Wykład 9

Implementacja języka SQL w systemach baz danych Oracle – tworzenie, modyfikowanie i usuwanie obiektów bazy danych: tabel i perspektyw, więzów integralności, komentarzy (DDL), manipulowanie danymi (DML), elementy języka sterowania transakcjami (TCL).

I. Język definiowania, modyfikowania i usuwania obiektów bazy - DDL (Data Definition Language).

I. 1 Uwagi ogólne dotyczące nazw tabel, perspektyw i ich kolumn.

- Nazwa tabeli lub perspektywy musi być unikalna wśród nazw obiektów dostępnych dla danego użytkownika

- Nazwa tabeli, perspektywy czy kolumny:

  - może mieć maksymalnie 30 znaków,
  - musi zaczynać się od litery,
  - może zawierać litery, cyfry, a także _, $, #
    (chociaż $ i # nie są zalecane)
  - jeżeli jest taka jak słowo kluczowe PL/SQL (co nie jest zalecane), to musi być ujmowana w cudzysłowy
  - jeżeli w definicji tabeli lub perspektywy nazwę tabeli lub perspektywy czy też kolumny napiszemy w cudzysłowach (***"emp"***), to duże i małe litery w takiej nazwie nie będą utożsamiane i używając tej nazwy w poleceniach zawsze będziemy musieli ujmować ją w cudzysłów.

I. 2 Tworzenie tabel - polecenie CREATE TABLE

Ograniczymy się do tworzenia tabel w schemacie bieżącego użytkownika:

Podstawowe wersje polecenia CREATE TABLE:

- Tworzenie typowej (nie tymczasowej) tabeli relacyjnej poprzez zdefiniowanie poszczególnych jej kolumn:

```
CREATE TABLE nazwa_tabeli
(
   nazwa_kolumny [typ_danych ][DEFAULT wartość lub wyrażenie ] [więz_kolumnowy][ ,
   nazwa_kolumny [typ_danych ][DEFAULT wartość lub wyrażenie ] [więz_kolumnowy],
   ... ,                                                                        ]
  [więzy_tabelowe]
)
 [możliwe inne własności]  ← np. określenie przestrzeni tabel dla definiowanej tabeli
;
```

Przykład:

```
create table PRACOWNICY
(
  ID_P number(6) not null,
  IMIĘ varchar2(30),
  NAZWISKO varchar2(50),
  PENSJA number(8,2),
  ID_BP number(6),
  ID_D number(3)
);
```

- Tworzenie typowej (nie tymczasowej) tabeli relacyjnej na podstawie już istniejącej tabeli:

```
CREATE TABLE nazwa_tabeli
[ ( nazwa kolumny [ NULL / NOT NULL ] , ... ) ]
AS zapytanie_do_istniejącej_tabeli;
```

Jeżeli nie podamy nazw kolumn - te, które są wymienione w *zapytaniu* zostaną utworzone w tabeli.

```
create table PRACOWNICY
as select * from emp;
```

Jeżeli podamy nazwy kolumn - *zapytanie* zostanie użyty tylko do wstawienia wartości do nowo utworzonej tabeli.

```
create table PRACOWNICY
(ID_P, NAZWISKO, PENSJA, ID_BP, ID_D)
as select empno, ename, sal, mgr, deptno from emp;
```

I. 3 Zmiana struktury tabeli - polecenie ALTER TABLE

Wybrane zastosowania polecenia ALTER TABLE:

- dodawanie nowych kolumn lub więzów do tabeli

```
ALTER TABLE [schemat_użytkownika.]tabela
ADD (nazwa_kolumny [typ_danych]
      [DEFAULT wartość lub wyrażenie] [więz_kolumnowy] [,
      … ,                                                ]
      więz_tabelowy);
```

Przykład:

```
alter table PRACOWNICY
add (MAKS_PENSJA_W_DZIALE number(8,2) DEFAULT 0.0
     NOT NULL);
```

- dodawanie nowych więzów do tabeli

```
ALTER TABLE [schemat_użytkownika.]tabela
ADD więz_tabelowy;

ALTER TABLE [schemat_użytkownika.]tabela
ADD (więz_tabelowy1, więz_tabelowy2, … );
```

- zmienianie (modyfikowanie) definicji istniejących kolumn w tabeli

```
ALTER TABLE [schemat_użytkownika.]tabela
MODIFY (kolumna [typ_danych]
            [DEFAULT wartość lub wyrażenie] [więz_kolumnowy] [,
            …                                                ]
            );
```

Przykład:

```
alter table PRACOWNICY
modify (NAZWISKO varchar2(60) NOT NULL,
          IMIĘ varchar2(40) NOT NULL);
```

Uwaga: w starszych wersjach Oracle'a (np. Oracle 8i) jedynymi więzami kolumnowymi, które można było zmienić w podklauzuli MODIFY polecenia ALTER TABLE były własności NOT NULL i NULL.
W Oracle 10g nie ma takich ograniczeń, tzn. można narzucić na kolumnę dowolny rodzaj więzu za pomocą podklauzuli MODIFY w poleceniu ALTER TABLE.

