identyfikacji logującego się użytkownika.
authentication_type	Metoda identyfikacji użytkownika. Wartościami, które można uzyskać, są: ♦ DATABASE *(baza danych)*: identyfikacja typu nazwa_użytkownika/hasło; ♦ OS: zewnętrzna identyfikacja użytkowników przez system operacyjny; ♦ NETWORK (sieć): identyfikacja przez protokół sieciowy lub identyfikacja ANO; PROXY: identyfikacja połączenia przez serwer proxy OCI.
bg_job_id	Jeśli proces drugoplanowy Oracle ustanowił bieżącą sesję, zostanie zwrócony identyfikator zadania.
client_info	Używany w połączeniu z pakietem DBMS_APPLICATION_INFO w celu magazynowania informacji. Ten parametr zwraca do 64 bajtów informacji o sesji użytkownika.
current_schema	Nazwa domyślnego schematu, używanego jako bieżący schemat.
current_user	Nazwa użytkownika, z którego uprawnieniami działa bieżąca sesja.
current_userid	Identyfikator użytkownika, z którego uprawnieniami działa bieżąca sesja.
db_domain	Domena bazy danych, identyczna z tą podaną w parametrze inicjalizacji DB_DOMAIN.
db_name	Nazwa bazy danych identyczna z tą podaną w parametrze inicjalizacji DB_NAME.
entryid	Dostępny identyfikator pola obserwacji. Ustawia się go, jeśli parametr inicjalizacji AUDIT_TRAIL został ustawiony na wartość TRUE w pliku parametrów inicjalizacji.
external_name	Zewnętrzna nazwa użytkownika bazy danych. Rozpoznana nazwa jest zapamiętywana w certyfikacie użytkownika i zwracana dla procedur identyfikacji protokołu SSL wykorzystujących certyfikaty v.503.
fg_job_id	Jeśli proces pierwszoplanowy Oracle ustanowił bieżącą sesję, zostanie zwrócony identyfikator zadania.
host	Nazwa komputera, z którego połączył się klient.
instance	Numer identyfikacyjny bieżącej instancji.
isdba	Jeśli rola *DBA* jest włączona, zwrócona zostanie wartość TRUE. W przeciwnym wypadku FALSE.
network_protocol	Protokół sieciowy używany do połączenia.
os_user	Nazwa użytkownika systemu operacyjnego, który zapoczątkował sesję.
proxy_user	Nazwa użytkownika bazy danych, który rozpoczął sesję jako SESSION_USER.
proxy_userid	Identyfikator nazwy użytkownika bazy danych, który rozpoczął sesję jako SESSION_USER.
session_user	Nazwa użytkownika bazy danych, pod którą użytkownik jest identyfikowany. Wartość ta pozostaje niezmienna przez cała sesję.
session_userid	Identyfikator nazwy użytkownika bazy danych, pod którą bieżący użytkownik jest identyfikowany.
sessionid	Identyfikator sesji obserwacji.
terminal	Identyfikator systemu operacyjnego dla klienta w bieżącej sesji.

```
/* OGRANICZA INSTRUKCJE SELECT BAZUJĄC NA WARTOŚCI PATIENT_ID */
 function MEDICAL_ID_SEC return VARCHAR2
 is
   MY_PREDICATE VARCHAR2 (2000);
   begin
     MY_PREDICATE := 'PATIENT_ID=SYS_CONTEXT(''MEDICAL_SEC_CTX'',''PATIENT_ID'')';
     return MY_PREDICATE;
    end MEDICAL_ID_SEC;
end MEDICAL_SEC;
/
```

Aby nie komplikować kodu, w procedurze zrezygnowano z obsługi błędów.

