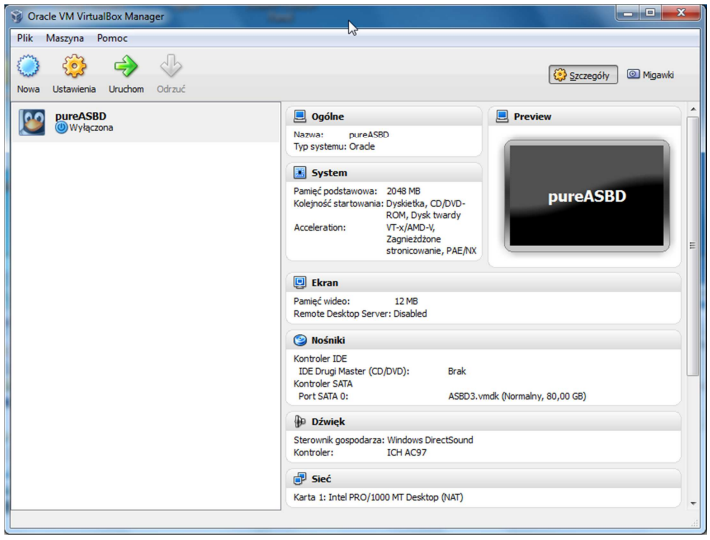
Ćwiczenie 5.
Strojenie poleceń SQL

1. Uruchomienie i skonfigurowanie środowiska do ćwiczeń

Czas trwania: 20 minut

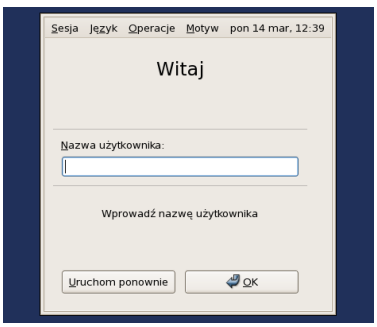
Zadaniem niniejszych ćwiczeń jest przedstawienie podstawowych zagadnień dotyczących strojenia poleceń SQL. Pierwsze ćwiczenia przygotowują środowisko, a następnie koncentrują się na metodach uzyskiwania informacji dotyczącej sposobu wykonywania (planów) poleceń SQL.

1. Uruchom środowisko wirtualizacji – kliknij na umieszczoną na pulpicie komputera-gospodarza ikonę *Oracle VM VirtualBox*.



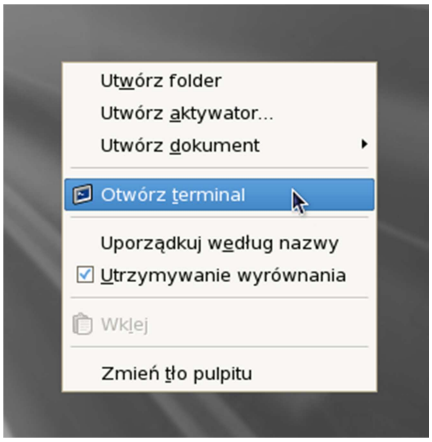
2. Spróbuj uruchomić maszynę wirtualną. W tym celu zaznacz w lewym panelu środowiska pozycję ***pureASBD*** i naciśnij umieszczony na pasku narzędzi przycisk **Uruchom** (możesz również wybrać pozycję **Uruchom** z menu kontekstowego, dostępnego po kliknięciu prawym klawiszem myszy na pozycji ***pureASBD***).

3. Po pomyślnym uruchomieniu maszyny wirtualnej powinien zostać wyświetlony ekran logowania (jeśli ekran maszyny wirtualnej nie zajmuje całego ekranu komputera-gospodarza, użyj odpowiedniego skrótu klawiszowego aby to zmienić).



4. Zaloguj się do systemu operacyjnego maszyny wirtualnej jako użytkownik *oracle* z hasłem *oracle*. Podaj powyższe informacje i naciśnij przycisk **OK**.

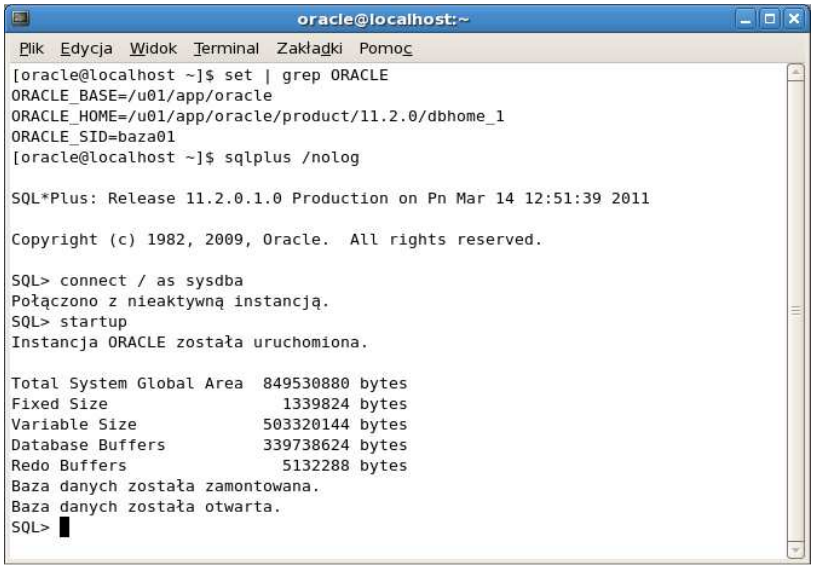
5. Uruchom terminal. Można to wykonać przez kliknięcie prawym klawiszem myszki na pulpit i wybranie z menu kontekstowego pozycji **Otwórz terminal**.



6. Sprawdź wartości zmiennych środowiskowych dotyczących instancji *Oracle*. Wykorzystaj w tym celu polecenie: `set | grep ORACLE`

7. Jeśli zmienne środowiskowe są ustawione poprawnie, uruchom program *sqlplus*. Wykorzystaj opcję *nolog*: `sqlplus /nolog`

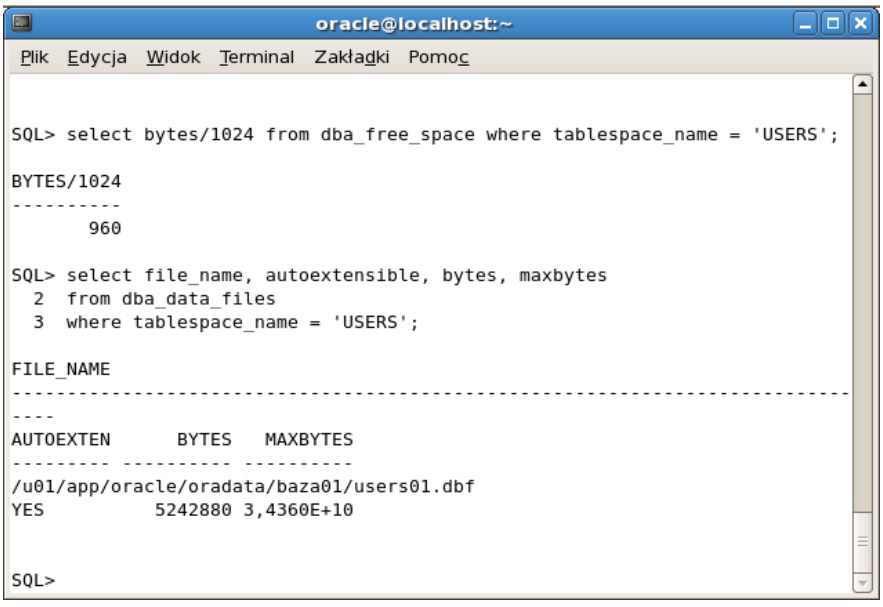
8. Następnie zaloguj się korzystając z autoryzacji użytkownika administracyjnego przez system operacyjny. Wykonaj w tym celu polecenie: `connect / as sysdba`
   Następnie uruchom bazę danych poleceniem `startup`



9. Sprawdź, czy jest jeszcze wolne miejsce w przestrzeni tabel USERS. Jeśli miejsce jest bliskie wyczerpania, sprawdź czy plik implementujący przestrzeń tabel USERS ma aktywną własność automatycznego zwiększania rozmiaru, a także jaką ma wielkość aktualną oraz maksymalną. Wykorzystaj w tym celu polecenia:

```
select bytes/1024 from dba_free_space
where tablespace_name = 'USERS';

select file_name, autoextensible, bytes, maxbytes
from dba_data_files
where tablespace_name = 'USERS';
```



10. Jeśli własność automatycznego zwiększania rozmiaru jest nieaktywna lub maksymalna wielkość przekracza 50MB wykonaj poniższe polecenie korygując te własności.

```
alter database datafile
'/u01/app/oracle/oradata/baza01/users01.dbf'
autoextend on next 5M maxsize 50M;
```

```
SQL> alter database datafile '/u01/app/oracle/oradata/baza01/users01.dbf'
  2  autoextend on next 5M maxsize 50M;

Baza danych została zmieniona.
```

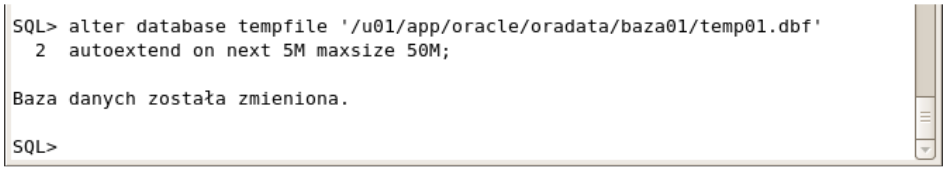
11. Odczytaj nazwę tymczasowej przestrzeni tabel w Twojej bazie danych, a następnie sprawdź: czy plik implementujący tymczasową przestrzeń tabel ma aktywną własność automatycznego zwiększania rozmiaru, a także jaką ma wielkość aktualną oraz maksymalną. Wykorzystaj w tym celu polecenia:

```
select tablespace_name
from dba_tablespaces
where contents = 'TEMPORARY';

select file_name, autoextensible, bytes, maxbytes
from dba_temp_files join dba_tablespaces using (tablespace_name)
where contents = 'TEMPORARY';
```

12. Jeśli własność automatycznego zwiększania rozmiaru jest nieaktywna lub maksymalna wielkość przekracza 50MB wykonaj poniższe polecenie korygując te własności.

