Rozproszone bazy danych – 3

Optymalizacja zapytań rozproszonych

Laboratorium przygotował:
Robert Wrembel

Plan laboratorium

- Zapytanie rozproszone i jego plan wykonania
- Narzędzia analizy planu wykonania zapytania
- Studium przypadku
  - filtrowanie danych z tabeli zdalnej
  - grupowanie i sortowanie danych z tabeli zdalnej
  - łączenie tabeli lokalnej i zdalnej
- Zastosowanie perspektywy w optymalizacji
- Zastosowanie wskazówek w optymalizacji

Celem laboratorium jest omówienie podstawowych problemów optymalizacji zapytań rozproszonych i technik ich rozwiązywania. Zostaną one zilustrowane implementacją w systemie Oracle9i/10g. W ramach laboratorium zostaną omówione następujące zagadnienia:

- charakterystyka zapytania rozproszonego i plan jego wykonania,

- narzędzia Oracle analizy planu wykonania,

- studium przypadku obejmujące analizę technik filtrowania danych z tabeli zdalnej, grupowania i sortowania danych z tabeli zdalnej, łączenie tabeli lokalnej i zdalnej,

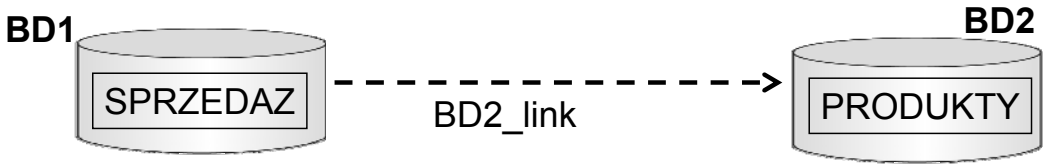
- zastosowanie tradycyjnych perspektyw (ang. view) w optymalizacji zapytań rozproszonych,

- zastosowanie wskazówek optymalizatora kosztowego w optymalizacji zapytań rozproszonych.

Zapytanie rozproszone

- Odwołuje się przynajmniej do dwóch różnych baz danych, z których przynajmniej jedna jest bazą zdalną

```
select s.ilosc, p.nazwa
from sprzedaz s, produkty@BD2_link
where s.produkt_id=p.produkt_id;
```



Zapytanie rozproszone to takie zapytanie, które odwołuje się przynajmniej do dwóch różnych baz danych, z których jedna jest bazą zdalną.

W przykładowej architekturze ze slajdu, zapytanie rozproszone zainicjowano z bazy lokalnej BD1. Zapytanie to łączy lokalną tabelę SPRZEDAZ (w bazie BD1) ze zdalną tabelą PRODUKTY (w bazie BD2). Odwołanie do tabeli w zdalnej bazie danych zostało zrealizowane za pomocą łącznika bazy danych (ang. database link) o nazwie BD2_link.

Definicje

- Baza lokalna/baza inicjująca – baza danych, w której wydano zapytanie rozproszone
- Baza zdalna – baza danych umieszczona na zdalnym serwerze, do której odwołuje się zapytanie z bazy lokalnej

W dalszej części laboratorium będą wykorzystywane dwa następujące pojęcia:

1. Baza lokalna (zwana również bazą inicjującą) jest to baza danych, w której wydano zapytanie rozproszone.

2. Baza zdalna jest to baza danych umieszczona na zdalnym serwerze, do której odwołuje się zapytanie z bazy lokalnej.

Wykonanie zapytania rozproszonego

- Krok 1: rozbicie na zapytania cząstkowe, każde do jednej bazy danych
- Krok 2: optymalizacja, wykonanie zapytania cząstkowego w obsługującej je bazie danych i przekazanie wyniku
- Krok 3: integrowanie wyników cząstkowych zgodnie z zapytaniem rozproszonym

Zapytanie rozproszone kierowane do kilku zdalnych baz danych jest realizowane w następujących krokach.

1. Lokalna baza danych, w której jest realizowane zapytanie rozproszone, rozbija je na zapytania cząstkowe. Każe z zapytań cząstkowych jest kierowanie niezależnie do odpowiedniej zdalnej bazy danych.