Ponadto możliwe jest tworzenie więzów tabelowych za pomocą podklauzuli MODIFY:

```
ALTER TABLE [schemat_użytkownika.]tabela
MODIFY więz_tabelowy;

ALTER TABLE [schemat_użytkownika.]tabela
MODIFY (więz_tabelowy1, więz_tabelowy2, … );
```

Uwaga:
Gdy więz istnieje, to najczęściej należy go najpierw usunąć (patrz koniec podrozdziału I.5), a później można go utworzyć ponownie w zmienionej formie!

Jednak podklauzula MODIFY pozwala także na pewne modyfikacje własności więzów. Poprzez tę klauzulę można zmieniać stan więzu (przykładowo włączony (ENABLE) albo wyłączony (DISABLE)):

```
ALTER TABLE [schemat_użytkownika.]tabela
MODIFY więz stan_więzu;
```

Wybrane stany więzu: ENABLE - włączony (domyślny),
DISABLE - wyłączony.

W przypadku każdego więzu można użyć składni:

```
ALTER TABLE [schemat_użytkownika.]tabela
MODIFY CONSTRAINT Nazwa_własna_więzu ENABLE | DISABLE;
```

Dodatkowo dla więzów typu:

- klucz główny - poprawne jest polecenie:

```
ALTER TABLE [schemat_użytkownika.]tabela
MODIFY PRIMARY KEY ENABLE | DISABLE;
```

- klucz unikalny - poprawne jest polecenie:

```
ALTER TABLE [schemat_użytkownika.]tabela
MODIFY UNIQUE (nazwa_kolumny1, nazwa_kolumny2, …)
ENABLE | DISABLE;
```

- usuwanie kolumny z tabeli

Usuwanie jednej kolumny tabeli:

```
ALTER TABLE [schemat_użytkownika.]tabela
DROP COLUMN kolumna
[CASCADE CONSTRAINTS | INVALIDATE];
```

Usunięcie kolumny, która nie jest kolumną klucza głównego lub unikalnego, do którego odnosi się jakiś klucz obcy:

```
alter table PRACOWNICY
drop column ID_P;
```

Powyższa komenda jest równoważna poleceniu:

```
alter table PRACOWNICY
drop column ID_P invalidate;
```

Usunięcie kolumny, powodujące usunięcie więzów ewentualnie odnoszących się do niej:

```
alter table PRACOWNICY
drop column ID_P cascade constraints;
```

Po wykonaniu powyższego polecenia wraz z usunięciem kolumny ID_P, która jest kolumną klucza głównego w tabeli PRACOWNICY, zostanie usunięty każdy więz typu klucz obcy (ale nie kolumna, na którą ten klucz obcy jest nałożony!), odnoszący się do tego klucza głównego.

Usuwanie wielu kolumn tabeli:

```
ALTER TABLE [schemat_użytkownika.]tabela
DROP (kolumna1, kolumna2, …)
[CASCADE CONSTRAINTS | INVALIDATE];
```

Przykład:

alter table PRACOWNICY
drop (ID_D, PROWIZJA);

Uwagi podsumowujące:

Parametr INVALIDATE (*ang.* unieważnić) jest domyślny i oznacza, że np. kolumna klucza głównego lub unikalnego, jeśli odnosi się do niej jakiś klucz obcy, nie zostanie usunięta.
Natomiast parametr CASCADE CONSTRAINTS zezwala na usunięcie takich kolumn klucza głównego wraz z wszystkimi więzami odnoszącymi się do niego.

