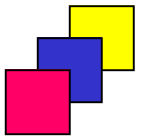
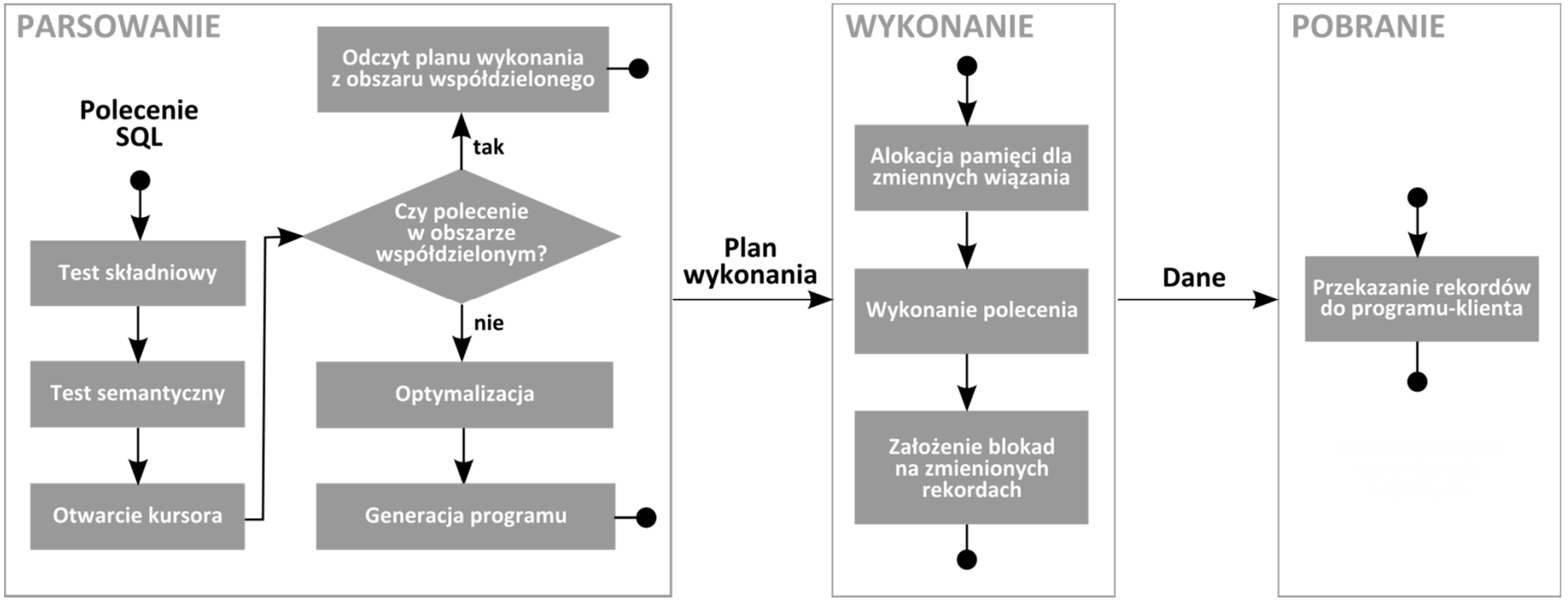
Optymalizacja poleceń SQL