W powyższym, przykładowym fragmencie kodu nastąpiło pobranie identyfikatora PATIENT_ID z kontekstu aplikacji MEDICAL_SEC_CTX i wygenerowane predykatu, który ma być dodany do zapytania na tabeli MEDICAL_NFORMATION. Aby przetestować działanie predykatu przyjęto, że wartością PATIENT_ID jest 2435678987. Zwrócony predykat będzie wyglądał tak:

```
PATIENT_ID = 2435678987
```

Ten predykat będzie użyty wraz z klauzulą **where**, co zapewni że pacjent będzie miał dostęp tylko do tych informacji, które zgadzają się z jego identyfikatorem.

Połączenie funkcji polityki z tabelami i widokami

System Oracle dostarcza pakietu PL/SQL o nazwie DBMS_RLS, który pozwala na zarządzanie zasadami polityki bezpieczeństwa. Każdy kto tworzy lub zarządza nimi, będzie musiał posiadać prawo do uruchamiania (**execute**) tego pakietu nadane przez użytkownika SYS. Tabela 10.8 pokazuje cztery procedury dostępne w pakiecie.

Tabela 10.8. *Procedury Pakietu DBMS_RLS*

Procedura	Funkcja
ADD_POLICY	Dołącza politykę bezpieczeństwa do tabeli bądź perspektywy.
DROP_POLICY	Usuwa politykę bezpieczeństwa z tabeli lub perspektywy.
REFRESH_POLICY	Wymusza ponowną analizę otwartych kursorów powiązanych ze starą polityką, aby natychmiast zastosować nową lub zmienioną politykę bezpieczeństwa.
ENABLE_POLICY	Włącza lub wyłącza politykę bezpieczeństwa, która uprzednio została dodana do tabeli bądź perspektywy.

Pakietu DBMS_RLS używa się, aby skojarzyć funkcje polityki z tabelami lub perspektywami. Istnieje wiele zmiennych, które można użyć z każdą z procedur. Są one zaprezentowane w tabeli 10.9.

Procedury DROP_POLICY, REFRESH_POLICY oraz ENABLE_POLICY wymagają użycia parametrów object_schema, object_name i policy_name. Dla procedury ENABLE_POLICY wymagany jest jedynie parametr enable.

W poniższym przykładzie dodano politykę nazwaną MEDICAL_POLICY do tabeli MEDICAL_INFORMATION:

Tabela 10.9. *Parametry pakietu DBMS_RLS dla procedury ADD_POLICY*

Parametr	Opis
object_schema	Nazwa schematu zawierającego tabelę lub perspektywę.
object_name	Nazwa tabeli lub perspektywy.
policy_name	Nazwa polityki, którą jest dodawana lub usuwana. Nazwa ta musi być unikatowa dla danej tabeli lub perspektywy.
function_schema	Schemat funkcji polityki.
policy_function	Nazwa funkcji, która generuje predykat dla polityki. Jeśli funkcja została zdefiniowana wewnątrz pakietu, to jego nazwa także musi być podana.
statement_types	Typy instrukcji do których stosuje się politykę. Może to być dowolna kombinacja instrukcji select, insert, update i delete. Domyślnie polityka odnosi się do nich wszystkich.
update_check	Kiedy jest ustawiony jako TRUE, każda wstawiona lub aktualizowana wartość będzie sprawdzana względem polityki bezpieczeństwa (argument opcjonalny).
enable	Określa, czy polityka jest włączona, czy też nie. Wartością domyślną jest TRUE.

```
execute DBMS_RLS.ADD_POLICY
('MEDICAL_DEPT', 'MEDICAL_INFORMATION', 'MEDICAL_POLICY', 'MEDICAL_DEPT',
'MEDICAL_SEC.MEDICAL_ID_SEC', 'SELECT',FALSE,TRUE);
```

Ta instrukcja tworzy politykę MEDICAL_POLICY, która z kolei powoduje, że funkcja MEDICAL_SEC.MEDICAL_ID_SEC zostanie uruchomiona, kiedy w tabeli MEDICAL_INFORMATION w schemacie MEDICAL_DEPT zostanie wykonane polecenie **select**.