I. 4 Usuwanie tabeli - polecenie DROP TABLE

Usuwanie tabeli, do której nie odnoszą się żadne klucze obce:

DROP TABLE [schemat_użytkownika.]tabela;

Usuwanie tabeli wraz z więzami do niej odnoszącymi się (ale bez usuwania kolumn, na które te więzy odnoszące się do tabeli usuwanej są narzucone):

DROP TABLE [schemat_użytkownika.]tabela
CASCADE CONSTRAINTS;

Usuwanie tabeli, do której nie odnoszą się żadne klucze obce, bez możliwości jej odzyskanie:

DROP TABLE [schemat_użytkownika.]tabela PURGE;

Usuwanie tabeli wraz z więzami do niej odnoszącymi się (ale bez usuwania kolumn, na które te więzy odnoszące się do tabeli usuwanej są narzucone) bez możliwości jej odzyskanie:

DROP TABLE [schemat_użytkownika.]tabela
CASCADE CONSTRAINTS PURGE;

I. 5 Więzy (CONSTRAINTS)

Klucz główny (w sensie własności tego więzu po jego zdefiniowaniu na serwerze bazy danych) to zestaw atrybutów (kolumn), których wartości są unikalne w każdej krotce (wierszu) relacji bazodanowej (tabeli), innymi słowy poprzez które można w jednoznaczny sposób zidentyfikować każdy wiersz tabeli. Tabela może posiadać co najwyżej jeden klucz główny. Klucz główny w bazach Oracle domyślnie otrzymuje własność NOT NULL.

Klucz unikalny ma podobne własności jak klucz główny. Różni się od niego tym, że:

- może przyjąć wartość NULL, chociaż nie jest to zalecane; warto przy tym pamiętać, że przykładowo, gdy klucz unikalny jest jednoatrybutowy to wartość NULL może być przyjęta przez niego tylko w jednym wierszu;
- może być więcej kluczy unikalnych w danej tabeli niż jeden.

Klucz obcy to taki zestaw atrybutów, który odnosi się do jakiegoś klucza głównego lub unikalnego i nie zezwala na wprowadzenie do swoich atrybutów innych wartości niż takie, które wcześniej zostały wprowadzone do klucza odniesienia. Wyjątkiem jest wartość pusta NULL, którą klucz obcy może przyjmować wielokrotnie niezależnie czy odnosi się do klucza głównego czy unikalnego.

Uwaga:

Klucz obcy można zdefiniować, gdy istnieje klucz główny lub unikalny, do którego ma się odnosić ten klucz obcy!

Typy więzów	Słowa kluczowe SQL
klucz główny	PRIMARY KEY
kłucz unikalny	UNIQUE
klucz obcy	FOREIGN KEY … REFERENCES
więzy NOT NULL / NULL	NOT NULL albo domyślnie NULL
więzy typu CHECK	CHECK

Dodatkowe informacje dotyczące definiowania kluczy obcych:

Definiując klucz obcy, możemy na końcu definicji dodać opcjonalną klauzulę ON DELETE, która określa, w jaki sposób traktowane będą wartości klucza obcego w przypadku usunięcia odpowiadającej wartości klucza odniesienia, tj. klucza głównego lub klucza unikalnego. Możliwe są dwie opcje tej klauzuli:

ON DELETE CASCADE – po usunięciu wiersza tabeli nadrzędnej (w której zdefiniowano klucz odniesienia) zostaną usunięte wszystkie wiersze tabeli podrzędnej (w której zdefiniowano odnoszący się do niego klucz obcy) z wartością klucza obcego odpowiadającą wartości klucza odniesienia z usuniętego wiersza tabeli nadrzędnej.

ON DELETE SET NULL – po usunięciu wiersza tabeli nadrzędnej (w której zdefiniowano klucz odniesienia) we wszystkich wierszach tabeli podrzędnej (w której zdefiniowano odnoszący się do niego klucz obcy) zostanie zmieniona na NULL wartość klucza obcego, odpowiadająca wartości klucza odniesienia z usuniętego wiersza tabeli nadrzędnej.

Gdy nie zastosujemy klauzuli ON DELETE w definicji klucza obcego, to nie zostanie wykonana żadna akcja na wierszach tabeli podrzędnej (w której zdefiniowano klucz obcy) w przypadku usunięcia wiersza tabeli nadrzędnej (w której zdefiniowano klucz odniesienia tego klucza obcego).

Wszystkie rodzaje więzów można definiować zarówno poprzez polecenie CREATE TABLE, jak i ALTER TABLE.