2. W zdalnej bazie danych zapytanie jest optymalizowane, wykonywane, a jego wynik jest przekazywany do bazy inicjującej zapytanie.

3. Po otrzymaniu wyników wszystkich zapytań cząstkowych, baza inicjująca realizuje operacje łączenia, agregowania i sortowania wyników cząstkowych.

Optymalizator zapytań w SZBD w lokalnej bazie danych nie ma informacji na temat dodatkowych struktur fizycznych wykorzystywanych w zdalnych bazach danych, ani też na temat statystyk dla struktur w zdalnych bazach danych.

Plan wykonania zapytania

- Sposób wykonania zapytania z podziałem na elementarne operacje: odczyt danych z dysku, filtrowanie, łączenie, agregowanie, sortowanie
- Budowany przez optymalizator zapytań

Każde zapytanie jest realizowane za pomocą tzw. planu wykonania (ang. execution plan). Plan ten opisuje sposób wykonania zapytania z podziałem na elementarne operacje odczytu danych z dysku (np. odczyt całej tabeli, odczyt rekordów z wykorzystaniem indeksu), filtrowanie danych, łączenie tabel, agregowanie wyników, sortowanie.

Plan wykonania zapytania jest budowany przez optymalizator zapytań przed fizycznym wykonaniem zapytania.

Przykładowy plan wykonania zapytania

```
OPERATION             OPTIONS    OBJECT_NAME OBJECT_NODE OTHER
--------------------- ---------- ----------- ----------- ------------------------
SELECT STATEMENT
  NESTED LOOPS
    REMOTE                                   DB2_link    SELECT "ILOSC","SKLEP_ID"
                                                         FROM "SPRZEDAZ" "SP"
    TABLE ACCESS      BY INDEX   SKLEPY
                      ROWID
      INDEX           UNIQUE     SKLEPY_PK
                      SCAN
```

zdalne zapytanie

algorytm łączenia

dostęp do tabeli lokalnej z wykorzystaniem indeksu unikalnego o nazwie SKLEPY_PK (założony na kluczu podst. tabeli SKLEPY)

Przykładowy plan wykonania zapytania przedstawiono na slajdzie. Jego analizę należy rozpocząć od pozycji bardziej zagnieżdżonych. W przykładowym planie dokonano połączenia tabeli zdalnej SPRZEDAZ z tabelą lokalną SKLEPY. Wykorzystano do tego celu algorytm NESTED LOOPS. Tabelą zewnętrzną w tym algorytmie jest zdalna tabela SPRZEDAZ (wiersz REMOTE). Odwołanie do niej jest realizowane za pomocą łącznika DB2_link, a zapytanie skierowane do tej bazy jest następujące:

select ilosc, sklep_id from sprzedaz sp

Dla każdego rekordu wynikowego tego zapytania, jest odczytywany odpowiedni rekord z lokalnej tabeli SKLEPY. Dostęp do właściwego sklepu jest realizowany za pomocą indeksu zdefiniowanego na atrybucie unikalnym.

Analiza planu wykonania zapytania

- Narzędzia
  - polecenie SQL: EXPLAIN PLAN
  - dyrektywa AUTOTRACE
  - narzędzie SQLTRACE i program tkprof

Rzeczywisty plan wykonania danego polecenia SQL może być udostępniony administratorowi systemu. Uzyskiwanie informacji na temat planu wykonania polecenia SQL nazywa się **wyjaśnianiem**. Wyjaśnienie planu polecenia można zrealizować na trzy następujące sposoby:

- za pomocą polecenia EXPLAIN PLAN (polecenie języka SQL),

- z wykorzystaniem dyrektywy AUTOTRACE programu *SQL*Plus*,

- z wykorzystaniem narzędzia SQLTRACE i programu *tkprof*.

