



Relacyjne Bazy Danych

Andrzej M. Borzyszkowski
PJATK/ Gdańsk

materiały dostępne elektronicznie
<http://szuflandia.pjwstk.edu.pl/~amb>

Relacyjne Bazy Danych © Andrzej M. Borzyszkowski

Architektura systemów zarządzania bazą danych

Relacyjne Bazy Danych © Andrzej M. Borzyszkowski

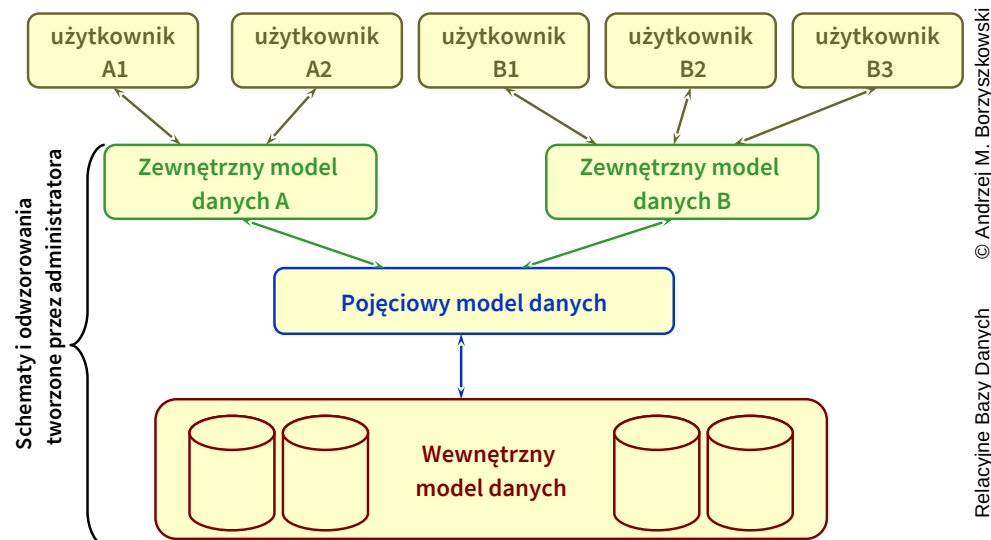
Architektura SZBD

Trzy poziomy architektury

1. wewnętrzny
 - fizyczne przechowywanie danych
 - typy rekordów, indeksy, reprezentacja pól, kolejność przechowywania
2. pojęciowy (konceptyjny)
 - reprezentacja całej zawartości informacyjnej bazy
 - również reguły spójności
3. zewnętrzny
 - perspektywa konkretnego użytkownika
 - typy, pola, rekordy widziane przez pewnego użytkownika mogą być różne dla różnych użytkowników

Relacyjne Bazy Danych © Andrzej M. Borzyszkowski

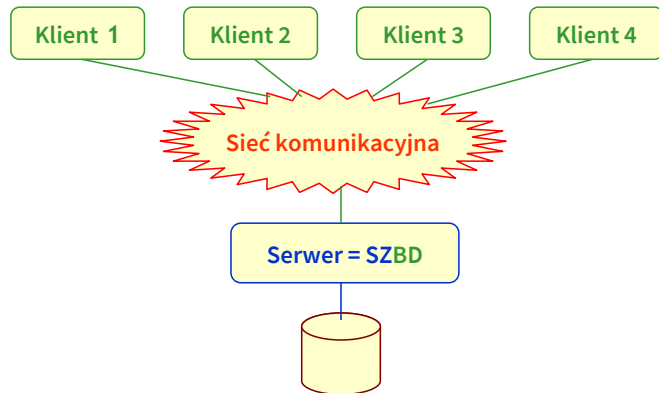
Architektura SZBD – schemat



Relacyjne Bazy Danych © Andrzej M. Borzyszkowski

Architektura SZBD – klient-serwer

- Serwer jest systemem zarządzania bazą danych
- Klientami są aplikacje poziomu zewnętrznego



5

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

Rekursja

- W standardowym SQL *nie ma* możliwości zażądania wyświetlenia wszystkich podzespołów na nieograniczonej głębokości
 - uwaga: standard SQL3 przewiduje taką możliwość
- W dowolnym języku programowania bez problemu można napisać pętlę (lub procedurę rekursywną) przeglądającą drzewo na całej głębokości

```
wyświetl(int mat){  
  print mat;  
  for Sub in BoM if Sub.part_of=mat wyświetl(Sub.id);  
}  
wyświetl(12345);
```

9

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

Programowanie po stronie serwera

7

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

Funkcje itp.