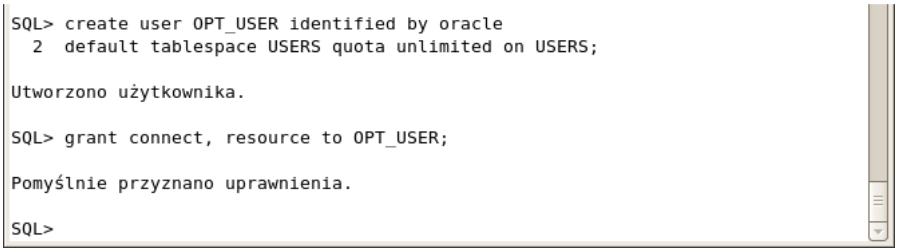
```
alter database tempfile
'/u01/app/oracle/oradata/baza01/temp01.dbf'
autoextend on next 5M maxsize 50M;
```



13. Ćwiczenia, wykonamy pod nowym użytkownikiem. W tym celu utwórz konto użytkownika OPT_USER, w schemacie którego będą wykonywane ćwiczenia. Domyślną przestrzenią tabel użytkownika powinna być przestrzeń USERS. Nadaj użytkownikowi prawa przyłączania się do bazy danych i tworzenia obiektów. Wykonaj następujące polecenia

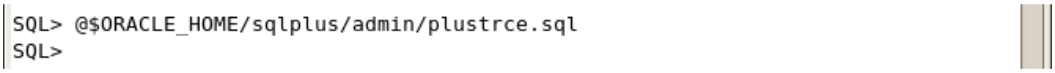
```
create user OPT_USER identified by oracle
default tablespace USERS quota unlimited on USERS;

grant connect, resource to OPT_USER;
```



14. Utwórz rolę PLUSTRACE wykonując skrypt `plustrce.sql`, zlokalizowany w katalogu `$ORACLE_HOME/sqlplus/admin`.

```
@$ORACLE_HOME/sqlplus/admin/plustrce.sql
```



15. Nadaj użytkownikowi OPT_USER rolę PLUSTRACE.

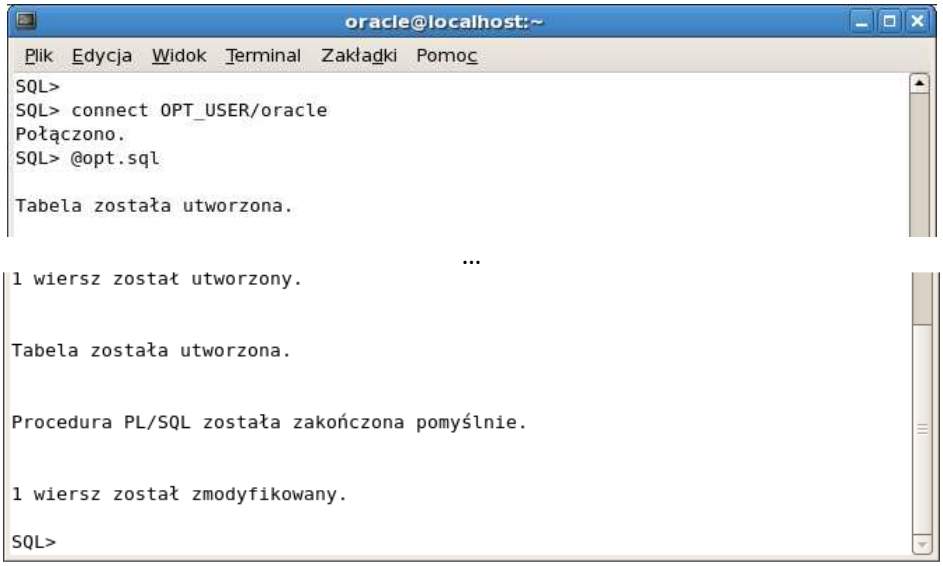
```
grant PLUSTRACE to OPT_USER;
```

16. Zaloguj się jako nowoutworzony użytkownik: `connect OPT_USER/oracle`

17. Pobierz do katalogu domowego użytkownika oracle skrypt `opt.sql`, który utworzy zestaw danych potrzebnych do ćwiczeń. W tym celu:

    a. Uruchom przeglądarkę i wejdź na stronę studium z materiałami do ćwiczeń z administracji: http://tpd.cs.put.poznan.pl/studia-podyplomowe/studium-sbd/materialy/adm/.
    b. W sekcji „Materiały uzupełniające” znajdź link do pliku `opt.sql`
    c. Z menu kontekstowego wybierz *Zapisz element docelowy jako…*
    d. Zapisz plik do katalogu domowego użytkownika *oracle*

18. Przełącz się na uprzednio otwarty terminal i uruchom skrypt `opt.sql`. Skorzystaj w tym celu z polecenia: `@opt.sql`



19. Zatwierdź wprowadzone zmiany poleceniem `commit`.

20. Sprawdź strukturę i liczbę rekordów utworzonych przez skrypt relacji PRAC i ZESP. Wykorzystaj do tego celu następujące polecenia:

```
desc PRAC;

select count(*) from PRAC;

desc ZESP;

select count(*) from ZESP;

select INDEX_NAME
from user_indexes
where TABLE_NAME in ('ZESP','PRAC');

select BYTES, BLOCKS, EXTENTS, SEGMENT_NAME
from user_segments
where SEGMENT_NAME in ('PRAC','ZESP');
```

Wypełnij poniższą tabelę:

Własność	**Tabela PRAC**	**Tabela ZESP**
Liczba wierszy		
Wielkość w bajtach		
Liczba bloków		
Liczba rozszerzeń		
Liczba indeksów		

2. Wyświetlanie planów zapytań

Czas trwania: 15 minut

Przystąpimy teraz do przeglądu technik pozwalających wydobyć informacje na temat planów zapytań.

1. Na początku, w celu wygenerowania a następnie wyświetlenia planu zapytań, wykorzystamy dyrektywę `explain plan` oraz skrypt `utlxpls.sql`.

2. Wygeneruj plan poniższego zapytania, używając polecenia `explain plan`, nadaj wygenerowanemu planowi identyfikator „`p1`”.

```
select nazwa, count(*) from zesp natural join prac
group by nazwa
order by count(*) desc;
```

W tym celu wykonaj następujące polecenie:

```
explain plan
set statement_id = 'p1' for
select nazwa, count(*) from zesp natural join prac
group by nazwa
order by count(*) desc;
```

3. Odczytaj plan zapytania umieszczonego w tabeli PLAN_TABLE, używając kolejno:
   a. zapytania:

```
select operation,object_name,id,cost,parent_id
from plan_table where statement_id='p1' order by id;
```

   w tym celu na początku sformatuj sposób wyświetlania kolumn wykonując następujące polecenia:

```
column operation format a20
column object_name format a15
```

   a następnie wykonaj zaplanowane polecenie:

```
select operation, object_name, id, cost,parent_id
from plan_table where statement_id='p1' order by id;
```

b. skryptu `$ORACLE_HOME/rdbms/admin/utlxpls.sql`

```
@$ORACLE_HOME/rdbms/admin/utlxpls.sql
```

4. Porównaj oba sposoby i uzyskane za ich pomocą plany zapytań.

```
SQL> select operation,object_name,id,cost,parent_id
  2  from plan_table where statement_id='p1' order by id
  3  ;

OPERATION            OBJECT_NAME              ID       COST  PARENT_ID
-------------------- --------------- ---------- ---------- ----------
SELECT STATEMENT                              0         21
SORT                                          1         21          0
HASH                                          2         21          1
HASH JOIN                                     3         19          2
TABLE ACCESS         ZESP                     4          3          3
TABLE ACCESS         PRAC                     5         15          3
```

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxpls.sql

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1250917565

-------------------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |     5 |   100 |    21  (15)| 00:00:01 |
|   1 |  SORT ORDER BY       |      |     5 |   100 |    21  (15)| 00:00:01 |
|   2 |   HASH GROUP BY      |      |     5 |   100 |    21  (15)| 00:00:01 |
|*  3 |    HASH JOIN         |      | 10000 |   195K|    19   (6)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| ZESP |     5 |    85 |     3   (0)| 00:00:01 |
|   5 |     TABLE ACCESS FULL| PRAC | 10000 | 30000 |    15   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("ZESP"."ID_ZESP"="PRAC"."ID_ZESP")
```

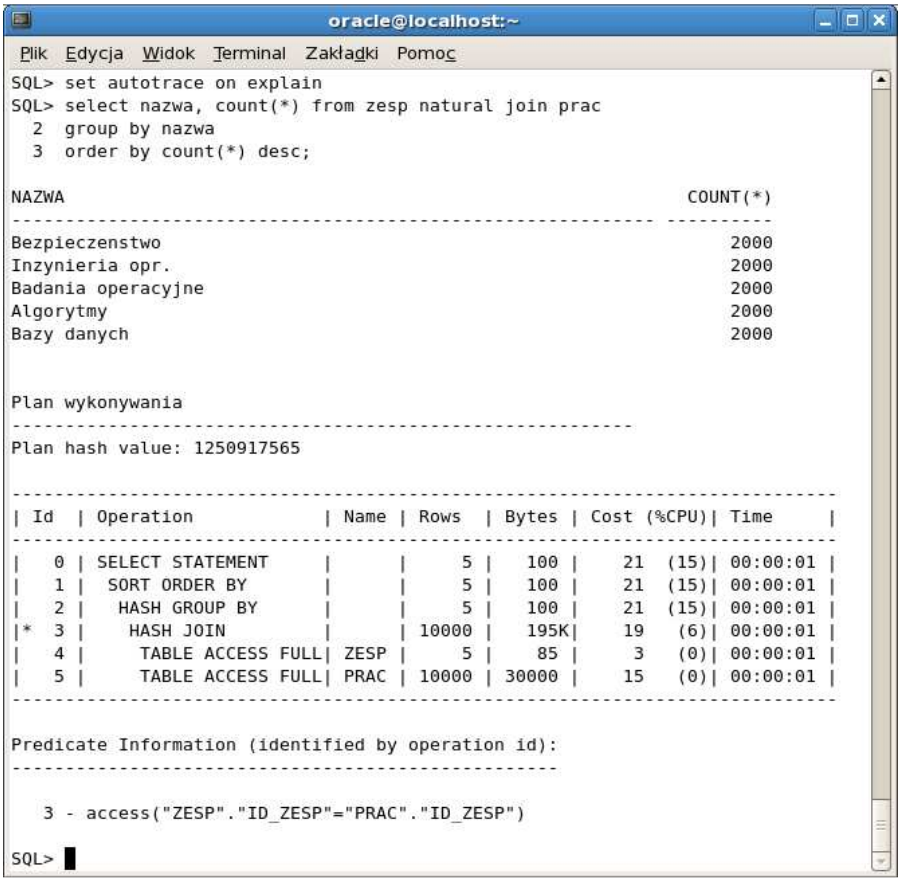
- Czy w obu przypadkach plan wykonania polecenia jest taki sam?
- Jaki jest planowany koszt wykonania przeanalizowanego polecenia?
- Który ze sposobów jest Twoim zdaniem prostszy?

5. Innym ze sposobów generowania i wyświetlania planów zapytań jest wykorzystanie dyrektywy programu sqlplus: `SET AUTOTRACE ON EXPLAIN`.

6. Wykonaj wspomnianą dyrektywę a następnie wywołaj polecenie, dla którego plan zostanie automatycznie wygenerowany i wyświetlony.

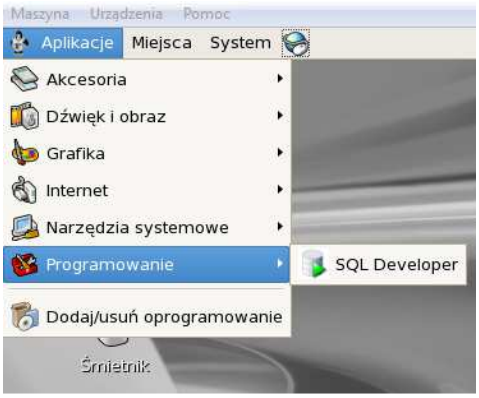
```
SET AUTOTRACE ON EXPLAIN