Ogólny schemat definicji więzu jest następujący:

[CONSTRAINT Nazwa_własna_więzu]
SŁOWO_KLUCZOWE_TYPU_WIĘZU [elementy składni wynikające z typu więzu]

Przykład definiowania więzów kolumnowych poprzez polecenie CREATE TABLE, których nazwy własne są nadawane automatycznie przez serwer:

```
create table PRACOWNICY
(
  ID_P number(6) primary key,
  IMIĘ varchar2(30) not null,
  NAZWISKO varchar2(50) not null,
  PENSJA number(8,2) check (PENSJA >= 0),
  ID_BP [ number(6) ] references PRACOWNICY,
  ID_D [ number(3) ] references DZIAŁY
);
```

Przykład definiowania więzów kolumnowych poprzez polecenie CREATE TABLE, których nazwy własne są nadawane przez użytkownika:

```
create table PRACOWNICY
(
  ID_P number(6) constraint PK_pracownicy primary key,
  IMIĘ varchar2(30) constraint NN1_pracownicy not null,
  NAZWISKO varchar2(50) constraint NN2_pracownicy not null,
  PENSJA number(8,2) constraint C1_pracownicy check (PENSJA >= 0),
  ID_BP [ number(6) ] constraint RFK_pracownicy references PRACOWNICY,
  ID_D [ number(3) ] constraint FK_pracownicy_DZIAŁY references DZIAŁY
);
```

Przykład definiowania więzów tabelowych oraz kolumnowych NOT NULL poprzez polecenie CREATE TABLE, których nazwy własne są nadawane automatycznie przez serwer:

```
create table PRACOWNICY
(
  ID_P number(6),
  IMIĘ varchar2(30) not null,
  NAZWISKO varchar2(50) not null,
  PENSJA number(8,2),
  ID_BP [ number(6) ],
  ID_D [number(3) ],
  primary key (ID_P),
  foreign key (ID_BP) references PRACOWNICY,
  foreign key (ID_D) references DZIAŁY,
  check (PENSJA >= 0)
);
```

Przykład definiowania więzów tabelowych oraz kolumnowych NOT NULL poprzez polecenie CREATE TABLE, których nazwy własne są nadawane przez użytkownika:

```
create table PRACOWNICY
(
  ID_P number(6),
  IMIĘ varchar2(30) constraint NN1_pracownicy not null,
  NAZWISKO varchar2(50) constraint NN2_pracownicy not null,
  PENSJA number(8,2),
  ID_BP [ number(6) ],
  ID_D [ number(3) ],
  constraint PK_pracownicy primary key (ID_P),
  constraint RFK_pracownicy foreign key (ID_BP) references PRACOWNICY,
  constraint FK_pracownicy foreign key (ID_D) references DZIAŁY,
  constraint C1_pracownicy check (PENSJA >= 0)
);
```

Zastosowanie polecenia ALTER TABLE do definiowania więzów:

Zdefiniujmy tabelę PRACOWNICY bez jakiegokolwiek więzu:

```
create table PRACOWNICY
(
  ID_P number(6),
  IMIĘ varchar2(30),
  NAZWISKO varchar2(50),
  PENSJA number(8,2),
  ID_BP number(6),
  ID_D number(3)
);
```

Wówczas można utworzyć więzy w tabeli PRACOWNICY, używając polecenia ALTER TABLE:

Przykład definiowania więzów tabelowych (opartych na istniejących kolumnach) poprzez polecenie ALTER TABLE z klauzulą ADD, których nazwy własne są nadawane automatycznie przez serwer:

```
alter table PRACOWNICY
add (primary key (ID_P),
      foreign key (ID_BP) references PRACOWNICY,
      foreign key (ID_D) references DZIAŁY,
      check (PENSJA >= 0)
      );
```

Przykład definiowania więzów tabelowych (opartych na istniejących kolumnach) poprzez polecenie ALTER TABLE z klauzulą ADD, których nazwy własne są nadawane przez użytkownika:

```
alter table PRACOWNICY
add (constraint PK_pracownicy primary key (ID_P),
    constraint RFK_pracownicy foreign key (ID_BP) references PRACOWNICY,
    constraint FK_pracownicy foreign key (ID_D) references DZIAŁY,
    constraint C_pracownicy check (PENSJA >= 0)
    );
```

Uwaga:
W powyższy sposób nie można zdefiniować więzów NOT NULL i NULL, które jako jedyne są restrykcyjnie traktowane jako więzy kolumnowe!

Przykład definiowania więzów kolumnowych NOT NULL (narzucanych na istniejące kolumny) poprzez polecenie ALTER TABLE z klauzulą MODIFY, których nazwy własne są nadawane automatycznie przez serwer:

```
alter table PRACOWNICY
modify (IMIĘ not null, NAZWISKO not null);
```

Przykład definiowania więzów kolumnowych NOT NULL (narzucanych na istniejące kolumny) poprzez polecenie ALTER TABLE z klauzulą MODIFY, których nazwy własne są nadawane przez użytkownika:

```
alter table PRACOWNICY
modify (constraint NN1_pracownicy IMIĘ not null,
         constraint NN1_pracownicy NAZWISKO not null
        );
```