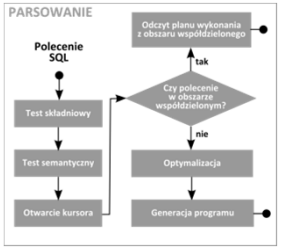
Wprowadzenie



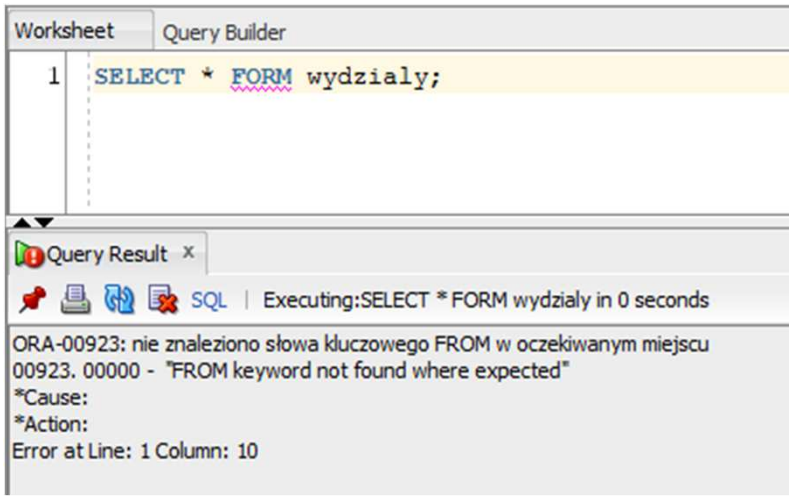
Fazy przetwarzania polecenia SQL



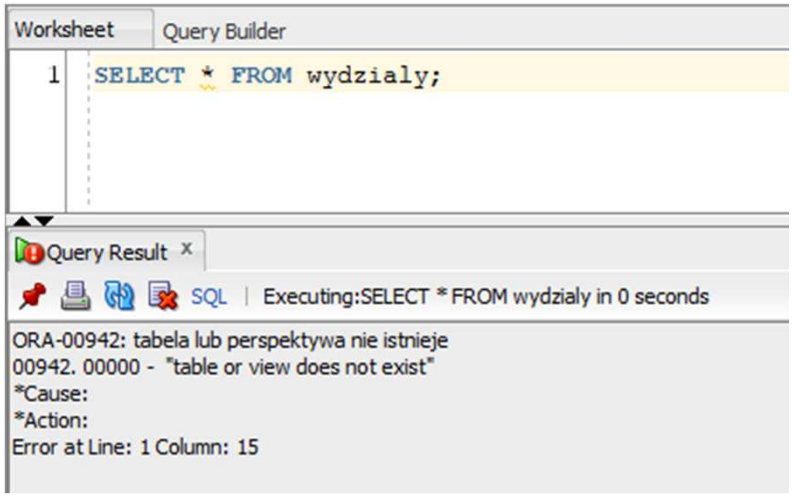
Faza parsingu (1)



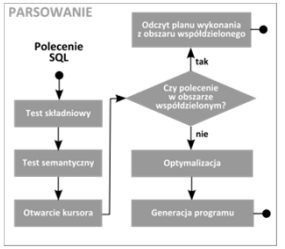
Krok 1. Test składniowy – weryfikacja poprawności składniowej polecenia SQL.



Krok 2. Test semantyczny – m.in. weryfikacja obecności obiektów adresowanych w poleceniu SQL.



Faza parsingu (2)



Krok 3. Otwarcie kursora

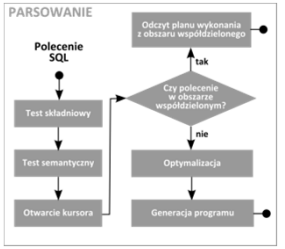
- **Kursor – obszar pamięci dla struktur danych, związanych z przetwarzaniem polecenia SQL, umieszczony w obszarze współdzielonym serwera.**
- **Następuje wyliczenie wartości identyfikatora polecenia SQL przy zastosowaniu funkcji haszującej.**

```
SQL> SELECT * FROM pracownicy;
...

SQL> SELECT sql_id FROM v$sql
     WHERE sql_text = 'SELECT * FROM pracownicy';

SQL_ID
-------------
01439xwqw8pjw
```

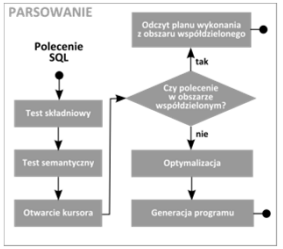
Faza parsingu (3)



Krok 4. Wyszukanie w obszarze współdzielonym serwera obecności identycznego, wcześniej wykonanego polecenia SQL

- **Wyszukanie na podstawie identyfikatora wyliczonego w kroku 3.**
- **Jeśli znaleziono identyczne polecenie, odczytywany jest zbudowany dla niego plan wykonania, faza parsingu ulega zakończeniu (tzw. miękki parsing).**
- **Jeśli wyszukanie zakończy się porażką (wcześniej nie zostało wykonane identyczne polecenie), następuje przejście do kroku optymalizacji polecenia (tzw. twardy parsing).**
- **Uwaga! Kolejne kroki są kosztowne!**

Faza parsingu (4)