select nazwa, count(*) from zesp natural join prac
group by nazwa
order by count(*) desc;
```

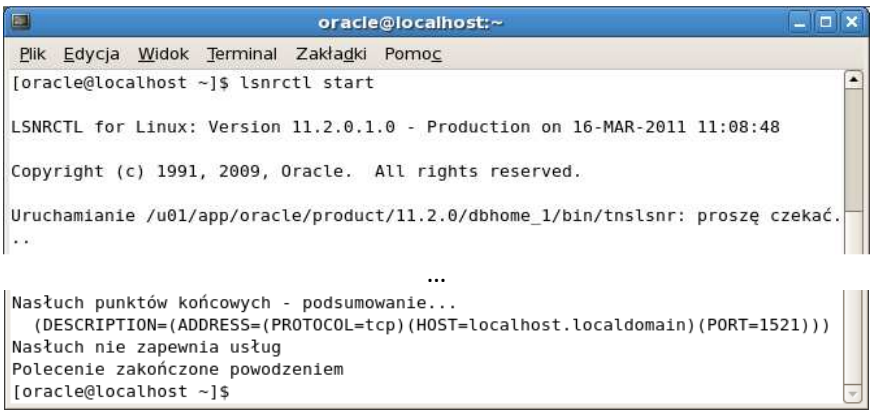


7. Znaczna część oprogramowania pozwalającego na łączenie się z bazą danych i wykonywanie poleceń SQL, posiada mechanizmy pozwalające na wyświetlanie planów wykonywania zapytań. Nie jest w tym wyjątkiem oprogramowanie *SQL Developer*.

8. Uruchom oprogramowanie *SQL Developer* wybierając z menu głównego systemu: *Aplikacje →Programowanie →SQL Developer*.

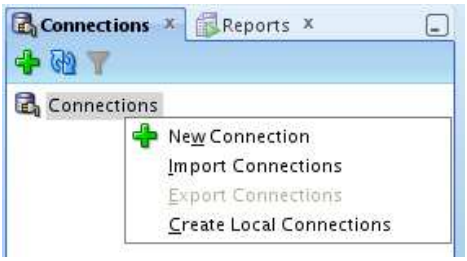


9. *SQL Developer* funkcjonuje jako klient bazy danych i do połączenia z nią wymaga aktywnego procesu nasłuchowego. Aby uruchomić proces nasłuchowy uruchom terminal tekstowy a następnie wystartuj proces nasłuchowy poleceniem: `lsnrctl start`.

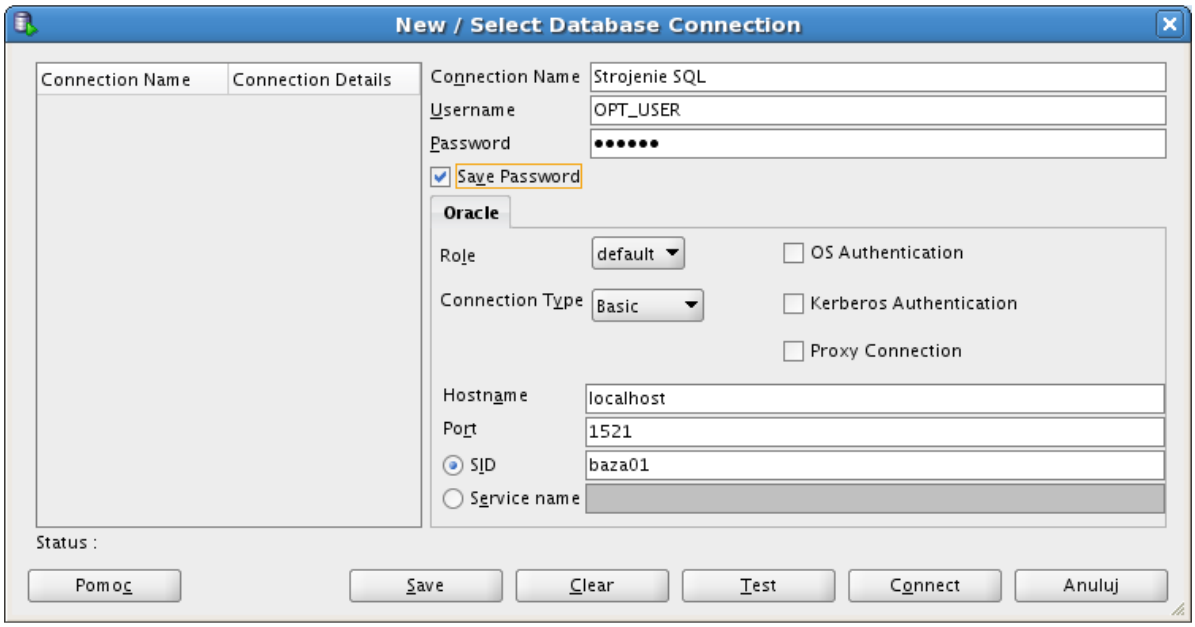


10. Zwróć uwagę na port, na którym nasłuchuje proces nasłuchowy. Przystąpimy teraz do skonfigurowania połączenia z bazą danych. W tym celu:

    a. Przełącz się z powrotem do *SQL Developera*
    b. Kliknij prawym klawiszem myszy na węźle *Connections*, a następnie wybierz z menu kontekstowego opcję *New Connections*.

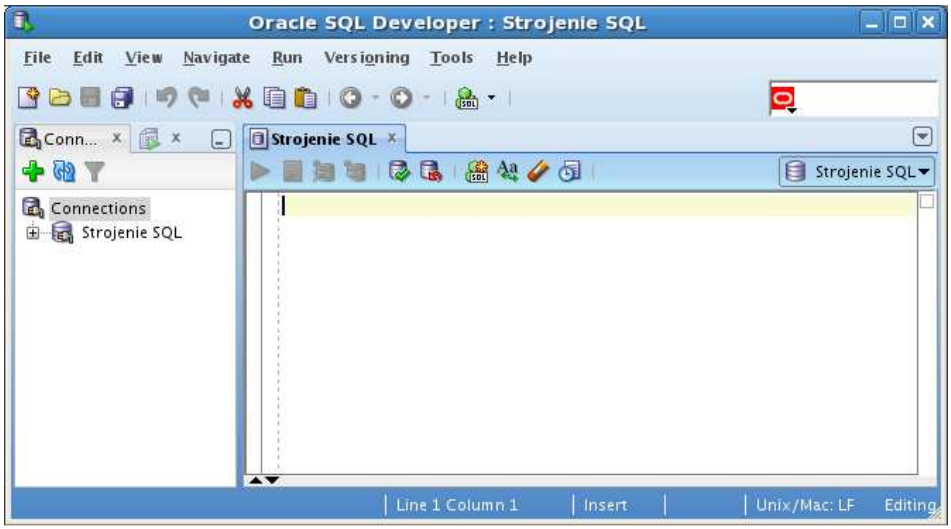


    c. W formatce pozwalającej na definicję nowego połączenia wpisz następujące wartości
        i. Connection Name: `Strojenie SQL`
        ii. Username: `OPT_USER`
        iii. Password: `oracle`
        iv. Save Password: `zaznaczone`
        v. Hostname: `localhost`
        vi. Port: `1521`
        vii. SID: `baza01`



d. Sprawdź definicję przyciskiem **Test**. I jeśli wszystko zostało zdefiniowane poprawnie (status połączenia posiada wartość *Success*) zapisz połączenie za pomocą przycisku **Save**.

e. Połącz się z bazą danych korzystając z przycisku **Connect**.



11. W pole edycji poleceń SQL wprowadź poniższe polecenie.

```
select nazwa, count(*) from zesp natural join prac
group by nazwa
order by count(*) desc;
```

12. SQL Developer posiada cztery przyciski pozwalające w różny sposób uruchamiać polecenia SQL.

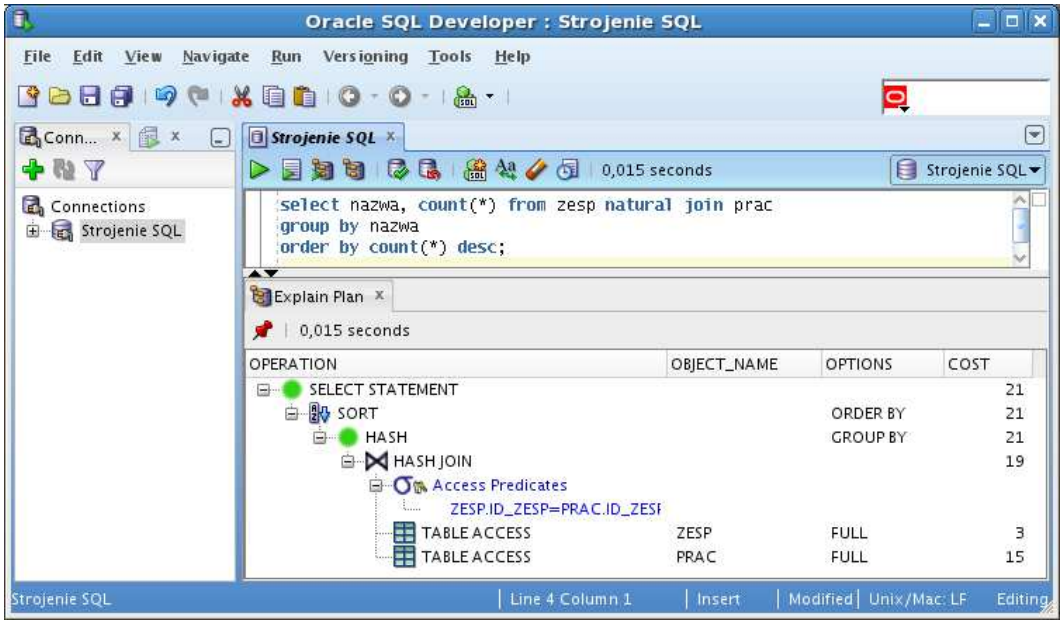
a.  – standardowe uruchamianie pojedynczych poleceń SQL. Wyniki wyświetlanie są w postaci interaktywnej tabeli.

b.  – standardowe uruchamianie serii poleceń SQL (np. skryptów). Wyniki wyświetlanie są w postaci tekstowej.

c.  – szczegółowe wyświetlenie planu oraz wykorzystania zasobów dla polecenia SQL – opcja *autotrace*. Opcja ta wymaga dodatkowych uprawnień.

d.  – wyświetlenie planu dla polecenia SQL – opcja *explain plan*.

13. Korzystając z opcji *explain plan* wyświetl plan dla naszego polecenia.



- Czy plan uzyskany w ten sposób jest tym samym planem, który otrzymaliśmy poprzednio?

W kolejnych ćwiczeniach w celu wykonywania poszczególnych poleceń, a także wyświetlania planów będziemy korzystali z narzędzia *SQL Developer*.

3. Metody dostępu

Czas trwania: 30 minut

Kolejne ćwiczenia będą służyły przedstawieniu w praktyczny sposób mechanizmów związanych z przetwarzaniem zapytań SQL i ich strojeniem. Zaczynamy od przeglądu różnych metod dostępu do tabel i indeksów.

1. Wyjaśnij plan poniższego zapytania.

```
select rowid, nazwisko, plec, placa
from prac
where id_prac = 900;
```

- Jaką metodą dostępu do tabeli PRAC wybrał optymalizator do wykonania tego zapytania?
- Dlaczego?
- Nie było innego rozwiązania?
- Wykonaj powyższe zapytanie i zanotuj wartość adresu rekordu (ROWID).

2. Zdefiniuj zapytanie, które odczyta te same dane. Tym razem posłuż się w tym celu odczytanym przed chwilą adresem rekordu (ROWID). Wyświetl plan tego zapytania.

```
select rowid, nazwisko, plec, placa
from prac
where rowid = 'AAASNIAAEAAAAIWAC8';
```

Pamiętaj aby zamienić powyższą wartość adresu rekordu na odczytaną przez Ciebie.

- Jaka metoda dostępu do danych została użyta?

3. Sprawdź w słowniku bazy danych, jakie indeksy założono na relacji PRAC. Wykonaj w tym celu poniższe polecenie.