Przykład definiowania więzów (opartych na istniejących kolumnach) poprzez polecenie ALTER TABLE z klauzulą MODIFY, których nazwy własne są nadawane automatycznie przez serwer:

- składnia kolumnowa:

```
alter table PRACOWNICY
modify (ID_P primary key,
         IMIĘ not null, NAZWISKO not null,
         ID_BP references PRACOWNICY,
         ID_D references DZIAŁY,
         PENSJA check (PENSJA >= 0)
        );
```

- składnia tabelowa:

```
alter table PRACOWNICY
modify (primary key (ID_P),
         IMIĘ not null, NAZWISKO not null,
         foreign key (ID_BP) references PRACOWNICY,
         foreign key (ID_D) references DZIAŁY,
         check (PENSJA >= 0)
        );
```

Przykład definiowania więzów (opartych na istniejących kolumnach) poprzez polecenie ALTER TABLE z klauzulą MODIFY, których nazwy własne są nadawane przez użytkownika:

- składnia kolumnowa:

```
alter table PRACOWNICY
modify (ID_P constraint PK_pracownicy primary key,
          IMIĘ constraint NN1_pracownicy not null,
          NAZWISKO constraint NN2_pracownicy not null,
          ID_BP constraint RFK_pracownicy references PRACOWNICY,
          ID_D constraint FK_pracownicy references DZIAŁY,
          PENSJA constraint C_pracownicy check (PENSJA >= 0)
         );
```

- składnia tabelowa (wymagana kolumnowa tylko dla więzów NOT NULL):

```
alter table PRACOWNICY
modify (constraint PK_pracownicy primary key (ID_P),
          IMIĘ constraint NN1_pracownicy not null,
          NAZWISKO constraint NN2_pracownicy not null,
          constraint RFK_pracownicy foreign key (ID_BP)
                                        references PRACOWNICY,
          constraint FK_pracownicy foreign key (ID_D) references DZIAŁY,
          constraint C_pracownicy check (PENSJA >= 0)
         );
```

Uwaga:
Poza zmianą stanu więzu (patrz opis podklauzuli MODIFY w podrozdziale I.3), np. włączony (ENABLE) / wyłączony (DISABLE) oraz zmianą nazwy własnej, w celu zmiany więzu (np. gdy chcemy dodać klauzulę ON DELETE) najczęściej należy go najpierw usunąć przy pomocy komendy:

```
ALTER TABLE  [schemat_użytkownika.]tabela
DROP CONSTRAINT Nazwa_własna_więzu;
```

Na przykład:

```
alter table PRACOWNICY
drop constraint C_pracownicy;
```

Dodatkowo więzy typu:

- klucz główny można usunąć poprzez polecenie:

```
ALTER TABLE  [schemat_użytkownika.]tabela
DROP PRIMARY KEY;
```

- klucz unikalny można usunąć poprzez polecenie:

```
ALTER TABLE  [schemat_użytkownika.]tabela
DROP UNIQUE (nazwa_kolumny1, nazwa_kolumny2, …);
```

Nazwę własną więzu można zmienić bez usuwania więzu, używając podklauzulę RENAME CONSTRAINT:

```
ALTER TABLE [schemat_użytkownika.]tabela
RENAME CONSTRAINT Stara_nazwa_więzu TO Nowa_nazwa_więzu;
```

Podobnie można zmienić nazwę kolumny tabeli:

```
ALTER TABLE [schemat_użytkownika.]tabela
RENAME COLUMN Stara_nazwa_kolumny TO Nowa_nazwa_kolumny;
```

Natomiast zmianę nazwy tabeli można wykonać na dwa sposoby:

```
ALTER TABLE [schemat_użytkownika.]tabela
RENAME TO Nowa_nazwa_tabeli;

RENAME Stara_nazwa_tabeli TO Nowa_nazwa_tabeli;
```

I. 6 Perspektywy (widoki) w bazach danych – wstępne informacje

Perspektywę można zdefiniować jako składowane w bazie danych zapytanie.

Ta definicja jest w pewien sposób uproszczona, nie oddaje na przykład takich własności perspektyw jak możliwość wstawiania do tabel i usuwania z nich danych poprzez perspektywy oparte na tych tabelach. Jednak wstępnie, do naszych potrzeb, jest ona wystarczająca. Ponieważ poprzestaniemy na najprostszej postaci definicji perspektywy, nie omawiając elementów decydujących o własnościach operacji DML (INSERT, UPDATE i DELETE) w przypadku perspektyw:

- tworzenie nieistniejącej perspektywy:

```
create view  [schemat_użytkownika.]Nazwa_perspektywy
[ ( kolumna1, kolumna2 ... ) ]
as zapytanie;
```