Instrukcje zostaną przetworzone i umieszczone w dzielonym obszarze SQL, aby były dostępne dla innych użytkowników z podobnymi uprawnieniami. Jeśli jedna z reguł mówi, że dostęp do tabel jest możliwy tylko w godzinach pracy, czyli od 9.00 do 17.00, to w celu ograniczenia dostępu do tabeli oprócz zaprogramowania procedury trzeba także przygotować zadanie uruchamiane każdego popołudnia pod koniec dnia roboczego. Zadanie to ma unieważniać dzielony kod SQL. Można to zrobić w następujący sposób:

```
execute DBMS_RLS.REFRESH_POLICY

('<schemat w którym przechowywana jest polityka>', '<nazwa_tabeli>',
'<nazwa_polityki>';
```

W powyższym przykładzie polityka funkcjonuje w tym samym schemacie, co tabele aplikacji. W praktyce należy jednak przechowywać funkcje w obszarze należącym do administratora odpowiedzialnego za kwestie bezpieczeństwa, aby uchronić ich zawartość przed celowym lub przypadkowym usunięciem z tabel lub perspektyw.

Tworzenie wyzwalacza

Kiedy wszystkie elementy znajdą się już na swoim miejscu, należy utworzyć wyzwalacz, który będzie wprowadzał politykę bezpieczeństwa w życie. Zaleca się używanie wyzwalacza **logon**, co zagwarantuje, że polityka będzie stosowana niezależnie od sposobu, w jaki użytkownik połączy się z bazą danych.

Należy jednak przestrzec, że jeśli wyzwalacz nie działa prawidłowo, może dojść do sytuacji, w której nikt nie będzie w stanie zalogować się do bazy danych. W takim przypadku należy się zalogować jako użytkownik z uprawnieniami SYSDBA i usunąć wyzwalacz do czasu, kiedy błędy zostaną usunięte. W systemie Oracle9i składnia, której należy użyć w polu GUI logon SQL*Plus wygląda następująco: należy zalogować się na konto SYSTEM z hasłem MANAGER. Wymagana fraza as sysdba zostanie dodana po ciągu znaków połączenia. W przykładzie pokazanym niżej nazwą instancji jest MYDB9.WORLD.

```
User Name: SYSTEM
Password: MANAGER
Host String: MYDB9.WORLD AS SYSDBA
```

Obserwacja

Baza danych Oracle ma zdolność prowadzenia obserwacji wszystkich działań, które w niej następują. Rekordy obserwacji mogą być zapisywane do tabeli SYS.AUD$ albo mogą tworzyć zapis obserwacji w systemie operacyjnym. Możliwość wykonywania zapisu obserwacji w systemie operacyjnym zależy od danego systemu operacyjnego.

Obserwacji mogą podlegać trzy różne rodzaje działań: próby logowania, uzyskiwanie dostępu do obiektów oraz operacje przeprowadzane na obiektach bazy danych. W kolejnych podrozdziałach omówiono sposoby śledzenia wszystkich wymienionych działań. Podczas prowadzenia obserwacji domyślnym sposobem pracy bazy danych jest rejestrowanie zarówno udanych, jak i nieudanych poleceń. Istnieje możliwość konfigurowania każdego typu prowadzonej obserwacji.

Warunkiem uruchomienia obserwacji w bazie danych jest wprowadzenie wpisu dla parametru AUDIT_TRAIL w pliku *init.ora* bazy danych. Możliwe są następujące wartości parametru AUDIT_TRAIL:

- NONE — wyłączenie obserwacji;
- DB — włączenie obserwacji i dokonywanie zapisów do tabeli SYS.AUD$;
- OS — włączenie obserwacji i tworzenie zapisu obserwacji na poziomie systemu operacyjnego (zależnego od rodzaju systemu operacyjnego).

Polecenia **audit** opisywane w następnych podrozdziałach mogą być wydawane bez względu na ustawienie parametru AUDIT_TRAIL. Należy jednak pamiętać, że polecenia te pozostaną nieaktywne aż do uruchomienia bazy danych, następującego po ustawieniu odpowiedniej wartości parametru AUDIT_TRAIL pliku *init.ora*.