```
select index_name
from user_indexes
where table_name = 'PRAC';
```

4. Brak indeksów na tabeli w praktyce determinuje pełne skanowanie (FULL SCAN) tabeli podczas dostępu do jej zawartości. W przypadku małych tabel nie jest to sytuacja wymagająca reakcji, jednak w przypadku tabel o dużej wielkości może to prowadzić do niskiej wydajności zapytań.
Aby poprawić wydajność analizowanego zapytania utwórz indeks B-drzewo o nazwie PRAC_IDX na atrybucie ID_PRAC relacji PRAC (na atrybucie wykorzystywanym podczas selekcji).

```
create index prac_idx on prac(id_prac);
```

5. Następnie ponownie wyjaśnij plan naszego zapytania (z punktu 1).

```
select rowid, nazwisko, plec, placa
from prac
where id_prac = 900;
```

- Czym różni się otrzymany plan?
- Jaka metoda dostępu do tabeli została wykorzystana?
- Jaka metoda dostępu do indeksu została wykorzystana?
- O ile zmniejszył się koszt zapytania?

6. Usuń indeks PRAC_IDX. Następnie na atrybucie ID_PRAC relacji PRAC zdefiniuj klucz podstawowy o nazwie PRAC_PK.

```
drop index prac_idx;

alter table prac add constraint prac_pk primary key(id_prac);
```

7. Ponownie sprawdź w słowniku bazy danych indeksy dla relacji PRAC. Wyświetl dodatkowo nazwy poindeksowanych atrybutów.

```
select c1.index_name, uniqueness, column_name, column_position
from user_indexes c1, user_ind_columns c2
where c1.table_name = 'PRAC'
and c1.index_name = c2.index_name;
```

- Dlaczego na tej tabeli PRAC istnieje założony indeks?
- Czy będzie on przydatny dla naszego zapytania (z punktu 1)?

8. Ponownie wyjaśnij plan naszego zapytania.

```
select rowid, nazwisko, plec, placa
from prac
where id_prac = 900;
```

- Jaki jest tym razem plan tego zapytania?
- Czy Twoje oczekiwania się sprawdziły?
- Jaka metoda dostępu do tabeli została wykorzystana?
- Jaka metoda dostępu do indeksu została wykorzystana?
- Dlaczego w tym przypadku metoda dostępu do indeksu była inna?

9. Utwórz indeks typu B-drzewo o nazwie PRAC_NAZWISKO_IDX na atrybucie NAZWISKO relacji PRAC.

```
create index prac_nazwisko_idx on prac(nazwisko);
```

10. Następnie dla każdego z poniższych zapytań, wyjaśnij jego plan. Czy potrafisz przewidzieć fakt użycia indeksu oraz metodę dostępu do niego?

```
select * from prac
where nazwisko = 'Prac155';

select * from prac
where nazwisko like 'Prac155%';

select * from prac
where nazwisko like '%Prac155%';
```

- W których zapytaniach optymalizator nie użył indeksu?
- Jakie były metody dostępu do indeksu?

11. Usuń indeks PRAC_NAZWISKO_IDX i na jego miejsce utwórz skonkatenowany indeks typu B-drzewo o nazwie PRAC_NAZW_PLACA_IDX na atrybutach NAZWISKO i PLACA relacji PRAC.

```
drop index prac_nazwisko_idx;

create index prac_nazw_placa_idx on prac(nazwisko, placa);
```

12. Następnie dla każdego z poniższych zapytań, wyjaśnij jego plan.

```
select count(*) from prac
where nazwisko like 'Prac1%';

select count(*) from prac
where nazwisko like 'Prac1%' and placa > 100;

select count(*) from prac
where placa > 100;
```

- Jakich metod dostępu do indeksu optymalizator użył dla każdego powyższych zapytań?
- Z czego wynika wybór różnych metod dostępu?
- Dlaczego plany zapytań nie uwzględniają w ogóle tabeli PRAC?

13. Utwórz indeks typu B-drzewo o nazwie PRAC_PLEC_IDX na atrybucie PLEC relacji PRAC.

```
create index prac_plec_idx on prac(plec);
```

14. Następnie sprawdź plan dla poniższego polecenia.

```
select count(*) from prac where plec='M'
and id_prac between 100 and 110;
```

15. Sprawdź, czy dla relacji PRAC zebrano statystyki. Skorzystaj z poniższych zapytań.

```
select table_name, last_analyzed, num_rows
from user_tables
where table_name='PRAC';

select column_name, num_distinct,
low_value, high_value, num_buckets
from user_tab_columns
where table_name = 'PRAC';

select index_name, last_analyzed, num_rows
from user_indexes where table_name='PRAC';

select * from user_tab_histograms
where table_name='PRAC';
```

16. Zbierz statystyki dla relacji PRAC.

```
begin
  dbms_stats.gather_table_stats(
    ownname=>'OPT_USER', tabname=>'PRAC');
end;
```

17. Następnie ponownie wykonaj zapytania sprawdzające zebrane statystyki.

```
select table_name, last_analyzed, num_rows
from user_tables
where table_name='PRAC';

select column_name, num_distinct,
low_value, high_value, num_buckets
from user_tab_columns
where table_name = 'PRAC';

select index_name, last_analyzed, num_rows
from user_indexes where table_name='PRAC';

select * from user_tab_histograms
where table_name='PRAC';
```

- Jakie rodzaje statystyk zgromadziło wywołanie procedury `dbms_stats.gather_table_stats`?

18. Utwórz indeks typu B-drzewo o nazwie PRAC_CZY_ETAT_IDX na atrybucie CZY_ETAT relacji PRAC. Zbierz statystyki dla tego indeksu.

```
create index prac_czy_etat_idx on prac(czy_etat)
compute statistics;
```

19. Następnie wyświetl plan poniższego zapytania.

```
select count(*) from prac where czy_etat='T' and plec='K';
```

- Czy umiesz wytłumaczyć plan tego zapytania?
- Na czym polegają konwersje występujące w tym planie?
- Zanotuj koszt wykonania tego zapytania.

20. Usuń indeksy na atrybutach PLEC i CZY_ETAT.

```
drop index prac_plec_idx;
drop index prac_czy_etat_idx;
```

21. Ponownie wyświetl plan poniższego zapytania.

```
select count(*) from prac where czy_etat='T' and plec='K';
```

- Czy uzyskany plan był przez Ciebie przewidziany?
- Zanotuj koszt wykonania tego zapytania.

22. Utwórz indeksy bitmapowe na relacji PRAC, na atrybutach PLEC i CZY_ETAT, o nazwach odpowiednio PRAC_PLEC_BMP_IDX i PRAC_CZY_ETAT_BMP_IDX. Zbierz statystyki dla obu indeksów.

```
create bitmap index prac_plec_bmp_idx on prac(plec)
compute statistics;

create bitmap index prac_czy_etat_bmp_idx on prac(czy_etat)
compute statistics;
```

23. Ponownie wyświetl plan poniższego zapytania.

```
select count(*) from prac where czy_etat='T' and plec='K';
```

- Zanotuj koszt wykonania tego zapytania.
- Który z trzech planów dla tego zapytania był najmniejszy?

24. Usuń indeks skonkatenowany Utwórz indeks typu B-drzewo o nazwie PRAC_PLACA_IDX na atrybucie PLACA relacji PRAC. Zbierz statystyki dla tego indeksu.

```
drop index prac_nazw_placa_idx;

create index prac_placa_idx on prac(placa) compute statistics;
```

25. Sprawdź plany następujących zapytań:

```
select nazwisko from prac where placa = 2;

select nazwisko from prac where ROUND(placa) = 2;

select nazwisko from prac where placa-1 = 2;
```

- Dla którego z powyższych zapytań optymalizator nie przewidział użycia indeksu?
- Z czego to wynika?
- Czy któreś z powyższych zapytań można zoptymalizować – nie zmieniając wyniku zapytania i jego logiki, doprowadzić do użycia indeksu?

26. Utwórz indeks funkcyjny o nazwie PRAC_PLACA_FUN_IDX na relacji PRAC, który będzie używany dla zapytania o zaokrągloną wartość płacy pracownika.

```
create index prac_placa_fun_idx on prac(round(placa));
```

27. Wyświetl ponownie plan dla poniższego polecenia.

```
select nazwisko from prac where ROUND(placa) = 2;
```

- Czy tym razem optymalizator przewidział użycie indeksu?

4. Sortowanie

Czas trwania: 10 minut

Poniższe ćwiczenie ma na celu ilustracje faktu, iż optymalizator wykorzystuje operację sortowanie (lub czasami bardziej optymalnego haszowania, np. gdy porządek nie jest istotny) nie tylko w przypadku występowania klauzuli ORDER BY w zapytaniu SQL. Ma to istotne znaczenie ze względu na fakt iż operacja sortowania, w szczególności dużych wolumenów danych, jest stosunkowo kosztowna.

1. Porównaj plany wykonania następujących zapytań.

```
select * from prac;

select * from prac order by id_prac;

select * from prac order by id_prac desc;

select * from prac order by nazwisko;

select distinct nazwisko from prac;

select nazwisko from prac group by nazwisko;
```

- Porównaj koszt pierwszego i trzech następnych zapytań. Czy można mówić o wielokrotnym wzroście kosztu w przypadku poleceń korzystających z sortowania?
- W przypadku których poleceń optymalizator uniknął sortowania korzystając z indeksu?
- Czy zauważasz powód, dla którego w przypadku dwóch ostatnich zapytań operacja SORT została zastąpiona operacją HASH?

5. Połączenia

Czas trwania: 15 minut

Większość poleceń wykonywanych w relacyjnych bazach danych to polecenia, które wykorzystują połączenia dwóch lub większej liczby tabel. Optymalizacja tego typu poleceń wymaga znajomości metod łączenia tabel, warunków, które muszą dla każdej z metod być spełnione.
Poniższe ćwiczenia koncentrują się na tej tematyce.

1. Sprawdź plan wykonania następujących zapytań z połączeniem naturalnym.

```
select * from zesp natural join prac
where nazwa = 'Algorytmy';

select * from prac natural join zesp
where nazwa = 'Algorytmy';
```

- Jaka metoda połączenia została wykorzystana?
- Dlaczego w każdym przypadku tabela ZESP jest tabelą zewnętrzną (na której budowana jest tablica haszowa)?

2. Zdefiniuj klucz podstawowy na atrybucie ID_ZESP relacji ZESP. Dodaj także klucz obcy oraz indeks na atrybucie połączeniowym ID_ZESP w tabeli PRAC.

```
alter table zesp add constraint zesp_pk primary key(id_zesp);

alter table prac add constraint prac_zesp_fk
  foreign key(id_zesp) references zesp(id_zesp);

create index prac_id_zesp_idx on prac(id_zesp);
```

3. W związku z tym, że tabele wykorzystywane przez nas są relatywnie małe, koszt wykonywania na niech operacji w pamięci jest niski. Dlatego też mimo utworzonych indeksów zapytania z poprzedniego punktu będą najprawdopodobniej używały metody połączeń HASH JOIN, dla której indeksy nie pełnią istotnej roli.
   Aby to zmienić dokonamy modyfikacji parametru odpowiadającego za sposób wyliczania kosztu użycia indeksów. Zmniejszymy go do 30% standardowego kosztu. Wykonaj polecenie:

```
alter session set optimizer_index_cost_adj = 30;
```

4. Sprawdź ponownie plan wykonania zapytania.