Przykład:

Tworzenie perspektywa DZIAL10 o tych samych nazwach kolumn, jakie są w tabeli DZIAŁY:

```
create view DZIAL10
as select * from DZIAŁY where ID_D=10;
```

Tworzenie perspektywa DZIAL10 o innych nazwach kolumn niż te, które są w tabeli DZIAŁY:

```
create view DZIAL10
(ID_DZIALU, NAZWA_DZIALU)
as select ID_D, NAZWA from DZIAŁY where ID_D=10;
```

- usuwanie i ponowne tworzenie perspektywy bez utraty uprawnień do niej przydzielonych:

```
create or replace view [schemat_użytkownika.]Nazwa_perspektywy
[ ( kolumna1, kolumna2 ... ) ]
as zapytanie;
```

Przykład:

```
create or replace view DZIAL10
(ID_DZIALU, NAZWA_DZIALU)
as select ID_D, NAZWA from DZIAŁY
   where ID_D=10 or ID_D=11;
```

- usuwanie perspektywy:

```
drop view [schemat_użytkownika.]Nazwa_perspektywy;
```

Przykład:

```
drop view DZIAL10;
```

I. 7 Komentarze do obiektów bazy danych

- Komentarz do tabeli lub perspektywy

Komentarz do tabeli:

```
COMMENT ON TABLE [schemat_użytkownika.]Nazwa_tabeli
IS 'tekst komentarza';
```

Na przykład:

```
comment on table PRACOWNICY
is ' Tabela zawiera dane o pracownikach firmy'
```

Komentarz do perspektywy jest definiowany identycznie:

```
COMMENT ON TABLE [schemat_użytkownika.]Nazwa_perspektywy
IS 'tekst komentarza';
```

- Komentarz do kolumny tabeli lub perspektywy

Komentarz do kolumny tabeli:

```
COMMENT ON COLUMN
[schemat_użytkownika.]Nazwa_tabeli.Nazwa_kolumny
IS 'tekst komentarza';
```

Na przykład:

```
comment on column PRACOWNICY.ID_P
is ' Kolumna ID_P zawiera unikalne identyfikatory pracowników firmy
     i jest kluczem głównym tabeli PRACOWNICY'
```

Komentarz do kolumny perspektywy jest definiowany identycznie:

```
COMMENT ON COLUMN
[schemat_użytkownika.]Nazwa_perspektywy.Nazwa_kolumny
IS 'tekst komentarza';
```

II. Język manipulowania danymi - DML (Data Manipulation Language). Polecenia INSERT, UPDATE, DELETE

II. 1 Wstawianie danych do tabel - polecenie **INSERT**

- polecenie INSERT wstawiające do tabeli wiersz danych podanych bezpośrednio:

  INSERT INTO [schemat_użytkownika.]tabela[@baza_danych]
  [(kolumna, ...)] VALUES (wartość lub wyrażenie, ...);

  Podstawowe przykłady:

  insert into PRACOWNICY (ID_P,IMIĘ,NAZWISKO,PENSJA)
  values (2000,'Tomasz','Zborowski',3000);

  Jeśli wstawiamy dane do wszystkich kolumn (tzn. przypisujemy wartości wszystkim atrybutom w krotce), to można pominąć nazwy kolumn w poleceniu INSERT, ale szczególnie w aplikacjach nie jest to zalecane ze względu na możliwość zmiany struktury tabeli (relacji).
  Gdy jednak zdecydujemy się na taką wersję polecenia INSERT to kolejność wartości atrybutów wprowadzanych po słowie kluczowym VALUES powinny być taka, jak kolejność nazw kolumn (atrybutów) wyświetlanych przez polecenie DESC:

  desc PRACOWNICY
  => kolumny: ID_P, IMIĘ, NAZWISKO, PENSJA, ID_BP, ID_D

  Wówczas można np. wykonać następujące polecenie INSERT:
  insert into PRACOWNICY values (2001, 'Michał','Stręk',2500,2000,10);

- polecenie INSERT wstawiające do tabeli wiersz danych pobierając je za pomocą podzapytania:

  INSERT INTO [schemat_użytkownika.]tabela[@baza_danych]
  [(kolumna, ...)] *podzapytanie*;

  Przykład:
  Załóżmy, że w naszej bazie istnieje tabela PRACOWNICY_DZIAŁU_11 o identycznym nagłówku jak tabela PRACOWNICY, która jest pusta. Wówczas możemy wstawić do niej wszystkie dane z tabeli PRACOWNICY dla działu 11, używając polecenia:

  insert into PRACOWNICY_DZIAŁU_11
  select * from PRACOWNICY where ID_D = 11;