Krok 5. Optymalizacja polecenia SQL

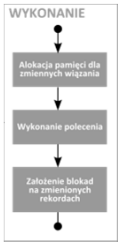
- **Wynik – plan wykonania polecenia SQL.**

Krok 6. Generacja binarnego programu wykonania polecenia SQL

- **Konwersja planu wykonania do binarnego programu wykonania polecenia SQL.**

- **Plan wykonania polecenia zostaje zapisany w obszarze współdzielonym.**

Faza wykonania



- **Następuje realizacja programu wygenerowanego dla polecenia SQL.**
- **Jeśli polecenie używało zmiennych wiązania, zostają one zamienione na konkretne wartości.**
- **System szuka danych dla polecenia w buforze bazy danych, jeśli ich tam nie ma, sprowadza je do bufora z nośników.**
- **Jeśli polecenie modyfikuje dane, następuje założenie odpowiednich blokad na danych.**

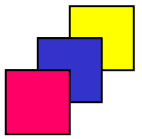
Faza pobrania

- **Występuje tylko dla poleceń SELECT lub poleceń DML z klauzulą RETURNING.**

- **Realizuje cykliczne pobieranie danych z bufora bazy danych i przesłanie ich do oprogramowania klienckiego.**

Optymalizacja

- Proces doboru odpowiednich struktur danych, metod dostępu i operacji, w celu zminimalizowania kosztu realizacji polecenia SQL.
- Wykonywana przez wyspecjalizowany moduł SZBD – optymalizator poleceń.
- Rodzaje:
  - regułowa:
    - oparta na rankingu metod dostępu do struktur danych,
    - niestosowana, preferowana dla aplikacji spadkowych,
  - kosztowa:
    - oparta na szacowaniu kosztu (czas zajętości procesora, liczby operacji we/wy, zajętość pamięci operacyjnej itp.), wykonania wszystkich potencjalnych sposobów wykonania polecenia SQL,
    - zalecana dla wszystkich nowo powstających aplikacji,
    - zakłada duże obciążenie systemu: dużą współbieżność operacji, niski współczynnik trafień w bufor danych.