EXPLAIN PLAN (1)

- Ogólna składnia polecenia

```
explain plan set statement_id='identyfikator'
for wyjaśniane_polecenie;
```

  - *statement_id*: określa unikalny identyfikator planu
  - *wyjaśniane_polecenie*: polecenie SQL, dla którego plan ma zostać wyjaśniony

Polecenie EXPLAIN PLAN wymusza na optymalizatorze wygenerowanie planu takiego, jaki byłby wykorzystany do wykonania wskazanego polecenia SQL. W rzeczywistości, samo polecenie nie jest wykonywane, a jedynie jego plan wykonania jest zapisywany w przeznaczonej do tego celu tabeli, najczęściej o nazwie *PLAN_TABLE*. Użytkownik wyjaśniający polecenie musi posiadać w swoim schemacie tabelę *PLAN_TABLE* lub posiadać do niej uprawnienia obiektowe *INSERT* i *SELECT*. Tabelę tę tworzy się za pomocą skryptu o nazwie *utlxplan.sql* dostarczanego wraz z *SZBD Oracle*.

Podstawowa składnia polecenia explain plan została przedstawiona poniżej:

explain plan set statement_id='identyfikator' for wyjaśniane_polecenie;

Klauzula *set statement_id* służy do określenia identyfikatora planu dla polecenia wskazanego klauzulą *for*. Domyślna wartość tego identyfikatora jest nieokreślona (*null*). Użytkownik powinien zagwarantować unikalną wartość identyfikatora planu dla każdego wyjaśnianego polecenia. W przeciwnym przypadku plan wykonania zostanie wyświetlony błędnie.

EXPLAIN PLAN (2)

- Przykład

```
select sk.nazwa, sp.ilosc
from sklepy sk,
sprzedaz@DB2_link sp
where sk.sklep_id=sp.sklep_id;
```


Przykład wykorzystania polecenia EXPLAIN PLAN przedstawiono poniżej.

explain plan set statement_id='1'

for

select sk.nazwa, sp.ilosc

from sklepy sk, sprzedaz@lab92 sp

where sk.sklep_id=sp.sklep_id;

Polecenie to umożliwia wyjaśnienie planu wykonania poniższego zapytania. Identyfikator jego planu określono jako 1 (klauzula *set statement_id='1'*).

select sk.nazwa, sp.ilosc

from sklepy sk, sprzedaz@lab92 sp

where sk.sklep_id=sp.sklep_id;

Odczyt planu

```
select lpad(' ',2*(level-1))||operation operation,
       options,
       object_name, optimizer,object_node,other
from plan_table
start with id=0 and statement_id='1'
connect by prior id=parent_id
and statement_id='1';
```


Plan wykonania można następnie wyświetlić, korzystając z polecenia SELECT skierowanego do tabeli *PLAN_TABLE*. Poniżej przedstawiono przykładowe polecenie wyświetlające plan wykonania zapytania o identyfikatorze 1 i jego wynik.

```
select lpad(' ',2*(level-1))||operation operation,options,
    object_name, optimizer,object_node,other
from plan_table
start with id=0 and statement_id='1'
connect by prior id=parent_id
and statement_id='1';
```

Dyrektywa AUTOTRACE

- Plan jest wyświetlany automatycznie po wykonaniu polecenia
- Włączenie wyświetlania planów z wykonaniem polecenia SQL

```
set autotrace on explain
```

- Włączenie wyświetlania planów bez wykonania polecenia SQL

```
set autotrace traceonly explain
```


Plan można również wyświetlić korzystając z dyrektywy AUTOTRACE programu *SQL*Plus*. Użytkownik wykorzystujący tę dyrektywę musi posiadać dostęp do tabeli *PLAN_TABLE* i musi posiadać rolę *PLUSTRACE*. Do definiowania tej roli służy skrypt *plustrce.sql*. Ponieważ skrypt ten nie jest uruchamiany podczas instalacji bazy danych, należy go wykonać jawnie w schemacie użytkownika *SYS*.