- Powód rozszerzeń
 - niewystarczalność SQL
 - wydajność, wygoda, etc.
- Rodzaje rozszerzeń
 - operatory
 - funkcje
 - procedury uruchamiane podczas startu bazy danych
 - procedury wyzwalane
- Możliwe języki programowania
 - SQL
 - PL/pgSQL
 - C
 - PL/Tcl, PL/Perl, PL/Python, i wiele innych

11

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

Definiowanie własnych funkcji

```
CREATE FUNCTION nazwa ([typ [,...]])  
  RETURNS typ_wyniku  
  AS definicja funkcji w jakimś języku  
  LANGUAGE nazwa_języka
```

```
CREATE FUNCTION plus_raz(int4)  
  RETURNS int4  
  AS '  
  BEGIN  
  RETURN $1+1; -- można też dać nazwę dla argumentu  
              -- ALIAS liczba FOR $1  
  END  
  ' LANGUAGE 'plpgsql'
```

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

15

Funkcje c.d.

- Sprawdzanie funkcji
`SELECT prosrc FROM pg_proc WHERE proname='plus_raz';`
- Usuwanie funkcji
`DROP FUNCTION plus_raz(int4)`
 - być może są inne funkcje plus_raz; nie zostaną one usunięte
 - “prawdziwa” nazwa funkcji zawiera jej typ
- Apostrof
 - może być potrzebny w definicji funkcji, wówczas podwójny
 - albo definicję objąć \$\$

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

17

Definiowanie funkcji c.d.

- Język
 - musi być znany postgresowi, tzn. musi być uruchomiony do działania
 - `createlang -U postgres plpgsql mojabaza -L/usr/local/pgsql/lib`
 - `SELECT * FROM pg_language`
 - `DROP language 'plpgsql'`
 - domyślnie NIE jest uruchamiany żaden język
 - w bazie danych przechowywany jest kod funkcji, kompilacja nastąpi przy pierwszym wywołaniu
 - wniosek: dopiero wówczas ujawnią się błędy

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

16

Definiowanie funkcji c.d.

- Można wprowadzać nazwy dla parametrów formalnych (wcześniejsze wersje Postgresa nie pozwalały na to)
- Zamiast CREATE można użyć REPLACE albo CREATE OR REPLACE
 - ale nie zmieni się w ten sposób typów argumentu/ wyniku
- Parametr może być zadeklarowany jako
 - wejściowy IN (wartość)
 - wyjściowy OUT (zapis), INOUT
 - jeśli występują parametry wyjściowe, to można zrezygnować z RETURNS

```
CREATE FUNCTION pisz(IN int, OUT int, OUT text) AS $$  
  SELECT $1, CAST($1 AS text)||' jest też tekstem'  
  $$ LANGUAGE 'SQL'
```

```
SELECT pisz(44)  
SELECT * FROM pisz(44)
```

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

18

Język PL/pgSQL

- Program składa się z bloków, każdy ma swe lokalne deklaracje
DECLARE deklaracje BEGIN instrukcje END
 - komentarze identyczne jak w SQL -- /* */
 - zakres deklaracji zmiennych oczywisty
 - zmienna może być inicjalizowana
 - zmienna może być zadeklarowana jako stała (constant), wówczas musi być inicjalizowana
 - typ zmiennej może odwołać się do innego typu
 - jeden integer :=1;
 - pi constant float8 := pi();
 - mójopis towar.opis%TYPE :='jakiś tekst'
 - wiersz record [typ ujawni się w momencie użycia]

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

19

Sterowanie

- Instrukcje warunkowe
 - IF () THEN ____
ELSEIF () THEN ____
ELSE ____
END IF
 - Wyrażenia warunkowe
 - NULLIF (wejście, wartość)
- zamienia określoną wartość na NULL
- CASE
WHEN ____ THEN ____
WHEN ____ THEN ____
ELSE ____
END

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

21

Zmienne wierszowe

- ```
DECLARE nowy_k, stary_k klient%ROWTYPE;
BEGIN
 nowy_k.miasto := 'Gdynia';
 nowy_k.ulica_dom := 'Tatrzańska 2';
 nowy_k.kod_pocztowy := '81-111';
 SELECT * INTO stary_k FROM klient WHERE nazwisko='Miszke';
 IF NOT FOUND THEN -----
 END IF;
END
```
- SELECT powinien zwrócić najwyżej jeden wiersz, dalsze zostaną zignorowane
  - chyba, że użyto SELECT \* INTO STRICT \_\_\_\_, wówczas błąd
  - istnieją sterowania FOR, LOOP, CONDITIONAL, RETURN (obowiązkowy), RAISE

