

# Komercyjne bazy danych 2

## Uzupełnienia

doc. dr inż. Tomasz Traczyk

Instytut Automatyki i Informatyki Stosowanej  
Politechnika Warszawska

Styczeń 2010

## Zastosowanie wyzwalacza instead of

### Niemodyfikowalna perspektywa

```
create or replace view kopie_do_druku as
select
  'U' || id_umowy as klucz,          wyrażenia
  'UMOWY' as tabela,
  'KOPIE_UMOWY' as kolumna,
  rowid as row_id,                 pseudokolumna
  kopie_umowy as kopie
FROM umowy
union                               i union
select
  'R' || id_rachunku as klucz,
  'RACHUNKI' as tabela,
  'KOPIE_RACHUNKU' as kolumna,
  rowid as row_id,
  kopie_rachunku as kopie

from rachunki
```

Chcemy móc zmieniać wartości kolumny kopie\_rachunku

## Wyzwalacz z użyciem dynamicznego SQL

```
create or replace trigger kopie_update_trg
instead of update on kopie_do_druku
declare
  s varchar2(1000);
begin
  s := 'update ' || :new.tabela ||
    ' set ' || :new.kolumna || '= :1 where rowid=:2';
  execute immediate s using :new.kopie, :new.row_id;
end;
```

Wyzwalacz zastępuje update

Sformowanie zdania DDL z użyciem zmiennych związanych (*bind variables*)

Wykonanie dynamicznego SQL z podstawieniem zmiennych

## Buforowanie wyników zapytań i funkcji

- ▶ Buforowanie wyników zapytań: hint `/*+ result_cache */`
- ▶ Buforowanie wyników funkcji:  
`create or replace function nazwa(parametry) return wynik  
result_cache relies_on (lista_tabel)  
is ...`
- ▶ Bufory są automatycznie odświeżane przy pierwszym zapytaniu po zmianie danych
- ▶ Do zarządzania buforowaniem służy pakiet `DBMS_RESULT_CACHE`, np. `dbms_result_cache.bypass(true)`;

## SecureFiles

- ▶ Nowa opcja przy tworzeniu kolumn typu LOB
- ▶ Możliwości
  - ▶ kompresja
  - ▶ deduplikacja – zastąpienie powtórzeń rodzajem wskaźników
  - ▶ szyfrowanie
  - ▶ zarządzanie logowaniem w segmentach powtórzeń
- ▶ Dostępność sterowana parametrem DBMS o nazwie DB\_SECUREFILE

## Kolumny wirtualne

```
create table sales
(...
nazwa_kolumny generated always as (wyrażenie) [virtual]
);
```

- ▶ Kolumny wyliczane na podstawie innych danych
  - ▶ można korzystać tylko z danych z tego samego wiersza
  - ▶ można używać deterministycznych funkcji PL/SQL (ale nie ma automatycznej inwalidacji!) – takie kolumny nie mogą jednak być kluczami partycjonowania
  - ▶ dozwolone są wszystkie wbudowane typy skalarne i XMLType
- ▶ Kolumny wirtualne
  - ▶ są niemodyfikowalne – nie można ich wartości wstawiać ani zmieniać
  - ▶ mogą wchodzić w skład kluczy głównych, obcych i kluczy partycjonowania
  - ▶ można indeksować – powstaje indeks funkcjonalny
  - ▶ można definiować tylko w tabelach *heap-organized*, nie można definiować w tabelach tymczasowych

## Partycjonowanie

- ▶ Partycjonowanie typu *reference*: partycjonowanie zgodne z partycjonowaniem innej tabeli (połączonej przez klucz obcy):
 

```
create table nazwa_tabeli (
  kolumny,
  constraint foreign key nazwa_fk (klucz_obcy) references tabela
) partition by reference (nazwa_fk);
```
- ▶ Partycjonowanie typu *interval* – automatyczne tworzenie kolejnych partycji dla przedziałów czasu, np.:
 

```
create table nazwa (kolumny)
partition by range (kolumna_z_data)
interval (numtoyminterval(1,'MONTH'))
(partition partycja1 values less than (data1));
```
- ▶ Partycjonowanie typu *system* – „ręczne” wskazywanie partycji
- ▶ Partycjonowanie dwupoziomowe (*composite*) w dowolnych kombinacjach typu partycjonowania
- ▶ Partycjonowanie na podstawie kolumn wirtualnych