- Uwagi ogólne dotyczące polecenia INSERT:

  a) Należy pamiętać o ograniczeniach wynikających z nałożonych więzów integralności oraz innych więzów:

     - nie można wprowadzić więcej niż jednego wiersza o takich samych wartościach atrybutów klucza głównego lub unikalnego;

     - nie można wprowadzić wartości atrybutu (atrybutów) tworzących klucz obcy, jeżeli nie zostały one wprowadzone wcześniej jako wartości atrybutu (atrybutów) klucza głównego lub unikalnego, do którego ten klucz obcy odnosi się;

     - nie można wprowadzić wartości atrybutu, która byłaby niezgodna z narzuconymi innymi więzami niż więzy integralności relacyjnej bazy danych, z tzw. więzami typu *check*, np. PENSJA >= 0

b) Wartości atrybutów typu numerycznych wprowadzamy bez apostrofów, natomiast wartości atrybutów typu tekstowego oraz daty i czasu zawsze ujmujemy w apostrofy!

Dobrym nawykiem jest używanie funkcji TO_DATE do wprowadzania wartości typu DATE. Załóżmy, że tabela PRACOWNICY ma dodatkowo kolumnę DATA_ZATRUDNIENIA. Wówczas powinniśmy wprowadzać do niej wartości w następujący sposób:

```
insert into PRACOWNICY
(ID_P, IMIĘ, NAZWISKO, DATA_ZATRUDNIENIA)
values (2000,'Paweł','Kłos',to_date('12.10.2006','DD.MM.YYYY'));
```

Wówczas unikniemy błędów wynikających z możliwej niepoprawnej konwersji wprowadzanej wartości do postaci zgodnej z domyślnym formatem daty i czasu.

Warto pamiętać, że jeśli wpiszemy datę do atrybutu typu DATE bez podania jawnie godziny, to zostanie ustawiona godzina (00:00:00), czyli północ.

Przykładowo, zakładając, że format wprowadzanej daty zatrudnienia jest zgodny z domyślnym formatem daty i czasu serwera, wyrażenia:

```
'12.10.2006'

to_date('12.10.2006','DD.MM.YYYY')

to_date('12.10.2006','DD.MM.YYYY HH24:MI:SS')
```

będą skutkować przyjęciem przez atrybut DATA_ZATRUDNIENIA wartości '12.10.2006 00:00:00'.

- polecenie INSERT ALL, umożliwiające wstawianie wielu wierszy danych do różnych tabel, zaprezentowane w przykładzie o pewnym poziomie ogólności:

```
INSERT ALL
   INTO tabela1 [ (kolumna, ...) ] VALUES (wartość lub wyrażenie, ...)
   INTO tabela1 [ (kolumna, ...) ] VALUES (wartość lub wyrażenie, ...)
   INTO tabela2 [ (kolumna, ...) ] VALUES (wartość lub wyrażenie, ...)
   INTO tabela2 [ (kolumna, ...) ] VALUES (wartość lub wyrażenie, ...)
SELECT * FROM DUAL;
```

II. 2 Modyfikacja (uaktualnianie) danych w tabeli - polecenie **UPDATE**

UPDATE [schemat_użytkownika.]tabela[@baza_danych] [alias]
SET *kolumna = wartość lub wyrażenie*,
kolumna = (*podzapytanie*), ...
[WHERE *warunek*];

Alternatywna składnia:

UPDATE [schemat_użytkownika.]tabela[@baza_danych] [alias]
SET (*kolumna,kolumna,...)* = (*podzapytanie)*
[WHERE *warunek*];

Przykłady:

```
update PRACOWNICY set PENSJA = 3500
where ID_P = 1002;

update PRACOWNICY
set PENSJA = (select PENSJA*1.5 from PRACOWNICY
                where ID_P=1001)
where ID_P = 1003;

update PRACOWNICY
set IMIĘ = 'Ewa', Nazwisko='Wierzba',
PENSJA = (select PENSJA from PRACOWNICY where ID_P=1002)
where ID_P = 1004;

Zakładając, że tabela PRACOWNICY ma dodatkową kolumnę
MAX_PENSJA_W_DZIALE
update PRACOWNICY P1
set MAX_PENSJA_W_DZIALE = (select max(PENSJA)
                            from PRACOWNICY P2
                            where P2.ID_D = P1.ID_D);

update PRACOWNICY
set (PENSJA,ID_D) = (select PENSJA,ID_D from PRACOWNICY
                        where ID_P = 1003)
where ID_P = 1005;
```

II. 3 Usuwanie danych z tabeli - polecenie DELETE

DELETE [FROM] [schemat_użytkownika.]tabela[@baza_danych] [alias] [WHERE *warunek*];

Należy pamiętać, że dozwolone niezastosowanie klauzuli WHERE spowoduje usunięcie danych z całej tabeli!!!