Składnia dyrektywy autotrace została przedstawiona poniżej.

set autotrace {on | traceonly} explain

Użycie słowa kluczowego *on* powoduje, że w pierwszej kolejności zostaje wyświetlony wynik wprowadzanego polecenia SQL, a następnie plan jego wykonania. Natomiast słowo kluczowe *traceonly* powoduje pomijanie wyświetlania wyników wprowadzonego polecenia. Po wykonaniu powyższego polecenia *set autotrace*, dla każdego polecenia SQL wprowadzonego z poziomu *SQL*Plus*, będzie wyświetlany jego plan wykonania. Wyłączenie wyjaśniania planów realizuje się poleceniem:

set autotrace off

Przykład

```
SQL> set autotrace traceonly explain
SQL> select nazwisko from pracownicy where id_prac=100;

Plan wykonania
----------------------------------------------------------
 0    SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=1 Bytes=22)
 1  0   TABLE ACCESS(BY ROWID)OF 'PRACOWNICY'(Cost=1 Card=1 Bytes=22)
 2  1   INDEX (UNIQUE SCAN) OF 'PK_PRAC' (UNIQUE)
```

rodzaj optymalizatora i całkowity koszt wykonania polecenia

dostęp do tabeli lokalnej PRACOWNICY z wykorzystaniem indeksu unikalnego

Przykład wykorzystania AUTOTRACE i jego wynik przedstawiono na slajdzie. Dla każdej operacji wchodzącej w skład planu wykonania są wyświetlane informacje z tabeli *PLAN_TABLE*. W pierwszej kolumnie jest wyświetlany identyfikator operacji, w drugiej — identyfikator operacji podrzędnej wykonywanej w ramach operacji wskazywanej wartością pierwszej kolumny. Następnie, jest podawana informacja o rodzaju operacji, jej opcje oraz nazwa obiektu, którego operacja dotyczy. Dodatkowo, w nawiasach umieszczony jest koszt (*Cost*), liczba (*Cardinality*) i rozmiar (*Bytes*) rekordów przetworzonych w ramach bieżącej operacji. Ponadto, dla kroku o identyfikatorze 0 jest wyświetlany rodzaj wykorzystanego optymalizatora (*Optimizer*) i całkowity koszt wykonania polecenia (*Cost*).

Powyższy plan został wygenerowany przez optymalizator kosztowy (*Optimizer=CHOOSE*), a łączny koszt wykonania zapytania wyniósł 1 (*Cost=1*). Podawany koszt jest mierzony w pewnych umownych jednostkach i może być wykorzystany jedynie do porównywania alternatywnych planów. Rekordy tabeli pracownicy (*TABLE ACCESS(BY ROWID)OF 'PRACOWNICY'*) odczytano z wykorzystaniem indeksu unikalnego o nazwie *pk_prac* (*INDEX (UNIQUE SCAN) OF 'PK_PRAC' (UNIQUE)*).

SQLTRACE i tkprof (1)

- SQLTRACE: zbieranie dokładnych statystyk i planu wykonania każdego polecenia
- Zbieranie statystyk dla wskazanej sesji albo dla wszystkich sesji w bazie danych
- Wynik działania jest zapisywany w tzw. pliku śladu (ang. trace file)
- Zawartość pliku śladu należy przetworzyć programem tkprof

Narzędzie SQLTRACE umożliwia zbieranie statystyk dotyczących przetwarzanego polecenia SQL (m.in., liczbę operacji parse, execute i fetch, zużyty czas procesora, liczbę logicznych i fizycznych odczytów danych) i analizę planów wykonania tych poleceń.

Narzędzie SQLTRACE można uaktywnić na dwóch poziomach: bazy danych i sesji. Uaktywnienie na poziomie bazy danych powoduje, że statystyki polecenia będą zbierane dla wszystkich sesji użytkowników bazy danych. Natomiast uaktywnienie na poziomie sesji powoduje, że statystyki będą zbierane jedynie dla sesji użytkownika.

Wynik działania *SQLTRACE* jest zapisywany do tzw. **plików śladu** (ang. *trace files*). Każda monitorowana sesja posiada własny plik. Lokalizację tych plików określa parametr konfiguracyjny user_dump_dest (pliku *initSID.ora*).

Bezpośrednio po zapisaniu, zawartość pliku śladu jest mało czytelna. Dlatego należy go przetworzyć za pomocą programu *tkprof*.