W razie dokonania wyboru opcji składowania rekordów obserwacji w tabeli SYS.AUD$ należy pamiętać o okresowej archiwizacji rekordów tej tabeli a następnie o jej obcinaniu za pomocą polecenia **truncate**. Tabela SYS.AUD$ znajduje się w słowniku danych, a zatem w przestrzeni tabel SYSTEM. Brak okresowego usuwania rekordów tej tabeli może powodować problemy związane z przestrzenią dyskową. W celu umożliwienia użytkownikowi usuwania rekordów z tabeli SYS.AUD$ można przyznać mu rolę usuwania katalogu DELETE_CATALOG_ROLE.

Obserwacja logowania

Istnieje możliwość obserwacji każdej próby połączenia się z bazą danych. Polecenie rozpoczęcia obserwacji prób logowania podano poniżej:

```
audit session;
```

Można dokonać takich ustawień, aby śledzone były tylko próby połączeń zakończone powodzeniem (pierwszy wiersz poniższego przykładu) albo niepowodzeniem (drugi wiersz). Poniżej przedstawiono sposób wydawania odpowiednich poleceń:

```
audit session whenever successful;
audit session whenever not successful;
```

Jeżeli rekordy obserwacji są składowane w tabeli SYS.AUD$, można je przeglądać za pomocą perspektywy słownika danych DBA_AUDIT_SESSION tej tabeli.

Poniżej przedstawiono przykładowe zapytanie służące do wyszukania rekordów obserwacji logowania z perspektywy DBA_AUDIT_SESSION. W ten sposób otrzymuje się informacje dotyczące wykorzystanego konta systemu operacyjnego (OS_Username), nazwę konta bazy danych Oracle (Username) oraz identyfikator ID wykorzystywanego terminala (Terminal). Przeglądana jest kolumna Returncode. Wartość 0 tej kolumny oznacza próbę logowania zakończoną powodzeniem, w przeciwnym razie sprawdzane są dwa najczęściej występujące numery błędów w celu określenia przyczyny niepowodzenia. Wyświetlony jest również czas zalogowania i wylogowania z danego konta.

```
select
  OS_Username,       /*Nazwa użytkownika systemu operacyjnego.*/
  Username,          /*Nazwa użytkownika konta bazy danych Oracle.*/
  Terminal,          /*Identyfikator użytego terminala.*/
  DECODE(Returncode,'0','Connected',
            '1005','FailedNull',
            '1017','Failed',Returncode),/*Sprawdzenie niepowodzenia*/
  TO_CHAR(Timestamp,'DD-MON-YY HH24:MI:SS'),    /*Czas zalogowania*/
  TO_CHAR(Logoff_Time,'DD-MON-YY HH24:MI:SS')   /*Czas wylogowania */
from DBA_AUDIT_SESSION;
```

Sprawdzanymi numerami błędów są ORA-1005 oraz ORA-1017. Te dwa kody oznaczają najczęściej występujące błędy logowania. Kod ORA-1005 jest zwracany, jeśli użytkownik wprowadzi nazwę użytkownika bez podania hasła. Zwrócenie kodu ORA-1017 następuje po wprowadzeniu nieprawidłowego hasła.

Aby wyłączyć śledzenie sesji, należy wydać polecenie **noaudit**, jak pokazano poniżej:

```
noaudit session;
```

Obserwacja działań

Istnieje możliwość obserwacji każdego działania mającego wpływ na poszczególne obiekty bazy danych, takie jak tabela, powiązanie baz danych, przestrzeń tabel, synonim, segment wycofania, użytkownik lub indeks. Obserwowane działania, wpływające na stan poszczególnych obiektów, takie jak tworzenie (polecenie **create**), zmiana (polecenie **alter**) lub usuwanie (polecenie **drop**) mogą być grupowane. Takie zgrupowania poleceń ułatwiają ustalenie i zachowanie ustawień prowadzonych obserwacji.