Polecenia

```
delete from PRACOWNICY where ID_P = 1001;
```

oraz

```
delete PRACOWNICY where ID_P = 1001;
```

dadzą ten sam efekt.

Przykład z wykorzystaniem aliasu:

```
delete from PRACOWNICY P1
where PENSJA > (select AVG(PENSJA) from PRACOWNICY P2
                    where P2.ID_D = P1.ID_D);
```

Czyszczenie danych – trwałe usuwanie danych z tabeli – polecenie TRUNCATE

TRUNCATE [schemat_użytkownika.]tabela[@baza_danych]

Komenda TRUNCATE nie jest komendą z grupy DML.

Komenda TRUNCATE jest zaliczana do komend DDL.

W związku z tym do komendy TRUNCATE stosują się te same reguły zatwierdzania transakcji jak do innych komend z grupy DDL, tzn. transakcja komendy TRUNCATE nie wymaga zatwierdzenia za pomocą komendy COMMIT. Dlatego skutków komendy TRUNCATE **nie można wycofać** komendą ROLLBACK.

III. Elementy języka sterowania transakcjami - TCL (Transaction Control Language).

III. 1 Zatwierdzanie i wycofywanie poleceń DML

W starszych wersjach serwera Oracle, ale również w Oracle 11g, 12g i 18c:

Jeśli nie zatwierdzimy komendą COMMIT poleceń DML, to transakcje z nimi związane tylko pozornie zostają wykonane (*pod warunkiem, że np. parametr AUTOCOMMIT środowiska SQL*Plus jest ustawiony na OFF*), tzn. możemy zobaczyć ich pozorne efekty w bieżącej sesji użytkownika (inni nawet odpowiednio uprawnieni użytkownicy nie widzą tych zmian), ale po jej zakończeniu wszystkie dane w schemacie użytkownika są w takim stanie jak przed wykonaniem poleceń DML.

W Oracle 10g Release 2:

Zatwierdzanie transakcji nawet związanych z poleceniami DML jest domyślne. Jednak jeśli program klienta bazy danych (np. SQL*Plus) zostanie zamknięty w sposób niewłaściwy (do którego należy zaliczyć także kliknięcie na przycisk zamknięcia okna tej aplikacji zamiast użycie komendy EXIT), to ostatnia transakcja typu DML, która nie został zatwierdzona poleceniem **COMMIT** zostaje wycofana.

Jednak opisane powyżej, domyślne zatwierdzanie transakcji DML nie jest realizowane podczas wykonywania programów skompilowanych prekompilatorem Oracle'a, tzn. zatwierdzanie transakcji DML nie jest domyślne w przypadku zastosowania składowanych w bazie danych Oracle funkcji i procedur oraz ich pakietów, a także najczęściej w przypadku poleceń DML wbudowanych w aplikacje bazodanowe napisane w różnych językach programowania.

Wspólna cecha sterowania transakcjami w Oracle 10g i w starszych wersjach:

Jeśli transkacja DML nie została zatwierdzona (poleceniem COMMIT lub też automatycznie przez serwer), to można ją wycofać poleceniem **ROLLBACK**.

Polecenie:

SAVEPOINT *Nazwa_punktu_zachowania*

definiuje tzw. *punkt zachowania*, który pozwala podzielić transakcje na mniejsze części. Ponowne zdefiniowanie punktu zachowania o tej samej nazwie usuwa starą definicję. Punkty zachowania są zorganizowane stosowo.

Po zdefiniowaniu *punktu zachowania* można *do niego* wycofywać transakcje:

ROLLBACK [WORK]
[TO [SAVEPOINT] *Nazwa_punktu_zachowania***];**

powoduje odtworzenie stanu bazy - albo do początku transakcji, albo do stanu zdefiniowanego punktem zachowania.

- Jeżeli podamy nazwę punktu zachowania, to zachowujemy ten punkt zachowania, lecz tracimy wszystkie punkty zachowanie zdefiniowane *później*.
- ROLLBACK bez nazwy punktu zachowania zamyka bieżącą transakcję i kasuje wszelkie punkty zachowania.

III. 2 Polecenia DDL są domyślnie zatwierdzane przez serwer bazy danych.

Jednak w dokumentacji Oracle'a zalecane jest zatwierdzanie wszystkich transakcji, aby mieć pewność, że zostały one przeprowadzone.