SQLTRACE i tkprof (2)

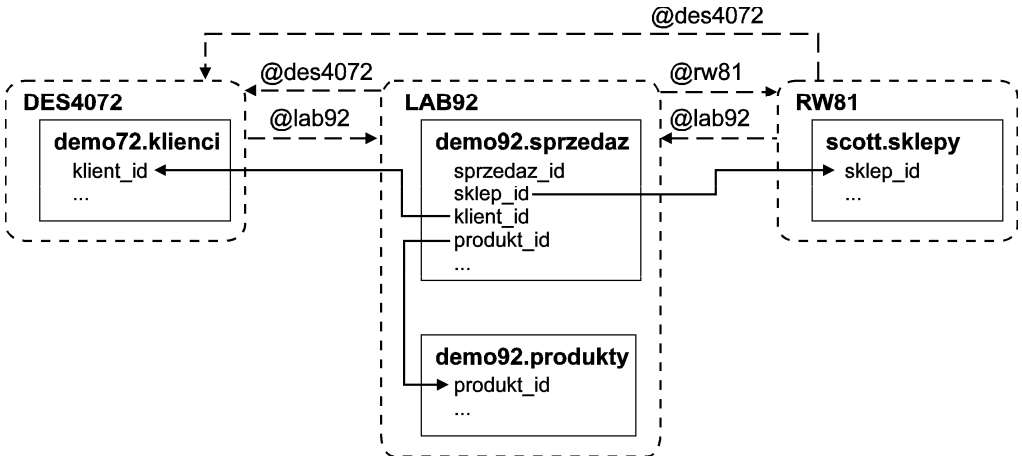
- Uaktywnienie SQLTRACE na poziomie bazy danych
  - parametr konfiguracyjny pliku initSID.ora
    - SQL_TRACE=TRUE
- Uaktywnienie SQLTRACE na poziomie sesji użytkownika
  - polecenie: ALTER SESSION
  - procedura systemowa: DBMS_SESSION.SQL_TRACE lub DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION

Do uaktywnienia SQLTRACE na poziomie bazy danych służy parametr konfiguracyjny sql_trace zapisany w pliku konfiguracyjnym *initSID.ora*. Wartość TRUE tego parametru uaktywnia narzędzie SQLTRACE. Domyślną wartością parametru jest false.

Uaktywnienie SQLTRACE na poziomie sesji użytkownika realizuje się z wykorzystaniem polecenia alter session lub wywołując procedury systemowe DBMS_SESSION.SET_SQL_TRACE bądź DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION.

Zapytania rozproszone: studium przypadku

- Środowisko testowe



Zapytania i plany ich wykonania, omawiane w dalszej części ćwiczeń zostały wykonane w środowisku trzech baz danych: *DES4072*, *LAB92*, *RW81*. W bazie *DES4072*, w schemacie użytkownika *demo72* utworzono tabelę *klienci* zawierającą 1000 rekordów. W bazie *LAB92*, w schemacie użytkownika *lab92* utworzono tabele *produkty* i *sprzedaz*, zawierające odpowiednio 40 i 120000 rekordów. W bazie *RW81*, w schemacie użytkownika *scott* utworzono tabelę *sklepy* z 30 rekordami.

Filtrowanie danych z tabeli zdalnej (1)

- Zapytanie

```
select nazwisko
from klienci@des4072
where
lower(miasto)='Sopot';
```

- Plan wykonania

```
OPERATION             OPTIONS    OBJECT_NAME OBJECT_NODE OTHER
--------------------- ---------- ----------- ----------- -----
SELECT STATEMENT      REMOTE
  TABLE ACCESS        FULL       KLIENCI     DES4072
```

```
SELECT "A1"."NAZWISKO"
FROM
 "KLIENCI" "A1" WHERE LOWER("A1"."MIASTO")='Sopot'
```


Poniżej przedstawiono zapytanie z bazy *rw81* do *des4072* zawierające warunek filtrowania danych tabeli zdalnej *klienci*. Dodatkowo, w klauzuli *where* wykorzystano funkcję *lower*.

select nazwisko

from klienci@des4072

where lower(miasto)='Sopot';

Plan wykonania tego zapytania przez optymalizator regułowy i kosztowy w bazie *rw81* jest taki sam. Przedstawiony na slajdzie plan otrzymano za pomocą polecenia *explain plan*. Dostęp do tabeli *klienci* jest realizowany za pomocą liniowego odczytu całej tabeli (rekord: *TABLE ACCESS FULL*).