```
select * from prac natural join zesp
where nazwa = 'Algorytmy ';
```

   - Jaka metoda połączenia tym razem została wykorzystana?
   - Która tabela jest tabelą zewnętrzną?

5. Powróć z parametrem wykorzystywanym do wyznaczania kosztów użycia indeksów do standardowej wartości. Wykonaj polecenie:

```
alter session set optimizer_index_cost_adj = 100;
```

6. Sprawdź tym razem plan wykonania zapytania.

```
select p.nazwisko, count(*) ilu_wiecej
from   prac p join prac w on (p.placa < w.placa)
group by p.nazwisko;
```

   - Jaka metoda połączenia tym razem została wykorzystana?
   - Dlaczego?

7. Jednym z dość kontrowersyjnych i stosunkowo rzadko wykorzystywanych indeksów jest bitmapowy indeks połączeniowy. Wykorzystywany jest on praktycznie tylko i wyłącznie w hurtowniach danych. Postaramy się poniżej zaobserwować jego wykorzystanie.
   Na początku utworzymy bitmapowy indeks połączeniowy na tabeli PRAC. Będzie on dotyczył jednak nie atrybutu z tabeli PRAC a atrybutu NAZWA z tabeli ZESP.

```
create bitmap index join_ind on prac(nazwa)
from prac p, zesp z
where p.id_zesp=z.id_zesp;
```

8. Sprawdźmy teraz plan następującego zapytania.

```
select count(*) from prac natural join zesp
where nazwa = 'Algorytmy';
```

- Czy w planie zapytania w ogóle pojawia się połączenie?
- Która tabela nie była w ogóle potrzebna do wykonania tego zapytania?

6. Statystyki, histogramy

Czas trwania: 20 minut

Dla optymalizatora kosztowego kluczową kwestią podczas opracowywania planu zapytania są aktualne i dokładne statystyki dotyczące tabel, atrybutów i indeksów, a także statystyki systemowe. Z tego względu kilka ćwiczeń związanych z tym tematem nie może zabraknąć.

1. Na początek usuń statystyki dla tabeli PRAC. Skorzystaj w tym celu z procedury `dbms_stats.delete_table_stats`. Wykonaj polecenie:

```
begin
  dbms_stats.delete_table_stats(
     ownname=>'OPT_USER',tabname=>'PRAC');
end;
```

2. Następnie dokonaj oszacowania statystyk dla tabeli PRAC na podstawie próbki 10%. W tym celu wykonaj poniższe polecenie.

```
begin
  dbms_stats.gather_table_stats (
     ownname=>'OPT_USER',tabname=>'PRAC',estimate_percent=>10);
end;
```

3. Sprawdź informacje o statystykach dla tabeli PRAC.

```
select table_name, last_analyzed, num_rows
from user_tables
where table_name='PRAC';

select column_name, num_distinct,
low_value, high_value, num_buckets, histogram
from user_tab_columns
where table_name = 'PRAC';

select index_name, last_analyzed, num_rows
from user_indexes where table_name='PRAC';

select * from user_tab_histograms
where table_name='PRAC';
```

- Czy potrafisz wskazać statystyki, które w wyniku estymacji zostały wyznaczone błędnie?

4. Sprawdzimy teraz wpływ histogramów na proces optymalizacji polecenia SQL. Na początek poznaj rozkład wartości atrybutu PLACA_DOD w tabeli PRAC.

```
select placa_dod, count(*)
from prac group by placa_dod;
```

5. Utwórz indeks typu B-drzewo o nazwie PRAC_PLACA_DOD_IDX na atrybucie PLACA_DOD tabeli PRAC.

```
create index prac_placa_dod_idx on prac(placa_dod);
```

6. Usuń ponownie wszystkie statystyki dla tabeli PRAC (również statystyki dla atrybutów i indeksów). Sprawdź czy rzeczywiście wszystkie statystyki zostały usunięte (patrz punkt 3.).

```
begin
  dbms_stats.delete_table_stats(
     ownname=>'OPT_USER',tabname=>'PRAC');
end;
```

7. Wyświetl plany dla poniższych zapytań.

```
select * from prac where placa_dod = 100;

select * from prac where placa_dod = 999;
```

- Czy oba plany są identyczne?
- Czy znasz mechanizm, który doprowadził do wykorzystania dwóch różnych planów?
- Jaki to mechanizm?
- Podpowiedź: w sytuacji braku statystyk optymalizator wykonuje dynamiczne próbkowanie statystyk. Ten mechanizm pozwala zebrać brakujące statystyki bezpośrednio przed procesem optymalizacji polecenia SQL.

8. Zbierz ponownie dla tabeli PRAC statystyki razem z histogramami dla kolumn tabeli. Kolejnym poleceniem usuń histogram dla kolumny PLACA_DOD tabeli PRAC. Spowoduje to, że optymalizator, budując plany wykonania, nie będzie znał dokładnego rozkładu wartości w tej kolumnie.

```
BEGIN
  DBMS_STATS.GATHER_TABLE_STATS(ownname=>'OPT_USER', tabname=>'PRAC');
  DBMS_STATS.DELETE_COLUMN_STATS(ownname=>'OPT_USER', tabname=>'PRAC',
             colname=>'PLACA_DOD', col_stat_type=>'HISTOGRAM');
END;
```

9. Dla pewności sprawdź, czy istnieje histogram dla kolumny PLACA_DOD tabeli PRAC.

```
SELECT num_distinct, low_value, high_value, num_buckets, histogram
FROM   user_tab_col_statistics
WHERE  table_name = 'PRAC' AND column_name = 'PLACA_DOD';
```

10. Wykonaj ponownie poniższe polecenia. Tym razem optymalizator ma dostęp do statystyk (nie musi posługiwać się dynamicznym próbkowaniem statystyk), jednak bez histogramów (czyli nie zna dokładnego rozkładu kolumny PLACA_DOD, według której następuje selekcja rekordów w zapytaniach).

```
select * from prac where placa_dod = 100;

select * from prac where placa_dod = 999;
```

- Czy tym razem plany są takie same?
- Który z planów nie jest wydajnym rozwiązaniem?

11. Uzupełnij statystyki dla tabeli PRAC o histogram kolumny PLACA_DOD.

```
begin
  dbms_stats.gather_table_stats(
    ownname=>'OPT_USER',tabname=>'PRAC',
    method_opt=>'FOR COLUMNS placa_dod SIZE AUTO');
end;
```

12. Sprawdź informacje o statystykach dotyczących dla atrybutu PLACA_DOD tabeli PRAC.

```
select column_name, num_distinct,
low_value, high_value, num_buckets, histogram
from user_tab_columns
where table_name = 'PRAC';

select * from user_tab_histograms
where table_name='PRAC';
```

- Z jakim typem histogramu mamy do czynienia?
- Czy dane zebrane w histogramie są przybliżone czy dokładne?

13. Wyświetl ponownie plany poniższych zapytań.

```
select * from prac where placa_dod = 100;

select * from prac where placa_dod = 999;
```

- Czy tym razem oba plany są identyczne?

7. Wskazówki

Czas trwania: 25 minut

Często w danych przetwarzanych za pomocą Systemów Baz Danych występują zależności, które nie są możliwe do odkrycia przez optymalizator nawet, gdy zbiór statystyk jest zarówno dokładny jak i pełny. Efektem tego mogą być plany zapytań dalekie od optymalnych.
Jednocześnie często zależności te są znane twórcom zapytań. Mogą oni wykorzystać swoją wiedzę do „poinformowania" optymalizator o sugerowanych metodach wykonywania zapytań. Możliwość taka jest dostępna za pomocą tzw. wskazówek.

1. Usuń indeks bitmapowy PRAC_PLEC_BMP_IDX, założony na atrybucie PLEC tabeli PRAC.

```
drop index PRAC_PLEC_BMP_IDX;
```

2. Następnie utwórz na tym samym atrybucie indeks B-drzewo o nazwie PRAC_PLEC_IDX.

```
create index prac_plec_idx on prac(plec);
```

3. Wskazówki wykorzystywane są do modyfikacji różnorodnych aspektów związanych z planami zapytań. Na początek wykorzystamy je do modyfikacji metod dostępu. Sprawdź plan poniższych zapytań.

```
select count(*) from prac where id_prac < 100 and plec = 'K';

select /*+INDEX (prac prac_plec_idx)*/ count(*)
from prac where id_prac < 100 and plec = 'K';
```

   - Czy wskazówka w drugim zapytaniu zmieniła postać plany zapytania?
   - Na czy polega różnica pomiędzy planami obu zapytań?

4. Ponownie to samo zapytanie, tym razem wymusimy wykonanie go za pomocą operacji pełnego przeglądnięcia tabeli. Sprawdź plan zapytania:

```
select /*+NO_INDEX (prac)*/ count(*)
from prac where id_prac < 100 and plec = 'K';
```

   - Czy wskazówka została uwzględniona podczas opracowania planu zapytania?

5. Wskazówki mogą być wykorzystane także do modyfikowania kolejności łączenia tabel.
   Aby móc przeprowadzić to ćwiczenie uzupełnimy nasz schemat o dodatkowe trzy tabele. W tym celu wykonaj „Skrypt tworzący obiekty" ze strony Studium Podyplomowego (Materiały do przedmiotu Systemy Baz Danych).
   Obecnie adres do tego skryptu jest następujący:
   http://admlab2.cs.put.poznan.pl/staff/materialy/Systemy%20Baz%20Danych/public/Pldemobld.sql
   Możesz skopiować zawartość całego skryptu do SQL Developera i uruchomić polecenia tam zawarte za pomocą przycisku  .
   W efekcie w schemacie powinny się pojawić dodatkowe tabele PRACOWNICY, ETATY i ZESPOLY.

6. Sprawdź plan wykonania poniższego zapytania:

```
select *
from pracownicy join etaty
     on placa_pod between placa_min and placa_max
     join zespoly using(id_zesp);
```

- Jaka jest kolejność połączeń poszczególnych tabel?
- Jakie typy połączeń zostały wykorzystane?

7. Następnie dodaj do zapytania wskazówkę ORDERED. Sprawdź plan wykonania poniższego zapytania:

```
select /*+ ORDERED */ *
from pracownicy join etaty
     on placa_pod between placa_min and placa_max
     join zespoly using(id_zesp);
```

- Czy plan uległ zmianie?
- Czy kolejność połączeń uległa zmianie?

8. Zmień kolejność relacji w klauzuli FROM i ponownie sprawdź plan zapytania.