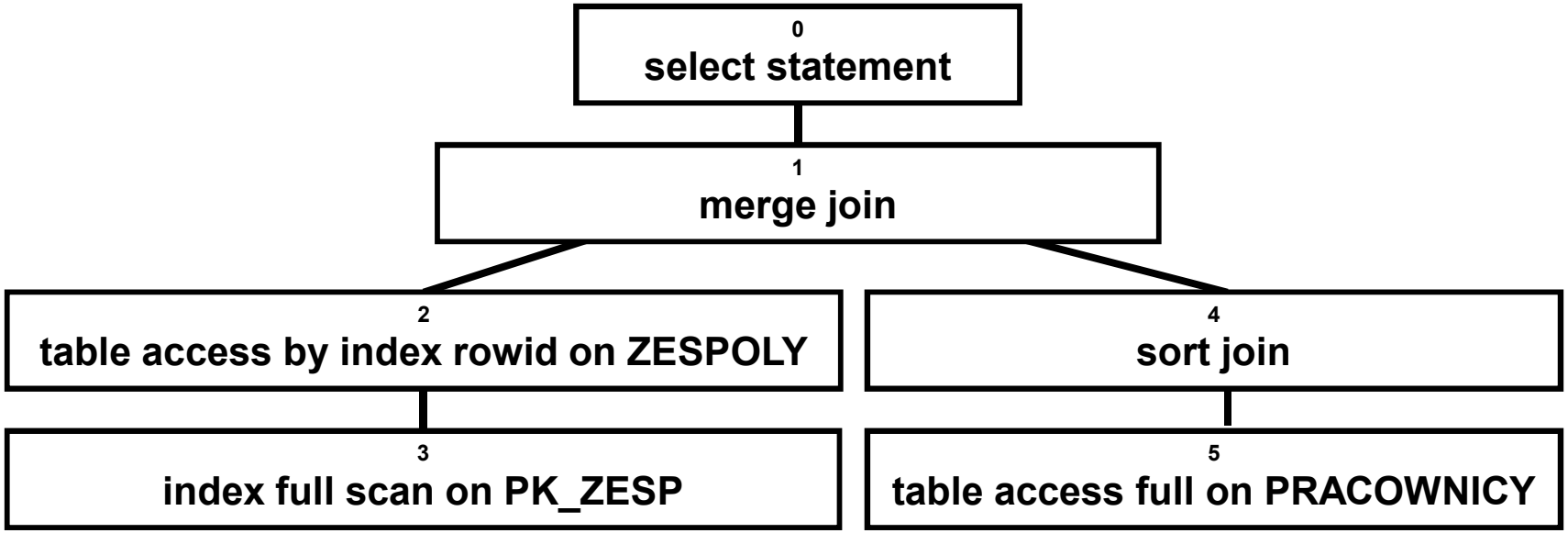


Plan wykonania polecenia SQL (1)

- Sekwencja operacji, które SZBD używa do wykonania polecenia SQL.
- Podstawowe informacje zawarte w planie wykonania:
  - ścieżki dostępu (ang. access path) do każdej z relacji, użytych w poleceniu SQL,
  - algorytmy łączenia relacji,
  - kolejność łączenia relacji,
  - operacje na danych, takie jak filtrowanie, sortowanie, agregacja.
- Dodatkowe informacje w planie:
  - koszt operacji (% wykorzystania czasu procesora),
  - czas wykonania operacji,
  - liczba rekordów i rozmiar danych, przetwarzanych przez operację.
- Operacje w planie tworzą drzewo.

Plan wykonania polecenia SQL (2)

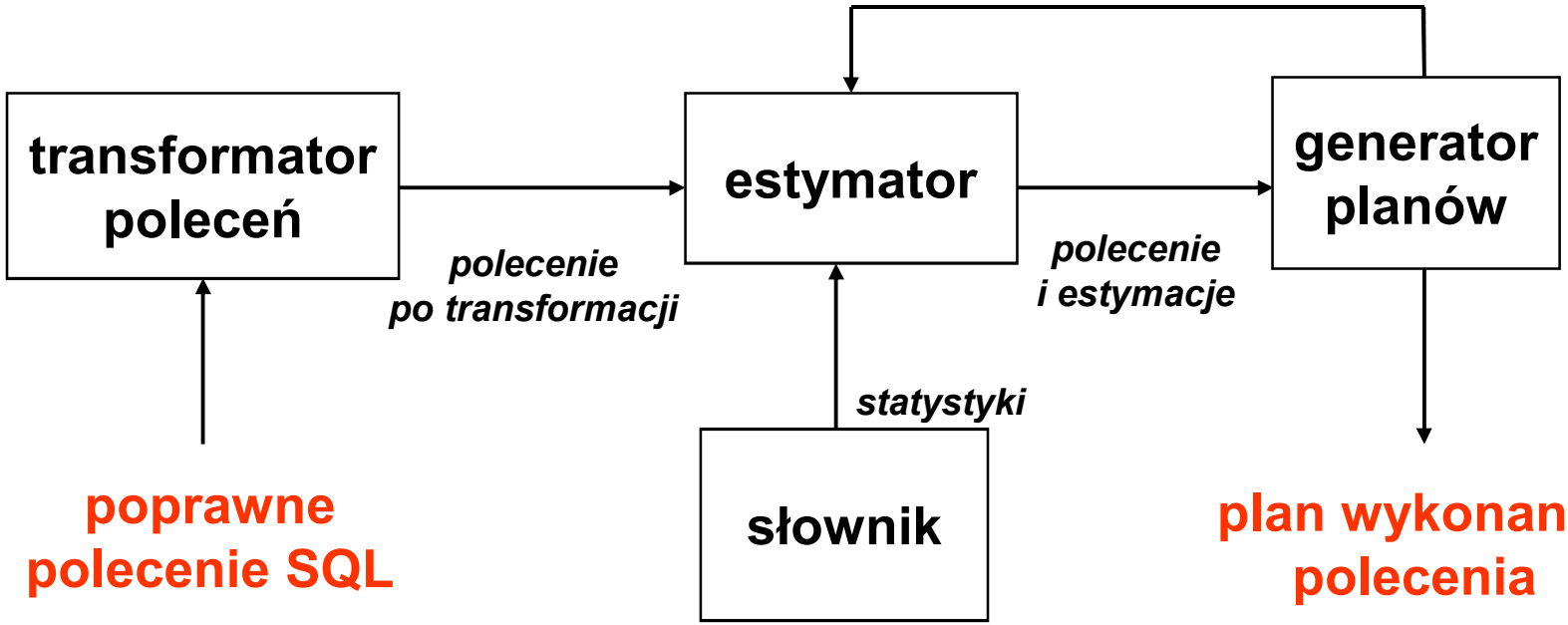
SELECT * FROM pracownicy NATURAL JOIN zespoly;




```
-------------------------------------------------------------------------------
| Id | Operation                    | Name       | Rows | Bytes| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |            | 14   | 1036 |      6 (17)| 00:00:01 |
|  1 |  MERGE JOIN                  |            | 14   | 1036 |      6 (17)| 00:00:01 |
|  2 |   TABLE ACCESS BY INDEX ROWID| ZESPOLY    | 6    | 174  |      2  (0)| 00:00:01 |
|  3 |    INDEX FULL SCAN           | PK_ZESP    | 6    |      |      1  (0)| 00:00:01 |