Do zdalnej bazy danych jest wysyłane poniższe zapytanie. Jak widać, filtrowanie danych jest wykonywane w bazie zdalnej, do której jest przesyłany warunek *lower(miasto)='sopot'*. Wynik zapytania jest następnie przesyłany z bazy zdalnej *des4072* do lokalnej *rw81*.

SELECT "A1"."NAZWISKO"

FROM

"KLIENCI" "A1" WHERE LOWER("A1"."MIASTO")='sopot‘

Filtrowanie danych z tabeli zdalnej (2)

- Wniosek:
  - filtrowanie danych z tabeli zdalnej jest wykonywane w bazie zdalnej
  - do bazy inicjującej zapytanie rozproszone są przesyłane tylko te rekordy, które spełniają warunki selekcji

Z analizy planu wykonania zapytania wynika, że filtrowanie danych z tabeli zdalnej jest wykonywane w bazie zdalnej, a do bazy inicjującej zapytanie rozproszone są przesyłane tylko te rekordy, które spełniają warunki selekcji.

Grupowanie i sortowanie danych z tabeli zdalnej (1)

- Zapytanie

```
select count(*), miasto
from klienci@des4072
where kredyt between 500
and 1100
group by miasto
order by miasto;
```

- Plan wykonania

```
OPERATION             OPTIONS    OBJECT_NAME OBJECT_NODE OTHER
--------------------- ---------- ----------- ----------- -----
SELECT STATEMENT      REMOTE
  SORT                ORDER BY
    SORT              GROUP BY
      TABLE ACCESS    FULL       KLIENCI     DES4072
```

zapytanie zdalne por. slajd 16

Na slajdzie przedstawiono zapytanie z bazy *rw81* do *des4072* zawierające filtrowanie, grupowanie i sortowanie danych.

select count(*), miasto

from klienci@des4072

where kredyt between 500 and 1100

group by miasto

order by miasto;

Plan wykonania tego zapytania przez optymalizator regułowy i kosztowy w bazie *rw81* jest taki sam. Przedstawiono go na slajdzie.

Grupowanie i sortowanie danych z tabeli zdalnej (2)

- Zapytanie zdalne

```
SELECT COUNT(*),"A1"."MIASTO"
FROM "KLIENCI" "A1"
WHERE "A1"."KREDYT">=500
AND "A1"."KREDYT"<=1100
GROUP BY "A1"."MIASTO"
ORDER BY "A1"."MIASTO"
```


Do zdalnej bazy danych jest wysyłane następujące zapytanie:

```
SELECT COUNT(*),"A1"."MIASTO"
FROM
 "KLIENCI" "A1" WHERE "A1"."KREDYT">=500 AND
"A1"."KREDYT"<=1100 GROUP BY
  "A1"."MIASTO" ORDER BY "A1"."MIASTO"
```

Z planu tego wynika, że grupowanie wyników i sortowanie odbywa się w zdalnej bazie danych (rekord *SELECT STATEMENT REMOTE*), a do bazy *rw81* jest przesyłany końcowy wynik zapytania.

Grupowanie i sortowanie danych z tabeli zdalnej (3)

- Wniosek:
  - operacje grupowania (klauzula *group by*), wyliczania agregatów (m.in. funkcje *sum*, *avg*, *min*, *max*, *count*, *sttdev*, *variance*) sortowania (*order by*) wykonywane w bazie zdalnej
  - wynik tych operacji jest przesyłany do bazy inicjującej zapytanie rozproszone

Jeżeli zapytanie do zdalnej bazy danych zawiera operacje grupowania (klauzula *group by*), wyliczania agregatów (m.in. funkcje *sum*, *avg*, *min*, *max*, *count*, *sttdev*, *variance*) sortowania (*order by*), wówczas operacje te są wykonywane w bazie zdalnej, a ich wynik jest przesyłany do bazy inicjującej zapytanie rozproszone

Łączenie tabeli lokalnej i zdalnej (1)

- Zapytanie

```
select sk.nazwa, sp.ilosc
from sklepy sk, sprzedaz@lab92 sp
where sk.sklep_id=sp.sklep_id
and sk.miasto='Poznań';
```