```
select /*+ ORDERED */ *
from pracownicy join zespoly using(id_zesp)
     join etaty
     on placa_pod between placa_min and placa_max;
```

- Czy kolejność połączeń uległa zmianie?
- Który z planów jest planem o mniejszym koszcie?

9. Nieco mniej „agresywną” wskazówką wpływającą na kolejność połączeń jest LEADING. Sprawdź plan zapytania:

```
select /*+ LEADING(etaty) */ *
from pracownicy join zespoly using(id_zesp)
     join etaty
     on placa_pod between placa_min and placa_max;
```

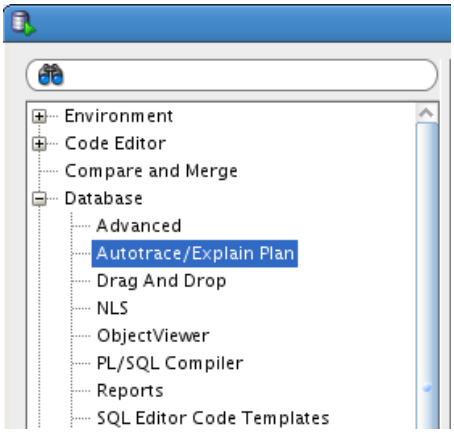
- Czy kolejność połączeń uległa zmianie?
- Jak nazywają się wykorzystane metody połączeń?
- Czy znasz je wszystkie?

10. Na zakończenie zadań dotyczących wskazówek wykorzystamy je do optymalizacji zapytania. Optymalizowanym zapytaniem będzie zapytanie (proszę nie „poprawiać” zapytania):

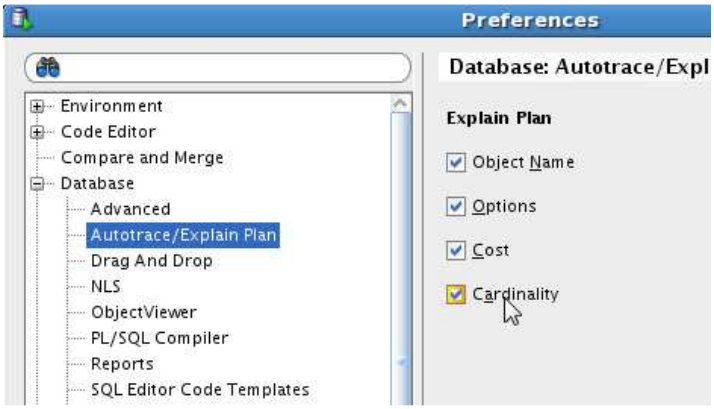
```
select *
from pracownicy p join zespoly z
     on (p.id_zesp = z.id_zesp+1)
     join etaty e
     on (etat = e.nazwa
         and placa_pod between placa_min and placa_max);
```

a. Jedną z podstawowych reguł rządzących strojeniem poleceń SQL jest reguła „śnieżnej kuli”. Ostateczny wynik zapytania jest określony, jednak sposób dochodzenia do wyniku może być różny. Ważne aby przez cały czas wykonywania zapytania operować na jak najmniejszym zbiorze danych. Dla przykładu, należy początkowo operować na jak najmniejszych tabelach, stosować na początku połączenia i selekcje o jak największej selektywności aby w efekcie tych pośrednich działań powstawały małe zbiory danych, które będą podlegały dalszemu przetwarzaniu.
Optymalizator nie zawsze właściwie ocenia selektywność połączeń lub złożonych warunków i dlatego nie potrafi wygenerować wydajnego zapytania – w którym na każdym etapie przetwarzany będzie jak najmniejszy zbiór danych.

Aby ocenić decyzje podejmowane przez optymalizator musimy znać estymowaną przez niego selektywność poszczególnych operacji. W ty celu uzupełnimy domyślnie wyświetlany plan o kolumnę *Cardinality*. Informującą nas o tym jaka jest estymowana przez optymalizator liczba krotek przetwarzanych na danym etapie zapytania.
W tym celu wybierz menu *Tools* w *SQL Developerze*. Następnie kliknij w *Preferences.*
Po prawej stronie rozwiń węzeł *Database* i zaznacz *Autotrace/Explain Plan*.



Zaznacz opcję *Cardinality* w obszarze *Explain Plan*.



Zatwierdź zmiany przyciskiem **OK**.

b. Aby dać optymalizatorowi szansę opracowania najlepszego możliwego planu dokonajmy wyliczenia statystyk dla tabel używanych w zapytaniu. Wykonaj poniższe polecenie.

```
begin
  dbms_stats.gather_table_stats (
     ownname=>'OPT_USER',tabname=>'PRACOWNICY');
  dbms_stats.gather_table_stats (
     ownname=>'OPT_USER',tabname=>'ZESPOLY');
  dbms_stats.gather_table_stats (
     ownname=>'OPT_USER',tabname=>'ETATY');
end;
```

c. Wyświetl plan dla naszego optymalizowanego zapytania.

```
select *
from pracownicy p join zespoly z
     on (p.id_zesp = z.id_zesp+1)
     join etaty e
     on (etat = e.nazwa
         and placa_pod between placa_min and placa_max);
```

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			10	8
HASH JOIN			10	8
Access Predicates				
MERGE JOIN			6	6
TABLE ACCESS	ETATY	BY INDEX ROWID	2	6
Filter Predicates				
INDEX	PK_ETAT	FULL SCAN	1	6
FILTER				
Filter Predicates				
SORT		JOIN	4	14
Access Predicates				
Filter Predicates				
TABLE ACCESS	PRACOWNICY	FULL	3	14
TABLE ACCESS	ZESPOLY	FULL	3	5

- Zwróć uwagę na estymowaną liczbę rekordów podczas odczytu poszczególnych tabel. Czy jest ona prawidłowa?
- Czy prawidłowa jest liczba wierszy powstałych w wyniku połączenia tabeli ETATY i PRACOWNICY? Są one łączone jako pierwsze, dopiero wynik tego połączenia łączony jest z tabelą ZESPOLY.
- Ile wierszy zwróci całe zapytanie?
- Czy jest to wartość zgodna z estymowaną?
- Który warunek połączeniowy posiada większą selektywność?

d. Sprawdź czy poprawnie wyznaczana jest selektywność połączenia pomiędzy tabelą PRACOWNICY i ZESPOLY (składowej konstrukcji naszego optymalizowanego zapytania). W tym celu sprawdź plan zapytania:

```
select *
from pracownicy p join zespoly z
     on (p.id_zesp = z.id_zesp+1);
```

OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			7	18
HASH JOIN			7	18
Access Predicates				
P.ID_ZESP=Z.ID_ZESP+1				
TABLE ACCESS	ZESPOLY	FULL	3	5
TABLE ACCESS	PRACOWNICY	FULL	3	14

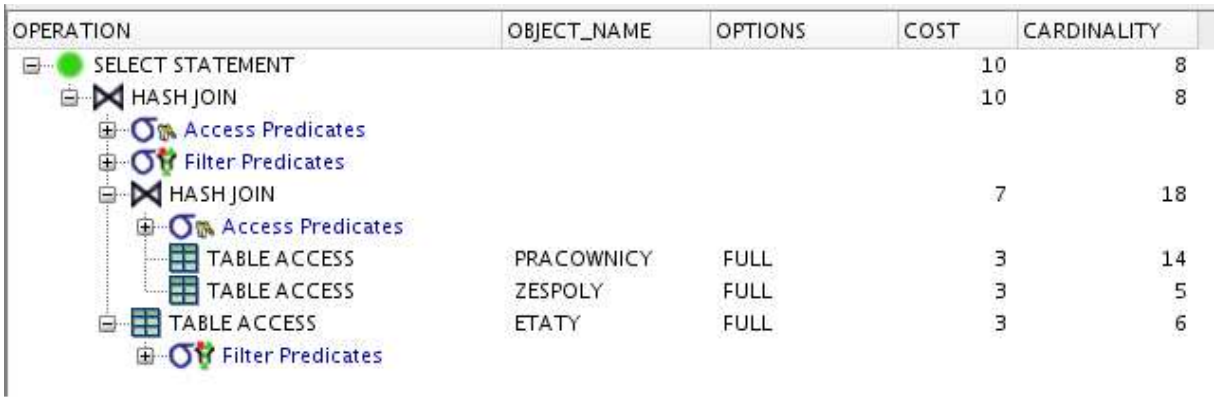
- Ile wierszy zwraca powyższe zapytanie?
- Czy jest to wartość zgodna z estymowaną?

Takie błędy w estymowanych selektywności warunków (połączeniowych i selekcji) są sygnałem, że programista może wiele zyskać „sugerując" pewne rozwiązania optymalizatorowi.

e. Samodzielnie zastanów się jaka wskazówka powinna pomóc wykonać poniższe optymalizowane zapytanie w sposób efektywny.

```
select *
from pracownicy p join zespoly z
     on (p.id_zesp = z.id_zesp+1)
     join etaty e
     on (etat = e.nazwa
         and placa_pod between placa_min and placa_max);
```

- Czy udało Ci się uzyskać poniższy plan?



OPERATION	OBJECT_NAME	OPTIONS	COST	CARDINALITY
SELECT STATEMENT			10	8
HASH JOIN			10	8
Access Predicates				
Filter Predicates				
HASH JOIN			7	18
Access Predicates				
TABLE ACCESS	PRACOWNICY	FULL	3	14
TABLE ACCESS	ZESPOLY	FULL	3	5
TABLE ACCESS	ETATY	FULL	3	6
Filter Predicates				

Uwaga! Reguła „śnieżnej kuli” to oczywiście nie jedyna zasada rządząca procesem strojenia zapytań.

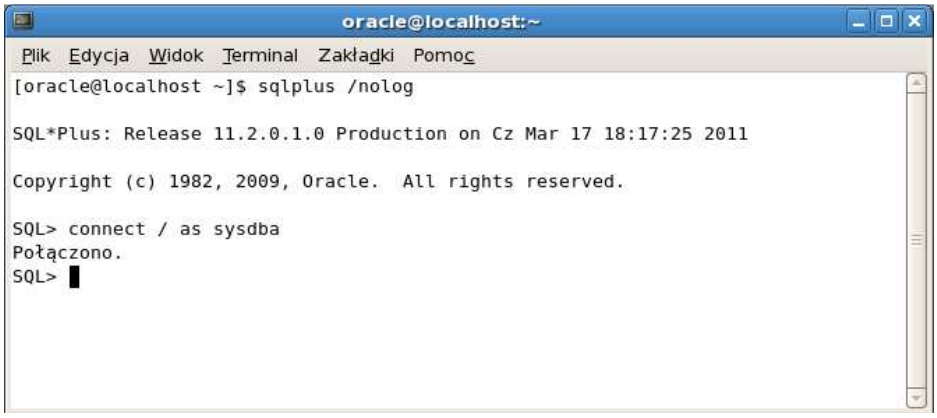
8. SQL*Trace

Czas trwania: 20 minut

Strojenie poleceń SQL nie powinno dotyczyć wszystkich zapytań. Większość zapytań funkcjonuje w sposób bardzo poprawny. Istotne jest aby wyłapywać te zapytania, które sprawiają problemy – są wykonywane bardzo często i/lub w sposób nieefektywny – i koncentrować siły na strojenie tylko tych zapytań.

Jednym ze sposobów uzyskania informacji dotyczących problematycznych zapytań jest narzędzie *SQL*Trace*.

1. Uruchom terminal tekstowy. Uruchom *SQL*Plus*. Zaloguj się jako użytkownik administracyjny.

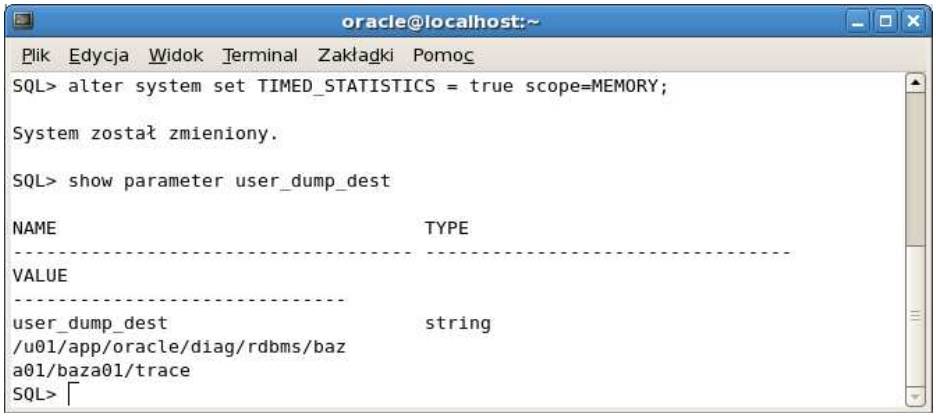


2. Ustaw parametr `TIMED_STATISTICS = true`.