Wszystkie polecenia przeprowadzane na poziomie systemu mogą być obserwowane, a grupy poleceń są określone z góry. Na przykład, aby obserwować wszystkie polecenia dotyczące ról na poziomie systemu, należy wprowadzić zapis:

```
audit role;
```

Aby wyłączyć to ustawienie, należy wprowadzić polecenie:

```
noaudit role;
```

Grupy poleceń SQL służące do ustawień obserwacji wyszczególniono w dodatku A., w opisie instrukcji AUDIT Każda z tych grup może służyć w celu śledzenia wszystkich poleceń SQL, które jej dotyczą. W tabeli 10.2 znajduje się szczegółowy wykaz odpowiadających im uprawnień. Na przykład, przedstawione wyżej ustawienie **audit role** spowoduje obserwację wydawanych poleceń **create role**, **alter role**, **drop role** oraz **set role**.

Oprócz podstawowych opcji śledzenia przedstawionych w dodatku A, można prowadzić obserwację stosowania każdego szczególnego polecenia (takiego jak **create table**). Baza danych Oracle dostarcza również następujących grup opcji instrukcji:

- CONNECT — obserwuje operacje zalogowania i wylogowania do i z bazy danych Oracle;
- DBA — obserwuje polecenia, które wymagają identyfikacji administratora bazy danych, takich jak **grant**, **revoke**, **audit**, **noaudit**, **create** oraz **alter tablespace** i **create** lub **drop public synonym**;
- RESOURCE — obserwuje polecenia **create** i **drop** dla tabel, klastrów, perspektyw, indeksów, przestrzeni tabel, typów i synonimów;
- ALL — obserwuje wszystkie polecenia;
- ALL PRIVILEGES — wszystkie poprzednie polecenia oraz dodatkowo **delete**, **insert**, **update** oraz kilka innych poleceń (patrz dodatek A).

Do każdego obserwowanego działania jest przypisany kod liczbowy w bazie danych. Te kody można przeglądać za pomocą perspektywy AUDIT_ACTIONS. Poniżej przedstawiono zapytanie, efektem którego jest wyświetlenie dostępnych kodów działań dla bazy danych:

```
select
      Action,       /*Kod działania.*/
      Name          /*Nazwa działania taka jak ALTER USER.*/
from AUDIT_ACTIONS;
```

Jeśli kod działania jest znany, można wykorzystać perspektywę DBA_AUDIT_OBJECT w celu określenia wpływu tego działania na dany obiekt. Poniżej przedstawiono przykładowe zapytanie wyszukujące rekordy obserwacji logowania z perspektywy DBA_AUDIT_OBJECT. W ten sposób otrzymuje się informacje dotyczące zastosowanego konta systemu operacyjnego (OS_Username), nazwę konta bazy danych Oracle (Username) oraz identyfikator ID wykorzystywanego terminala (Terminal). Wybrano dane dotyczące właściciela obiektu (Owner) oraz nazwy obiektu (Obj_Name) wraz z kodem działania (Action_Name) dla wykonanej czynności. Przeglądana jest kolumna Returncode: wartość tej kolumny równa 0 oznacza próbę połączenia zakończoną powodzeniem, w przeciwnym razie podawany jest numer błędu. Wyświetlony jest również czas zalogowania i wylogowania.

```
select
  OS_Username,          /*Nazwa użytkownika systemu operacyjnego.*/
  Username,             /*Nazwa konta użytkownika bazy danych Oracle.*/
  Terminal,             /*Identyfikator ID wykorzystanego terminala.*/
  Owner,                /*Właściciel obiektu oddziaływania.*/
  Obj_Name,             /*Nazwa obiektu oddziaływania.*/
  Action_Name,          /*Kod liczbowy działania.*/
  DECODE(Returncode,'0','Success',Returncode),/*Sprawdzenie
                                                niepowodzenia*/
  TO_CHAR(Timestamp,'DD-MON-YY HH24:MI:SS')   /*Znacznik czasu*/
from DBA_AUDIT_OBJECT;
```