Na slajdzie przedstawiono zapytanie wykorzystujące połączenie lokalnej tabeli *sklepy* ze zdalną tabelą *sprzedaz*, znajdującą się w bazie danych *lab92*.

select sk.nazwa, sp.ilosc
from sklepy sk, sprzedaz@lab92 sp
where sk.sklep_id=sp.sklep_id
and sk.miasto='Poznań';

Łączenie tabeli lokalnej i zdalnej (2)

- Plan wykonania (optymalizator regułowy)

```
OPERATION             OPTIONS    OBJECT_NAME OBJECT_NODE OTHER
--------------------- ---------- ----------- ----------- -------------------------
SELECT STATEMENT
  NESTED LOOPS
    REMOTE                                   LAB92       SELECT "ILOSC","SKLEP_ID"
                                                         FROM "SPRZEDAZ" "SP"
    TABLE ACCESS      BY INDEX   SKLEPY
                      ROWID
      INDEX           UNIQUE     SKLEPY_PK
                      SCAN
```


Plan wykonania tego zapytania przez optymalizator regułowy w bazie *rw81* przedstawiono na slajdzie.

Obie tabele są łączone za pomocą algorytmu *nested-loops* (*OPERATION=NESTED LOOPS*). Tabelą zewnętrzną w tym przypadku jest *sprzedaz@lab92*, która jest w całości odczytywana ze zdalnej bazy danych (*OPERATION=REMOTE* dla trzeciej linii planu zapytania) poniższym zapytaniem. Natomiast dostęp do tabeli wewnętrznej jest realizowany z wykorzystaniem indeksu (rekord *TABLE ACCESS BY INDEX ROWID* i *INDEX UNIQUE SCAN SKLEPY_PK*) *sklepy_pk*, na atrybucie kluczowym *sklep_id*.

Łączenie tabeli lokalnej i zdalnej (3)

- Plan wykonania (optymalizator kosztowy)

```
OPERATION             OPTIONS    OBJECT_NAME OBJECT_NODE OTHER
--------------------- ---------- ----------- ----------- -------------------------
SELECT STATEMENT
  NESTED LOOPS
    TABLE ACCESS      FULL       SKLEPY
    REMOTE                                   LAB92.WORLD SELECT "ILOSC","SKLEP_ID"
                                                         FROM "SPRZEDAZ" "SP" WHE
                                                         RE :1="SKLEP_ID"
```


W przypadku wykorzystania optymalizatora kosztowego w obu bazach danych, plan wykonania powyższego zapytania będzie inny, jak pokazano na slajdzie. W tym przypadku, tabelą zewnętrzną jest *sklepy*, która jest odczytywana w całości (*TABLE ACCESS FULL SKLEPY*). Tabelą wewnętrzną jest *sprzedaz@lab92*.