```
alter system set TIMED_STATISTICS = true scope = MEMORY;
```

3. Sprawdź wartość zmiennej USER_DUMP_DEST, wskazującej katalog, w którym zostanie umieszczony plik śladu.

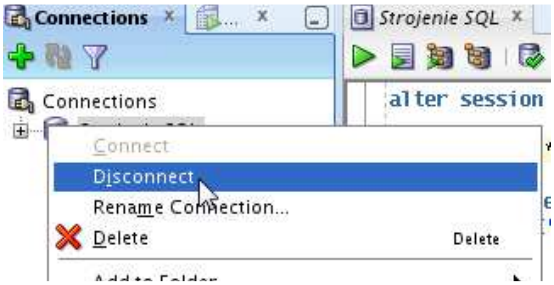
```
show parameter user_dump_dest
```



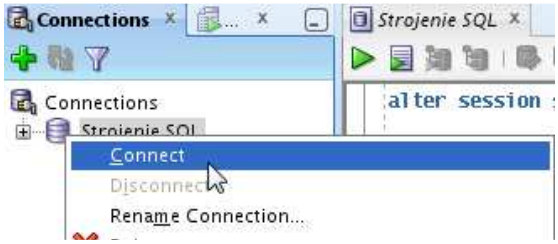
4. Nadaj prawa użytkownikowi OPT_USER do wykonywania polecenia `alter session`.

```
grant alter session to OPT_USER;
```

5. Przełącz się do z powrotem narzędzia SQL Developer. Rozłącz sesję.



6. Połącz się ponownie aby dla nowej sesji zastosować zmieniony parametr `TIMED_STATISTICS`.



7. Uruchom narzędzie *SQL*Trace*, wykonaj kilka zapytań, następnie wyłącz *SQL*Trace*. Wykonaj kolejno następujące polecenia.

```
alter session set sql_trace=true;

select count(*) from prac group by id_zesp;

update prac set placa=placa+34
where plec='K' and czy_etat='T' and id_prac < 100;

alter session set sql_trace=false;
```

8. Czas na przeanalizowanie wygenerowanego pliku śladu. Na początku musimy go zlokalizować. Uruchom kolejny terminal (lub skorzystaj już z otwartego). I na poziomie systemu operacyjnego wykonaj polecenie.

```
ls -al -St /u01/app/oracle/diag/rdbms/baza01/baza01/trace | more
```

9. Przerwij wykonywanie polecenia klawiszem **q**. Pierwszy z wyświetlonych plików będzie naszym plikiem śladu.

```
[oracle@localhost ~]$ ls -al -St /u01/app/oracle/diag/rdbms/baza01/baza01/trace |more
razem 1148
drwxr-x---  5 oracle oinstall 12288 mar 17 18:26 .
-rw-r-----  1 oracle oinstall 25298 mar 17 18:26 baza01_ora_30361.trc
-rw-r-----  1 oracle oinstall   157 mar 17 18:26 baza01_ora_30361.trm
-rw-r-----  1 oracle oinstall 58741 mar 17 18:19 alert_baza01.log
```

10. Użyj programu *TKProf* do sformatowania wyniku, który znajduje się w pliku śladu.

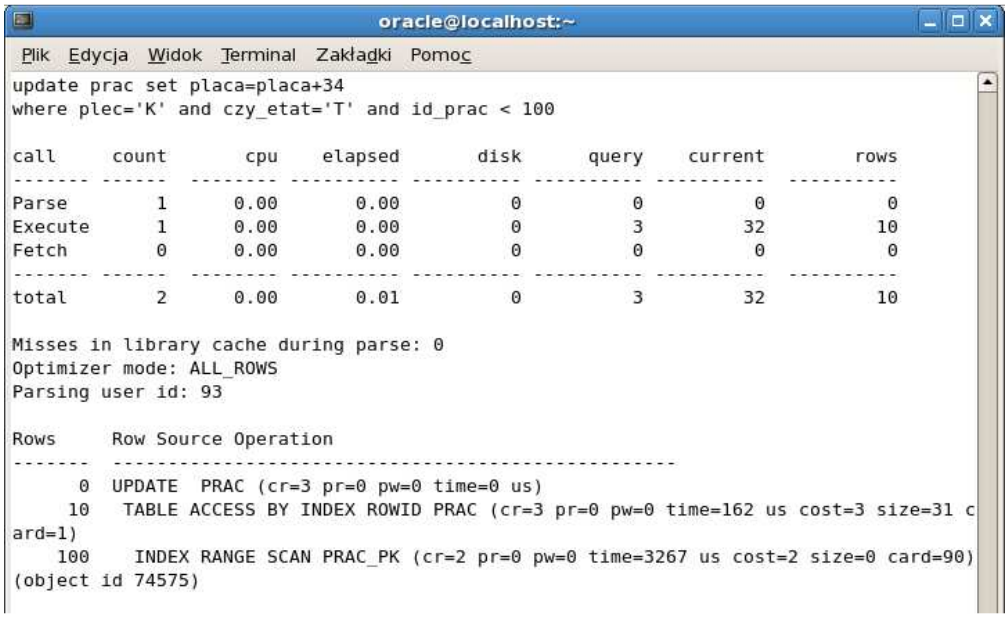
```
tkprof <plik_sladu> plik.txt aggregate=yes sys=no
```

dla przykładu:

```
tkprof
   /u01/app/oracle/diag/rdbms/baza01/baza01/trace/baza01_ora_30361.trc
   plik.txt aggregate=yes sys=no
```

11. Obejrzyj wynik zawarty w pliku raportu. Możesz w tym celu użyć polecenia:

```
more plik.txt
```



12. Włączanie narzędzia *SQL*Trace* może odbywać się także dla obcych sesji – np. dla działającej w danym momencie aplikacji. Załóżmy, że naszą aplikacją będzie *SQL Developer*. Aby z zewnątrz uruchomić generowanie śladu dla tej aplikacji musimy znać informacje na temat wykorzystywanej przez nią sesji.

13. Ponownie rozłącz się i połącz w programie *SQL Developer* aby zainicjować nową sesję.
Powróć do narzędzia *SQL*Plus*, w którym jesteś zalogowany jako użytkownik administracyjny. odczytaj z perspektywy v$session dane o sesji użytkownika OPT_USER.

```
select sid, serial# from v$session
where username = 'OPT_USER';
```

14. Następnie uruchom w sposób niewidoczny dla użytkownika OPT_USER generowanie pliku śladu dla jego poczynań poniższym poleceniem (w miejsce sid i serial# wpisz wartości uzyskane w poprzednim zapytaniu):

```
exec dbms_system.set_sql_trace_in_session(<sid>,<serial#>,true);
```

dla przykładu:

```
exec dbms_system.set_sql_trace_in_session(34,1347,true);
```

15. Przełącz się do programu SQL Developer i zasymuluj działanie aplikacji wykonując poniższe polecenia. Wykonaj każde z nich po 10 razy.

```
select count(*) from prac;
```

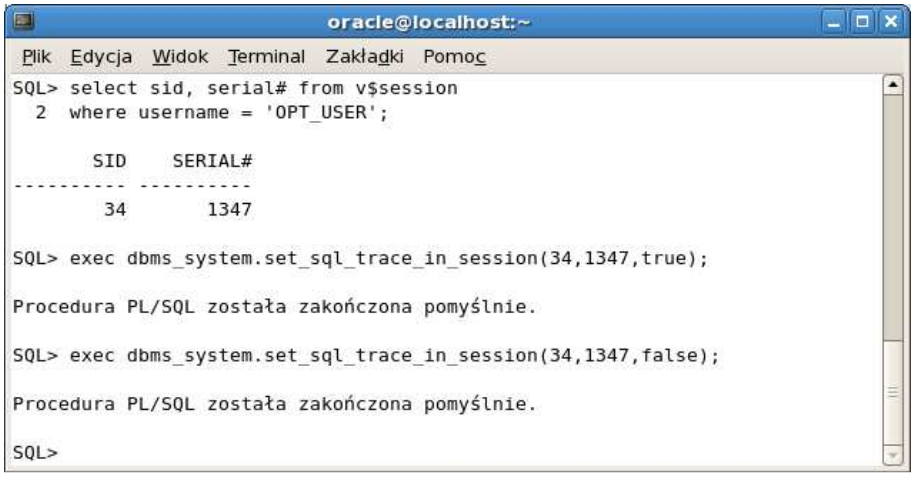
```
select max(placa) from prac;
```

16. Powróć do narzędzia SQL*Plus, w którym jesteś zalogowany jako użytkownik administracyjny. Zakończ śledzenie sesji użytkownika OPT_USER poleceniem:

```
exec dbms_system.set_sql_trace_in_session(<sid>,<serial#>,false);
```

Pamiętaj o wstawieniu odpowiedzich wartości w miejsce sid i serial#. Dla przykładu:

```
exec dbms_system.set_sql_trace_in_session(34,1347,false);
```



17. Ponownie odszukaj na poziomie systemu operacyjnego plik śladu wykonując polecenie.

```
ls -al -St /u01/app/oracle/diag/rdbms/baza01/baza01/trace | more
```

18. Przerwij wykonywanie polecenia klawiszem **q**. Pierwszy z wyświetlonych plików będzie naszym plikiem śladu.

19. Użyj programu *TKProf* do sformatowania wyniku, który znajduje się w pliku śladu.

```
tkprof <plik_sladu> plik.txt aggregate=yes sys=no
```

dla przykładu:

```
tkprof
   /u01/app/oracle/diag/rdbms/baza01/baza01/trace/baza01_ora_698.trc
   plik.txt aggregate=yes sys=no
```

20. Obejrzyj wynik zawarty w pliku raportu.