## Nowe elementy zarządzania schematami

### Tabele *read-only*

```
alter table nazwa read only | read write;
```

### Niewidzialne indeksy

```
alter index nazwa invisible;
```

- ▶ Indeks przestaje być dostępny dla optymalizacji zapytań
- ▶ Hint `/* INDEX (nazwa) */` udostępnia indeks „ręcznie”
- ▶ `alter session set optimizer_use_invisible_indexes = true;` – udostępnia wszystkie niewidzialne indeksy

## Zmiany w PL/SQL

### Ulepszona kompilacja do kodu natywnego

- ▶ Kompilacja przez DBMS, bez konieczności instalacji kompilatora C
- ▶ Znacznie przyspieszona
- ▶ perspektywa USER\_PLSQL\_OBJECT\_SETTINGS pokazuje tryb w którym skompilowano poszczególne moduły

### Intra-unit Inlining

- ▶ Kopiowanie procedur lokalnych<sup>1</sup> zamiast ich wywoływania
- ▶ Znacznie przyspiesza wykonanie iteracji
- ▶ Wykonywane automatycznie po ustawieniu parametru  
alter session set plsql\_optimize\_level = 3;  
lub wymuszane pragmatą  
pragma inline (nazwa\_procedury\_lokalnej, 'YES');

<sup>1</sup>podrzędnych w stosunku do modułu podprogramu

## Zapytania typu *cross-tab*

### Operacja pivot z listą kolumn

```
select * from
  ( select nazwisko as wykladowca, id_przedmiotu, liczba_godzin
    from przedmioty natural join wykladowcy
  )
pivot
  ( sum(liczba_godzin) for id_przedmiotu in
    ('OTW' as ogólna, 'STNR' as szczególna, 'APOT' as aspekty)
  );
```

	WYKLADOWCA	OGÓLNA	SZCZEGÓLNA	ASPEKTY
1	Abacki	10	5	(null)
2	Babacki	(null)	(null)	3

- ▶ Istnieje także operacja unpivot

### Operacja pivot xml z podzapytaniem

```
select * from
  ( select nazwisko as wykladowca, id_przedmiotu, liczba_godzin
    from przedmioty natural join wykladowcy
  )
pivot xml
  ( sum(liczba_godzin) as suma for id_przedmiotu in
    (select id_przedmiotu from przedmioty)
  );
```

	WYKLADOWCA	ID_PRZEDMIOTU_XML
1	Abacki	<PivotSet><item><column name = "ID_PRZEDMIOTU">APOT</column><column name = "SUMA"></column></item></PivotSet>
2	Babacki	<PivotSet><item><column name = "ID_PRZEDMIOTU">APOT</column><column name = "SUMA">3</column></item></PivotSet>

```
<PivotSet>
  <item>
    <column name="ID_PRZEDMIOTU">
      APOT
    </column>
    <column name="SUMA"/>
  </item>
  <item>
    <column name="ID_PRZEDMIOTU">
      OTW
    </column>
    <column name="SUMA">
      10
    </column>
  </item>
  <item>
    <column name="ID_PRZEDMIOTU">
      STNR
    </column>
    <column name="SUMA">
      5
    </column>
  </item>
</PivotSet>
```