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

20

## Pętle

- Pętle

```
LOOP n:=n+1; EXIT już WHEN n>1000; END LOOP;
WHILE n<=1000 LOOP n:=n+1 END LOOP;
FOR i IN 1..1000 LOOP _____ END LOOP;
FOR wiersz IN SELECT ____ LOOP ____ END LOOP;
EXIT
```

  - albo warunkowo: EXIT WHEN (coś się stało)
  - EXIT z\_miejsca, opuszcza nie tylko bieżącą pętlę, ale i pętle wyżej położone, aż do etykiety z\_miejsca

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

22

## Wynik działania procedury

- RETURN, normalne zakończenie działania
  - oznacza koniec obliczeń, nawet przed końcem bloku
  - *musi* wystąpić, brak RETURN jest błędem
  - RETURN NEXT nie kończy obliczeń, dodaje tylko kolejny wynik gdy spodziewamy się wyniku SETOF *typ*
- Wyjątki/komunikaty
  - RAISE DEBUG, zapisuje komunikat do pliku logów
  - RAISE NOTICE, wyświetla komunikat na ekran
  - RAISE EXCEPTION, j.w. + przerywa działanie procedury
  - RAISE NOTICE "wartość = %", zmienna
  - po zdefiniowaniu w funkcji i wywołaniu wyświetli na ekranie komunikat o wartości zmiennej
  - można podstawiać za % wyłącznie napisy

23

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Procedury wyzwalane

- Nazwa: *trigger*, trygier, wyzwalacz
- Jak?
  - procedury są wyzwalane “automatycznie” przez zdarzenia w bazie danych
- Dlaczego?
  - poprawność danych (pojedynczych, zależnych od innych)
  - śledzenie zmian, audyt, raport, zapis zmian
  - naruszenie postaci normalnej, kopie danych, dane wynikowe
  - dane bieżące vs. archiwalne
  - „denormalizacja”, spowodowane ergonomią, wydajnością, specjalny format danych dla innych aplikacji

30

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Procedury wyzwalane

29

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Przykłady „dlaczego” – poprawność

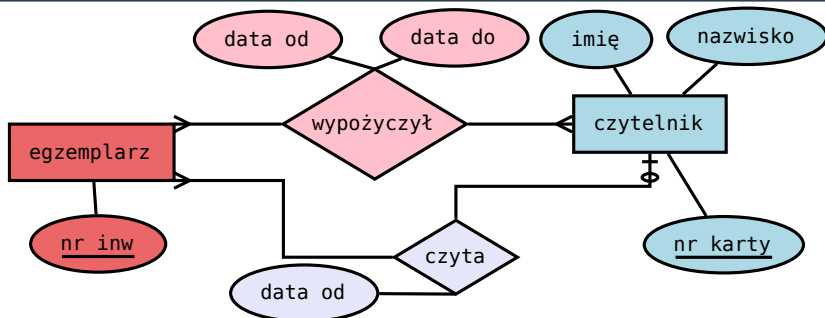
- Formalna poprawność danych
  - pesel ma 11 cyfr
- Przykłady pętli w diagramach związków i encji w projektach:
  - lekarz <wykonuje> zabieglekarz <ma> specjalizację
- zabieg <wymaga> specjalizacji
  - wymagane sprawdzenie zgodności
  - nauczyciel <uczy> klasa, przedmiot (związek 3 encji)nauczyciel <jest wychowawcą> klasa
  - tylko jeśli uczy jakiegoś przedmiotu w klasie
  - klub <jest gościem> w meczuklub <jest gospodarzem> w meczu
  - ale nie może być jednym i drugim

31

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Przykłady „dlaczego” – archiwizacja



- W projekcie biblioteki zapisuje się związek <czyta>
  - notując czas wypożyczenia automatycznie
  - po zwrocie książki zapisywane są pełne dane wypożyczenia
  - po zmianie ceny towaru zapisywana jest odrębnie dawna cena
  - po usunięciu faktury dane są archiwizowane

32

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Przykłady „dlaczego” – dane wynikowe, kopie