```
more plik.txt
```

- Czy potrafisz znaleźć w nim informacje na temat obu wykonanych w 'aplikacji' poleceń?
- Które z tych dwóch poleceń wykonało się szybciej?
- Które z tych dwóch poleceń użyło indeksu?

9. Top 10 SQL

Czas trwania: 20 minut

Na zakończenie wykorzystamy narzędzie *Enterprise Manager* w celu znalezienia problematycznych poleceń SQL.

1. W terminalu, na poziomie systemu operacyjnego uruchom konsolę programu *Enterprise Manager*.

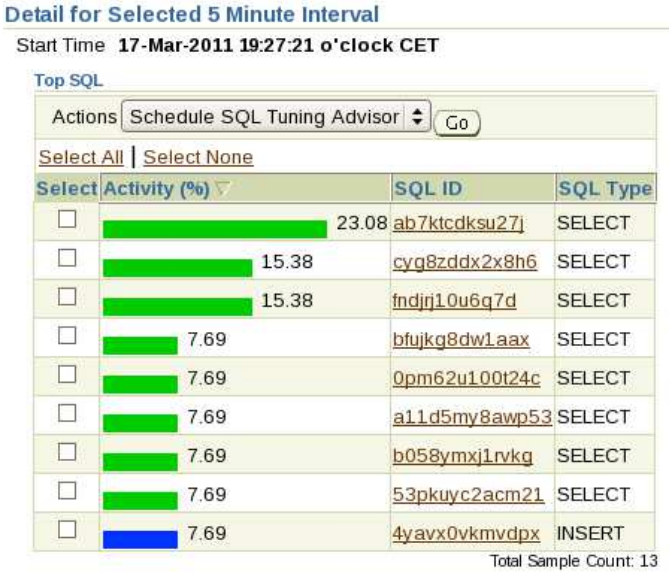
```
emctl start dbconsole
```

2. Uruchom przeglądarkę i zaloguj się pod adresem: `https://localhost:1158/em/`

3. Przejdź do zakładki *Performance*

4. Kliknij w link *Top Activity*

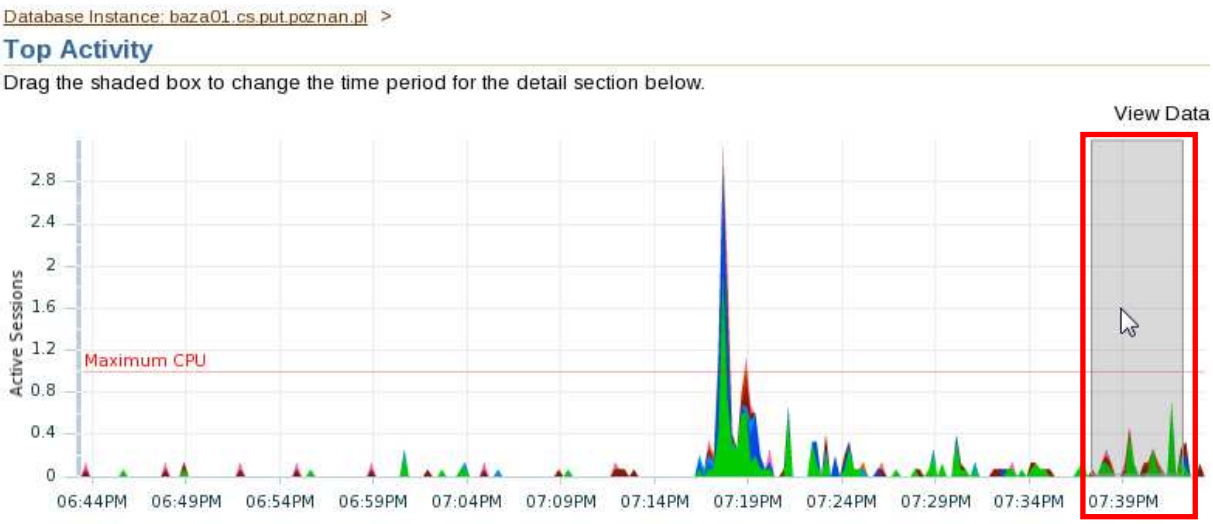
5. Przejdź do obszaru dotyczącego 'problematycznych' poleceń SQL – *Top SQL*.



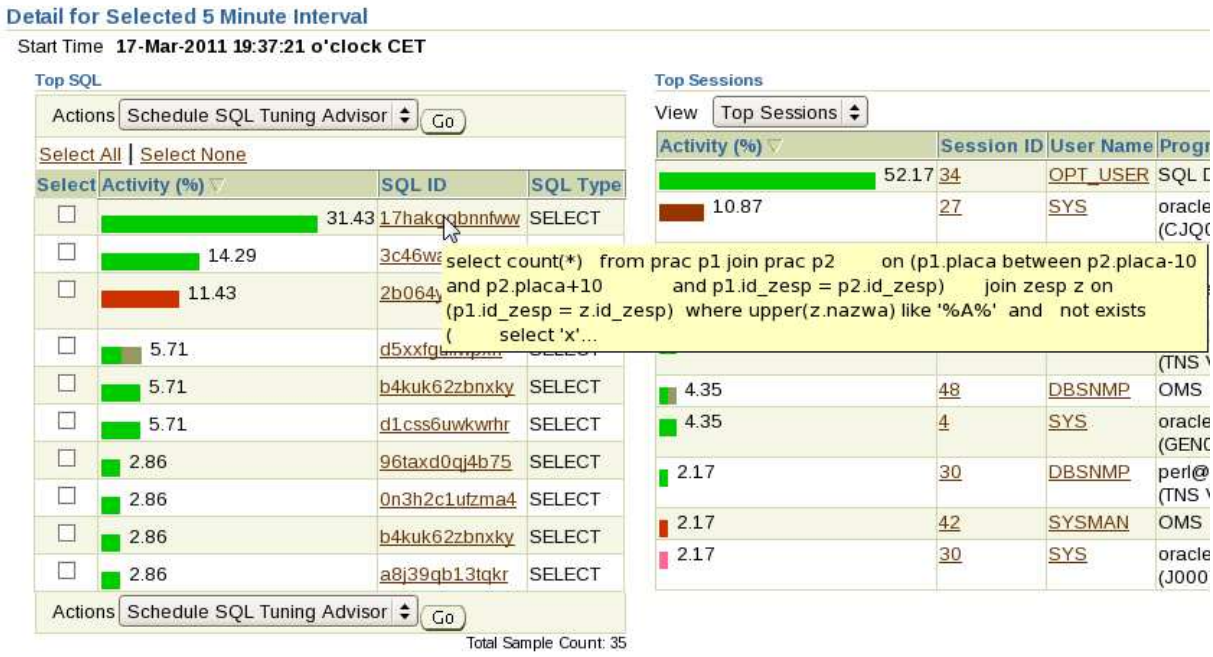
6. Przełącz się do programu SQL Developer w celu wygenerowania problematycznych poleceń SQL. Uruchom poniższe polecenie.

```
select count(*)
from prac p1 join prac p2
     on (p1.placa between p2.placa-10 and p2.placa+10
         and p1.id_zesp = p2.id_zesp)
     join zesp z on (p1.id_zesp = z.id_zesp)
where upper(z.nazwa) like '%A%'
and   not exists (
      select 'x'
      from   prac
      where  plec <> p1.plec
      and    p2.id_zesp = p1.id_zesp
      and    p2.placa = p1.placa);
```

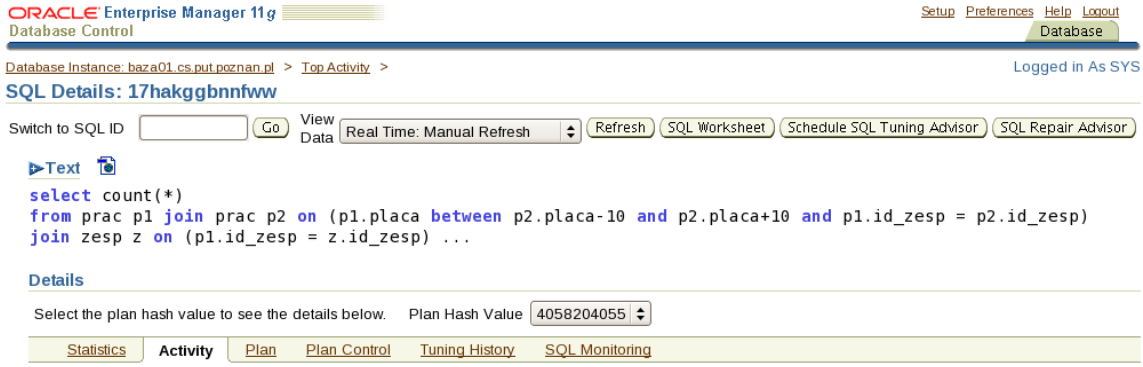
7. Powróć do konsoli EM I przesuń “okno” Twojego zainteresowania tak aby objąć czas bieżący.



8. Przejdź ponownie do obszaru dotyczącego ‘problematycznych’ poleceń SQL – *Top SQL*. Sprawdź czy pierwsze z wyświetlonych poleceń jest tym, które przed chwilą było uruchomione.



9. Kliknij w link tego polecenia przechodząc do jego szczegółów.



10. Sprawdź plan tego polecenia. Wykorzystaj widok tabelaryczny.

View ○ Graph ◉ Table

Expand All | Collapse All

Operation	Object	Order	Rows	Bytes	Cost	CPU (%)	Time	Query Block Name/Object Alias	Predicate	Filter
SELECT STATEMENT		12			26	100				
SORT AGGREGATE		11	1	72				SEL$9E43CB6E		
FILTER		10								IS NULL
HASH JOIN		7	1	72	24	4	0:0:1		"SYS_ALIAS_5"."ID_ZESP"="SYS_A...	("SYS_ALIAS_
NESTED LOOPS		5								
NESTED LOOPS		3	500	22.461K	8	0	0:0:1			
TABLE ACCESS FULL	ZESP	1	1	17	3	0	0:0:1	SEL$9E43CB6E / Z@SEL$2		UPPER("Z"."NA
INDEX RANGE SCAN	PRAC_ID_ZESP_IDX	2	32		4	0	0:0:1	SEL$9E43CB6E / P1@SEL$1	"SYS_ALIAS_5"."ID_ZESP"="Z"."I...	
TABLE ACCESS BY INDEX ROWID	PRAC	4	2,000	56.641K	5	0	0:0:1	SEL$9E43CB6E / P1@SEL$1		
TABLE ACCESS FULL	PRAC	6	10,000	253.906K	15	0	0:0:1	SEL$9E43CB6E / P2@SEL$1		

Strojenie poleceń SQL wymaga dużego doświadczenia. Aby ułatwić to zadanie wiele aplikacji posiada narzędzia wspomagające w tym procesie. Przykładem może być aplikacja *Enterprise Manager* i narzędzie *SQL Tuning Advisor*.

11. Zobaczymy co dla naszego zapytania zaproponuje to narzędzie. Aby go uruchomić kliknij w przycisk **Schedule SQL Tuning Advisor**.

Refresh | SQL Worksheet | Schedule SQL Tuning Advisor | SQL Repair Advisor

```
a-10 and p2.placa+10 and p1.id_zesp = p2.id_zesp)
```

12. W następnym oknie kliknij w przycisk **Submit** aktywizując sesję narzędzia *SQL Tuning Advisor*.

Cancel | Submit

13. Uruchomiona sesja może wykonywać swoje działania jakiś czas. Cierpliwie czekaj na zakończenie procesu strojenia.

Time Limit (seconds) **1800**

Status **EXECUTING**
Started **18-Mar-2011 16:18:21**
Elapsed Time (seconds) **0**

✔ Creating a new SQL Tuning task
➡ **0** out of **1** SQL Statements proceed.
0% accumulated potential benefit has been achieved.

14. W wyniku procesu strojenia pojawią się w naszym przypadku rekomendacje dotyczące: statystyk a także utworzenia tzw. *Profili SQL* – dodatkowych systemowych wskazówek dla optymalizatora pozwalających mu, w przypadku pojawienia się ponownie analogicznego zapytania, podjąć trafniejsze decyzje dotyczące planu jego wykonania.

Select Recommendation

Original Explain Plan (Annotated)

Implement

Select	Type	Findings	Recommendations	Rationale	Benefit (%)	Other Statistics	New Explain Plan	Compare Explain Plans
◉	Statistics	Tabela "OPT_USER"."ZESP" nie została przeanalizowana.	Proszę rozważyć gromadzenie statystyk optymalizatora dla tej tabeli.	Aby można było wybrać dobry plan wykonania, optymalizator wymaga aktualnych statystyk tabeli.				
○	SQL Profile	Dla tej instrukcji znaleziono potencjalnie lepszy plan wykonywania.	Proszę rozważyć zaakceptowanie zalecanego profilu SQL. No SQL profile currently exists for this recommendation.		1.0			