|  4 |   SORT JOIN                  |            | 14   | 630  |      4 (25)| 00:00:01 |
|  5 |    TABLE ACCESS FULL         | PRACOWNICY | 14   | 630  |      3  (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

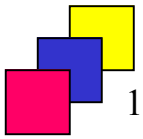
Proces optymalizacji polecenia SQL

1. Przetransformowanie polecenia do mniej kosztownej postaci.
2. Wygenerowanie zbioru potencjalnych planów dla polecenia SQL.
3. Oszacowanie kosztu wykonania każdego planu na podstawie tzw. statystyk.
4. Wybór do realizacji planu z najniższym kosztem wykonania.



Transformator polecenia

- **Przekształca polecenie do mniej kosztownej postaci.**
- **Polecenia: oryginalne i po transformacji, są semantycznie identyczne.**
- **Przykładowe operacje:**
  - **Scalanie perspektyw (ang. view merging).**
  - **Przesuwanie predykatów pomiędzy blokami polecenia (ang. predicate pushing).**
  - **Zastąpienie podzapytania połączeniem (ang. subquery unnesting).**
  - **Eliminacja połączeń.**
  - **Zastąpienie zapytania z warunkami z operatorami OR zapytaniami, połączonymi operatorami UNION ALL (ang. OR expansion).**
  - **Zastąpienie zapytań z operatorami MINUS i INTERSECT przez połączenie.**



Przykłady transformacji