Istnieje również możliwość ustawienia obserwacji poszczególnych użytkowników. W tym celu wykorzystuje się klauzulę **by username** polecenia **audit**. Poniżej zamieszczono odpowiedni przykład. Efektem tego polecenia jest śledzenie wszystkich działań modyfikacji (polecenia **update**) dokonywanych przez użytkownika MCGREGOR.

```
audit update table by MCGREGOR;
```

Obserwacja obiektów

Istnieje także możliwość prowadzenia obserwacji działań manipulowania danymi. W tym przypadku śledzi się operacje na tabelach wykonywane za pomocą poleceń: **select**, **insert**, **update** oraz **delete**. Działania tego typu są obserwowane w bardzo podobny sposób, jak te opisane w poprzednim podrozdziale. Jedyną różnicą jest dodanie nowej klauzuli w poleceniu **audit**.

Obserwacja sesji lub dostępu

W celu prowadzenia obserwacji obiektów jest stosowana klauzula **by session** lub klauzula **by access**. Określają one, czy rekord obserwacji powinien być zapisany jeden raz dla każdej sesji (klauzula **by session**) lub jeden raz dla każdego uzyskania dostępu do obiektu (klauzula **by access**). Na przykład, jeżeli podczas jednej sesji użytkownik wykonał cztery różne instrukcje modyfikacji **update** na tej samej tabeli, wtedy wynikiem obserwacji z klauzulą **by access** byłoby zapisanie czterech rekordów obserwacji — jeden zapis dla każdego dostępu tabeli. Z drugiej strony obserwacja tej samej sytuacji z klauzulą **by session** dałaby w rezultacie zapis tylko jednego rekordu obserwacji.

Z powyższego wynika, że prowadzenie obserwacji z klauzulą **by access** może radykalnie zwiększyć szybkość zapisywania rekordów obserwacji. Ten typ obserwacji jest generalnie wykorzystywany w ograniczonym zakresie w celu pomiaru liczby konkretnych działań zachodzących podczas określonego czasu. Po zakończeniu takiego testowania należy ponownie zmienić tryb obserwacji na status **by session**.

Odpowiednie przykłady wykorzystania omówionych wyżej opcji pokazano poniżej. Dzięki pierwszemu zapisowi obserwowane są wszystkie polecenia wstawiania (**insert**) wykonywane na tabeli EMPLOYEE. Drugi wpis powoduje obserwację każdego działania dotyczącego tabeli TIME_CARDS. Trzecie polecenie powoduje obserwację wszystkich operacji usuwania (**delete**) wykonywanych na tabeli DEPARTMENT. W tym przypadku zastosowano klauzulę **by session**.

```
audit insert on THUMPER.EMPLOYEE;
audit all on THUMPER.TIME_CARDS;
audit delete on THUMPER.DEPARTMENT by session;
```

Otrzymane rekordy obserwacji mogą być przeglądane za pomocą zapytania na perspektywie DBA_AUDIT_OBJECT. Przykład takiego zapytania znajduje się w poprzednim podrozdziale.

Precyzyjna obserwacja obiektów

Jedyną wadą obserwacji obiektów jest to, że mimo iż można zorientować się, jaki obiekt był zmieniany i kto uzyskał do niego dostęp, brak jednak możliwości sprawdzenia, jakie dane zostały zmienione i jakie były ich poprzednie wartości. Oracle9i wprowadza pakiet PL/SQL, który umożliwia precyzyjną obserwację obiektów, pomocną w śledzeniu uzyskiwania dostępu do informacji w bazie danych oraz sposobu, w jaki były one zmieniane.