- ▶ Zamiast podzapytania można użyć klauzuli ALL

## Role niedomyślne

- ▶ Są to role nieaktywne po przyłączeniu do bazy
- ▶ Używane do nadawania dodatkowych uprawnień, gdy użytkownik działa za pomocą uprawnionych aplikacji
- ▶ Definiowane np. przez  
create role rola\_niedomyślna identified by hasło;  
grant rola\_niedomyślna to użytkownik;  
alter user użytkownik default role all except rola\_niedomyślna;
- ▶ Role takie można aktywować poleceniem  
set role rola\_niedomyślna [identified by hasło];  
lub wywołaniem DBMS\_SESSION.SET\_ROLE
- ▶ Jeśli roli przypisane było hasło, to trzeba je podać
- ▶ tzw. *Secure Application Roles* aktywować można tylko procedurą z podanego pakietu PL/SQL:  
create role rola identified using pakiet;
- ▶ Możliwe jest także zewnętrzne aktywowanie roli, np. przez LDAP

## Fine-grained access control i VPD

### Kontekst

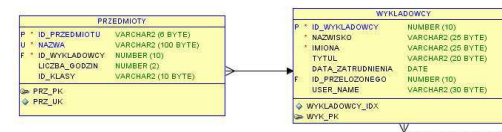
- ▶ Jest to mechanizm umożliwiający programowe wyliczanie parametrów używanych np. do autoryzacji dostępu typu *fine-grained*
- ▶ Dostępny we wszystkich „wagach” bazy danych
- ▶ Zasięg wyliczonych parametrów to sesja
- ▶ Wykorzystanie kontekstu np. w perspektywach:  

```
create or replace view moje_przedmioty as
select * from przedmioty where id_wykladowcy =
sys_context('CTX_PRZEDMIOTY', 'ID_WYKLADOWCY')
with check option;
```

- ▶ pobrano parametr ID\_WYKLADOWCY z kontekstu CTX\_PRZEDMIOTY

### Definiowanie kontekstu – przykład

- ▶ Kontekst ma podawać identyfikator wykładowcy, który jest bieżącym użytkownikiem
- ▶ Ten identyfikator może być użyty do ograniczenia dostępu do tabeli PRZEDMIOTY



- ▶ Tworzenie kontekstu:  

```
create context ctx_przedmioty using ctx_przedmioty_pkg;
```
- ▶ Pakiet podany w klauzuli using steruje zawartością kontekstu  
  - ▶ Pakiet [15]
- ▶ Zawartość kontekstu może także być określana zewnętrznie, np. przez LDAP

### Pakiet sterujący kontekstem

```
create or replace package body ctx_przedmioty_pkg is
  procedure set_app_context is          Ustawianie kontekstu
  begin
    idw wykladowcy.id_wykladowcy%TYPE;
  begin
    select id_wykladowcy into idw from wykladowcy where user_name=user;
    dbms_session.set_context('CTX_PRZEDMIOTY', 'ID_WYKLADOWCY', idw);
  end;

  procedure show_app_context is        Odczyt kontekstu
  begin
    dbms_output.put_line(
      sys_context('CTX_PRZEDMIOTY', 'ID_WYKLADOWCY'));
  end;

  function the_predicate(p_schema varchar2, p_name varchar2)
  return varchar2 is
  begin
    return
      'ID_WYKLADOWCY='||sys_context('CTX_PRZEDMIOTY', 'ID_WYKLADOWCY');
  end;
begin
  ctx_przedmioty_pkg.set_app_context;
end;
```

## Virtual Private Database

- ▶ Automatyczne ograniczenie widoczności danych w tabelach, np. przy użyciu kontekstu
- ▶ Działanie: niejawne zastępowanie odwołań do tabeli podzapytaniem z automatycznie budowaną klauzulą WHERE
- ▶ Mechanizm dostępny tylko w *Enterprise Edition*, do zarządzania służy pakiet DBMS\_RLS
- ▶ Dostęp jest sterowany przez tzw. polityki, np.:  

```
begin
  dbms_rls.add_policy(
    schemat, 'PRZEDMIOTY', polityka,
    schemat, 'CTX_PRZEDMIOTY_PKG.THE_PREDICATE',
    'SELECT', FALSE, TRUE
  );
end;
```
- ▶ Po zdefiniowaniu polityki na danej tabeli użytkownik widzi tylko te jej wiersze, dla których predykat zwraca wartość TRUE
- ▶ Użytkownicy z przywilejem EXEMPT ACCESS POLICY widzą wszystkie dane