```
SELECT * FROM pracownicy
WHERE etat = 'PROFESOR
OR etat = 'ASYSTENT';
```

```
SELECT * FROM pracownicy WHERE etat = 'PROFESOR'
UNION ALL
SELECT * FROM pracownicy WHERE etat = 'ASYSTENT';
```

```
CREATE VIEW bogaci_v AS SELECT * FROM pracownicy WHERE placa_pod > 10000;
```

```
SELECT nazwisko, nazwa
FROM zespoly
     NATURAL JOIN bogaci_v ;
```

```
SELECT nazwisko, nazwa
FROM zespoly NATURAL JOIN pracownicy
WHERE placa_pod > 10000;
```

```
SELECT nazwisko
FROM bogaci_v
WHERE etat = 'ASYSTENT';
```

```
SELECT nazwisko
FROM (SELECT * FROM pracownicy
      WHERE placa_pod > 10000 AND etat = 'ASYSTENT');
```

```
SELECT nazwisko
FROM pracownicy
WHERE id_zesp IN
(SELECT id_zesp
 FROM zespoly
 WHERE nazwa = 'ALGORYTMY');
```

```
SELECT nazwisko
FROM pracownicy JOIN zespoly USING(id_zesp)
WHERE nazwa = 'ALGORYTMY';
```

Generator planów wykonania polecenia

- Tworzy zbiór różnych planów wykonania tego samego polecenia.
- Plany wykonania różnią przez:
  - zastosowanie różnych kombinacji ścieżek dostępu,
  - zastosowanie różnych algorytmów połączenia relacji,
  - inną kolejność łączenia relacji.
- Dla każdego planu następuje oszacowanie kosztu wykonania.
- Moment zakończenia generacji zbioru planów:
  - Jeśli koszt aktualnego planu jest wysoki, generator szuka lepszego planu bardziej intensywnie (rozpatruje więcej alternatywnych planów).
  - Jeśli koszt aktualnego planu jest niski, wówczas generator szybko kończy poszukiwania z uwagi na małe prawdopodobieństwo poprawy.

Estymator kosztu planu wykonania

- Zajmuje się szacowaniem kosztu plany wykonania polecenia, wyliczając tzw. miary.
- Zbiór rekordów – wynik wykonania operacji w planie: relacja, perspektywa, wynik operacji połączenia, wynik działania operatora GROUP BY.
- Miary, pozwalające na oszacowanie **całkowitego kosztu planu**:
  - **Selektywność** – ułamek reprezentujący liczbę rekordów odczytanych ze zbioru przez operację w stosunku do liczby rekordów w zbiorze. Ściśle związana z warunkami, zdefiniowanymi w poleceniu SQL. Zakres: <0, 1>, selektywność 0 – nie zostały odczytane żadne rekordy, selektywność 1 – zostały odczytane wszystkie rekordy ze zbioru.
  - **Liczność** – liczba rekordów, odczytanych ze zbioru.
  - **Koszt** – reprezentuje jednostki pracy lub zasoby użyte do realizacji operacji w planie wykonania, np. liczba operacji we/wy, użycie procesora, użycie pamięci.

Wybór celu optymalizacji (1)

- **Najlepsza przepustowość – wykorzystanie jak najmniejszych zasobów systemowych do uzyskania wszystkich rekordów polecenia SQL.**
  - **Dla aplikacji wykonywanych wsadowo, np. drukowanych raportów.**
  - **Najważniejszy krótki czas zakończenia całego zadania, mniej ważny czas odpowiedzi.**
- **Najkrótszy czas odpowiedzi – wykorzystanie jak najmniejszych zasobów do uzyskania pierwszego rekordu polecenia SQL.**
  - **Dla aplikacji interaktywnych, np. formularzy ekranowych, zapytań w SQL*Plus.**
  - **Najważniejszy krótki czas odpowiedzi – użytkownik chce jak najszybciej zobaczyć pierwszy rekord (lub kilka pierwszych rekordów) polecenia.**

Wybór celu optymalizacji (2)

- **Dla bieżącej sesji – parametr OPTIMIZER_MODE:**
  - **ALL_ROWS – optymalizacja maksymalizująca przepustowość**
  - **FIRST_ROWS_n – optymalizacja minimalizująca łączny czas odczytania pierwszych n krotek (n może być równe (1,10,100 lub 1000)**

```
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS_1;
```

- **Przy użyciu wskazówki (dla konkretnego polecenia SQL):**

```
SELECT /* + ALL_ROWS */ nazwisko FROM ...
```

Zmienne wiązania w poleceniu SQL (1)

- **Pozwalają na „sparametryzowanie” polecenia**

```
SQL> variable zespol number;
SQL> exec :zespol := 10
Procedura PL/SQL została zakończona pomyślnie.
SQL> print :zespol
ZESPOL
------
    10
SQL> SELECT count(*) FROM PRACOWNICY WHERE id_zesp = :zespol;
COUNT(*)
--------
       2
SQL> exec :zespol := 20
Procedura PL/SQL została zakończona pomyślnie.

SQL> SELECT count(*) FROM PRACOWNICY WHERE id_zesp = :zespol;
COUNT(*)
--------
       7
```

Zmienne wiązania w poleceniu SQL (2)

- **Umożliwiają wielokrotne użycie tego samego planu wykonania przy kolejnych wywołaniach polecenia z różnymi wartościami zmiennej wiązania – tzw. „współdzielenie kursora” (domyślne działanie)**
- **Przy pierwszym wywołaniu polecenia ze zmienną wiązania optymalizator „spogląda” na wartość zmiennej celem wygenerowania optymalnego planu**
- **Problem – kolejne wywołania tego samego polecenia z innymi wartościami dla zmiennej wiązania mogą przetwarzać dane o innej charakterystyce niż te z pierwszego wywołania**
- **Rozwiązanie – optymalizator obserwuje kolejne wywołania i podejmuje decyzje, czy dla kolejnego wywołania polecenia z inną wartością zmiennej wiązania wygenerować nowy plan**
- **Efekt – być może wiele planów wykonania dla tego samego polecenia**