Aby włączyć opcję precyzyjnej obserwacji obiektów, należy utworzyć predykat SQL, który będzie opisywał warunki, dla jakich rekord obserwacji powinien zostać zapisany w dzienniku. Predykat SQL definiuje warunki dostępu do danych, które uruchomią obserwację obiektu. Aby tego dokonać i zarządzić precyzyjną obserwację, należy użyć pakietu PL/SQL DBMS_FGA. Działania, jakie można dzięki niemu podjąć, są następujące:

- add_policy — dodaje strategię precyzyjnej obserwacji tabeli lub perspektywy;
- drop_policy — usuwa strategię precyzyjnej obserwacji tabeli lub perspektywy;
- enable_policy — włącza politykę bezpieczeństwa dla tabeli lub perspektywy;
- disable_policy — wyłącza politykę bezpieczeństwa dla tabeli lub perspektywy

Dla przykładu przyjęto, że farmer McGregor chce obserwować każdą osobę, która sprawdza aktualną liczbę marchwi (CARROTS) w magazynie. Musi zatem utworzyć procedurę, definiującą sposób, w jaki ma być powiadamiany o tym, że ktokolwiek wykonuje zapytanie na tabeli PRODUCE a następnie utworzyć odpowiednią politykę, aby ustanowić kryteria obserwacji.

```
/* dodaj politykę */
exec DBMS_FGA.ADD_POLICY( -
obcject_schema  => 'FM', -
object_name     => 'PRODUCE', -
policy_name     => CHK_CARROT_COUNT, -
audit_condition => 'VEGETABLE = '' CARROTS''   ', -
adit_column     => 'QUANTITY' ,  -
handler_schema  => 'SEC' ,  -
handler_module  => 'LOG_ACTION' ,  -
enable          => TRUE) ;
```

W opisywanym przykładzie politykę obserwacji stanowi przechwytywanie informacji o każdym użytkowniku, który wybiera wartość powiązaną z wpisem CARROTS w tabeli PRODUCE. Należy także ustawić procedurę obsługi błędu, taką jak wyzwalacz, aktywowaną, gdy spełnione będą warunki polityki. W tym przypadku będzie używany wyzwalacz LOG_ACTION w schemacie SEC.

Ochrona zapisu obserwacji

Ze względu na składowanie tabeli zapisów obserwacji bazy danych SYS.AUD$ w bazie danych, wszystkie zapisywane tu rekordy obserwacji muszą być chronione. W przeciwnym razie użytkownik może dokonać próby usunięcia odpowiednich rekordów zapisów obserwacji po ewentualnym przeprowadzeniu nieautoryzowanych działań w bazie danych.

Istnieje możliwość rozwiązania tego problemu przez wprowadzenie mechanizmu zapisu rekordów obserwacji na poziomie systemu operacyjnego. Umożliwia to zachowanie rekordów poza bazą danych. Jednak ta opcja nie jest dostępna w przypadku wszystkich systemów operacyjnych.

Jeżeli zachodzi konieczność składowania informacji dotyczących zapisów obserwacji w tabeli SYS.AUD$, *jest wymagana* ochrona tej tabeli. Pierwszym posunięciem jest obserwacja działań prowadzonych na tej tabeli za pomocą następującego polecenia:

```
audit all on SYS.AUD$ by access;
```

Wszystkie działania wykonywane na tabeli SYS.AUD$ (z wyjątkiem wstawień generowanych przez obserwacje innych tabel) są teraz rejestrowane w zapisie obserwacji. Wpisy w tabeli SYS.AUD$ mogą być usunięte tylko przez tych użytkowników, którzy mają możliwość połączenia wewnętrznego przez opcję CONNECT *<nazwa użytkownika/hasło>* AS SYSDBA (zatem są członkami grupy *DBA*). Każde działanie wykonane podczas połączenia AS SYSDBA jest automatycznie wprowadzane do zapisu obserwacji.