- Dane o towarach wprowadzane w centrali firmy
  - są automatycznie kopiowane w oddziałach
- Zamówienie złożone przez klienta
  - zmienia atrybut „łącznie suma zamówień”
  - warto mieć pod ręką dane zbiorcze by łatwiej obliczyć rabat dla kolejnego zamówienia
  - gol w meczu zmienia wynik meczu, zmienia pozycję klubu w lidze, zmienia statystyki i ranking piłkarza

33

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Definiowanie procedur wyzwalanych

- **CREATE TRIGGER nazwa BEFORE|AFTER INSERT|DELETE|UPDATE ON nazwa\_tablicy FOR EACH ROW|STATEMENT EXECUTE PROCEDURE nazwa\_funkcji(arg)**
- procedura używana w definicji wyzwalacza musi być wcześniej zdefiniowana
  - typ wynikowy musi być TRIGGER
  - zwraca albo NULL, albo wiersz pasujący do typu tabeli występującej w wyzwalaczu
  - formalnie nie ma argumentów, naprawdę argumenty odczytuje z tablicy tg\_argv[] o wielkości tg\_nargs
  - może odwoływać się do **new** i **old**, nowa i stara wartość zmienianego wiersza (dla wyzwalaczy FOR EACH ROW)

34

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Przykład: odnowienie zapasów

- ```
CREATE TRIGGER uzupelnij_trig
AFTER INSERT OR UPDATE ON zapas
FOR EACH ROW EXECUTE PROCEDURE uzupelnij_trig_proc(13);
```
- jeśli zostanie dokonana zmiana w tabeli zapasów, to zostanie wywołana procedura `uzupelnij_trig_proc`, która bada, czy zapasy nie są zbyt małe i być może trzeba złożyć zamówienie (do specjalnej tabeli nowych zamówień)
- ```
CREATE FUNCTION uzupelnij_trig_proc()
RETURNS TRIGGER AS $$
DECLARE
 prog INTEGER;
wiersz RECORD;
```
- ciąg dalszy na następnym slajdzie

35

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

## Przykład, c.d.

**BEGIN**

```
prog := tg_argv[0];
RAISE NOTICE 'próg wynosi %', prog;
IF new.ilosc < prog
THEN
 SELECT * INTO wiersz FROM towar
 WHERE nr = new.towar_nr;
 INSERT INTO nowe_zamowienie
 VALUES (wiersz.nr, wiersz.opis, prog-new.ilosc, now());
 RAISE NOTICE 'trzeba zamówić towar: %', wiersz.opis;
END IF;
RETURN NULL;
END; $$ LANGUAGE 'plpgsql'
```

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

36

## Reguły dla perspektyw

- Reguły są jedyną możliwością aktualizacji perspektyw
  - założmy, że mamy zadeklarowaną perspektywę

```
CREATE VIEW towar_zysk AS
 SELECT *, cena - koszt AS zysk FROM towar
```

  - nie ma możliwości usuwania wierszy z perspektywy
  - ale można zdefiniować regułę

```
CREATE RULE towar_zysk_del AS
 ON DELETE TO towar_zysk
 DO INSTEAD DELETE FROM towar WHERE nr=old.nr
```

  - wówczas polecenie usuwania z perspektywy usunie odpowiadający wiersz w tabeli

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

38

## Reguły

- Mogą mieć podobne skutki jak wyzwalacze

```
CREATE RULE name AS ON [UPDATE | INSERT | DELETE]
 TO table [WHERE condition]
 DO [ALSO | INSTEAD] { NOTHING | command | (command ;
 command ...) }
```

  - np.: archiwizacja danych

```
CREATE RULE archiwizuj AS ON UPDATE
 TO towar WHERE old.cena != new.cena
 DO INSERT INTO towar_log
 VALUES (old.nr, old.cena, now())
```

  - gdzie tabela towar\_log musiała być przedtem odpowiednio zdefiniowana

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

37

## Reguły vs. wyzwalacze

- Reguła powoduje (może spowodować) wykonanie kolejnego polecenia SQL
  - polecenie dotyczące wielu wierszy może być dobrze zoptymalizowane przez SZBD
- Wyzwalacz może zażądać wywołania funkcji dla wielu wierszy w tabeli
  - może to nie być tak efektywne jak reguła
- Wyzwalacze używają funkcji, a te mogą więcej niż polecenia SQL
  - np. integralność referencyjna wymaga sprawdzenia istnienia danych i ew. zgłoszenie błędu

© Andrzej M. Borzyszkowski

Relacyjne Bazy Danych

39