Zaleca się prowadzenie skoordynowanej obserwacji bazy danych i systemu operacyjnego. Ułatwia to śledzenie ewentualnych problemów i zapewnia skuteczniejszą ochronę całej aplikacji. Najczęściej osoby zarządzające systemami nie życzą sobie większej liczby zapisów obserwacji, a zatem administrator bazy danych jest zmuszony do przeprowadzenia dokładnej analizy sposobu prowadzenia obserwacji. Należy dążyć do sytuacji, w której każdy rekord zapisu obserwacji jest istotny. Zastosowanie poleceń omówionych w niniejszym rozdziale umożliwia taką modyfikację opcji obserwacji działania bazy, aby śledzone były najistotniejsze punkty pracującej aplikacji.

Zabezpieczenie w środowisku rozproszonym

Otwieranie bazy danych przez inne serwery w celu uzyskania do niej dostępu powoduje powstanie potencjalnych zagrożeń bezpieczeństwa ze strony tych serwerów. Ze względu na to, że dostęp ten jest realizowany za pomocą programu *Oracle Net*, odpowiednia modyfikacja parametrów tego programu może zapewnić większość zabezpieczeń przeciwko nieautoryzowanemu zdalnemu dostępowi. Szczegółowe informacje na temat aspektów bezpieczeństwa programu *Oracle Net* znajdują się w części III niniejszej książki. Przewodnią zasadą powinno być, aby dostęp do danych odbywał się na zasadzie *trzeba wiedzieć* (aby dostać się do jakiegoś elementu bazy, trzeba wiedzieć jak to zrobić, znać hasło itd.). Rozszerzając tę zasadę, uzyskiwanie jakiegokolwiek dostępu do serwerów i systemu operacyjnego powinno się odbywać na zasadzie *trzeba wiedzieć*.

Należy okresowo przeglądać bieżące uprawnienia związane z dostępem do bazy danych oraz na poziomie systemu operacyjnego. Konieczna jest współpraca z zespołami zarządzania systemami w celu oceny bieżących uprawnień dostępu sieciowego.

Rozwiązania

Efektywne zarządzanie bezpieczeństwem w bazie danych Oracle wymaga rozwiązania wszystkich zagadnień zabezpieczeń oraz prowadzenia obserwacji prób naruszenia zabezpieczeń. Plan zabezpieczenia powinien zawierać przynajmniej następujące elementy:

1. Zmiana domyślnych haseł kont `SYS` oraz `SYSTEM`.
2. Regularna zmiana haseł dla wszystkich kont z uprawnieniami *DBA*.
3. Usunięcie utworzonych kont demonstracyjnych (jak `SCOTT/TIGER`).
4. Zmiana hasła konta `DBSNMP` i umieszczenie nowego hasła w pliku *snmp.ora*.
5. Ustawienie odpowiednich poziomów ochrony dla wszystkich plików bazy danych.
6. Jeżeli opracowywana baza danych zawiera dane z produkcyjnej bazy danych, należy upewnić się, że reguły bezpieczeństwa stosowane dla produkcyjnej bazie danych są również zastosowane dla opracowywanej bazy danych.
7. Obserwacja wszystkich prób uzyskania dostępu do tabeli `SYS.AUD$`.
8. Obserwacja wszystkich nieudanych prób połączenia.
9. Obserwacja wszystkich działań administratora bazy danych.
10. Regularne generowanie raportów wykazywania obserwacji oraz usuwanie nieaktualnych rekordów z tabeli `SYS.AUD$`.
11. Zabezpieczenie kopii zapasowych bazy danych.
12. Zabezpieczenie pomieszczeń, w których znajdują się serwer bazy danych oraz kopie zapasowe.

Przestrzeganie wszystkich zasad wyszczególnionych powyżej umożliwi zabezpieczenie bazy danych. Jak wspomniano wcześniej w tym rozdziale, w dalszym ciągu jest konieczna współpraca administratora bazy danych z zespołami zarządzania systemem oraz zarządzania siecią w celu wyeliminowania uzyskiwania nieautoryzowanego dostępu do systemu operacyjnego serwera.