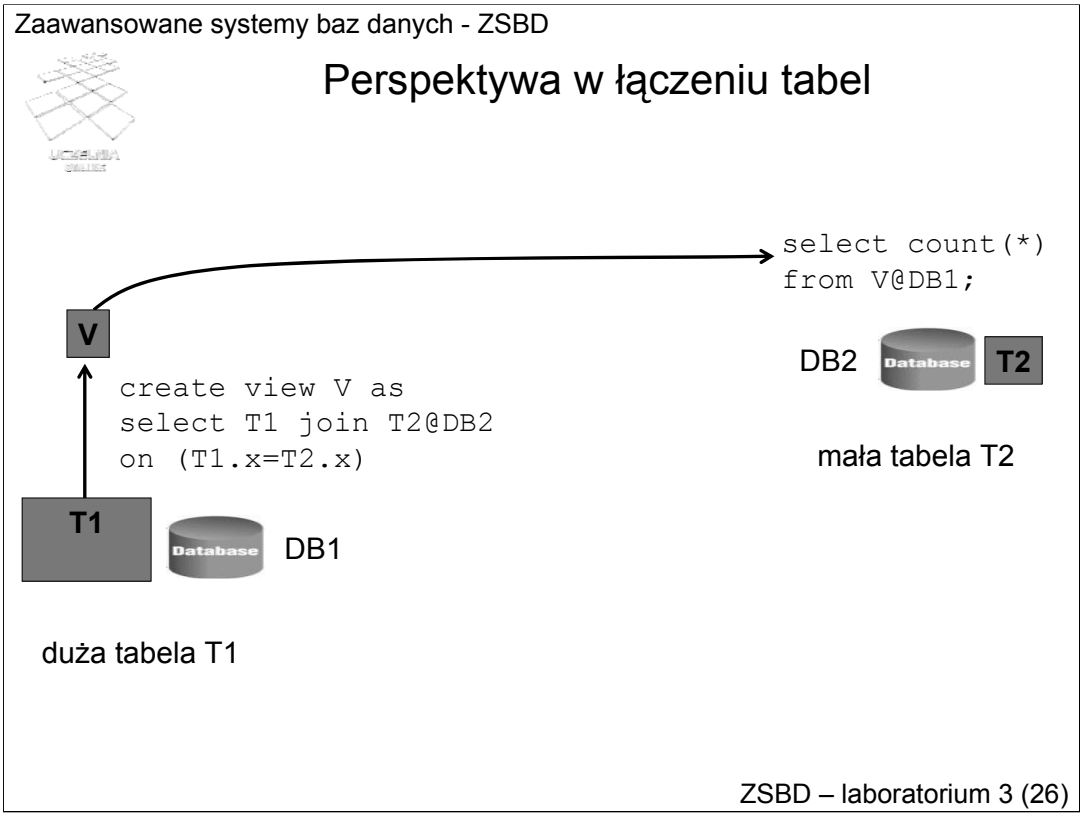
Do zdalnej bazy danych jest wysyłane poniższe zapytanie, rozszerzone o warunek *where :1=sklep_id*, wynikający z warunku połączenia tabel. W konsekwencji, z bazy *lab92* będą przesyłane tylko te rekordy, które spełniają warunek połączenia z rekordami tabeli *sklepy*.

```
SELECT "ILOSC","SKLEP_ID"
  FROM "SPRZEDAZ" "SP"
  WHERE :1="SKLEP_ID"
```

Łączenie tabeli lokalnej i zdalnej (4)

- Wniosek:
  - optymalizator kosztowy uzupełnia zapytanie wysyłane do zdalnej bazy danych o warunek umożliwiający wybór tylko tych rekordów, które łączą się z rekordami tabeli lokalnej

W przypadku łączenia tabeli zdalnej z lokalną, optymalizator kosztowy uzupełnia zapytanie wysyłane do zdalnej bazy danych o warunek umożliwiający wybór tylko tych rekordów, które łączą się z rekordami tabeli lokalnej.



W przypadku realizowania połączenia tabeli zdalnej z lokalną, krytycznym czynnikiem wpływającym na efektywność wykonania takiego połączenia jest liczba dostępów do zdalnej tabeli i rozmiar przesyłanych danych. W poprzednim zapytaniu, optymalizator regułowy zbudował plan, w którym cała zawartość tabeli *sprzedaz* była przesyłana siecią do bazy lokalnej *rw81* i tu dopiero realizowane było połączenie z tabelą *sklepy*. Zwróćmy uwagę, że siecią przesyłana była zawartość znacznie większej tabel (120000 rekordów) niż rozmiar tabeli lokalnej (30 rekordów). Z punktu widzenia liczby przesyłanych danych, bardziej efektywnym może być przesłanie mniejszej tabeli do zdalnej bazy danych, wykonanie tam połączenia i odesłanie wyników. Na opłacalność takiej operacji ma wpływ liczba rekordów z dużej tabeli, która spełnia warunki połączenia.

Wymuszenie wykonania połączenia po stronie mniejszej tabeli można wymusić definiując perspektywę po stronie większej tabeli (T1 w bazie DB1 na slajdzie). Perspektywa taka łączy wtedy małą tabelę zdalną (T2 w bazie DB2) z dużą tabelą lokalną (T1 w DB1). Wynik połączenia jest przesyłany do bazy kierującej zapytanie do perspektywy, czyli w naszym przypadku do bazy DB2.

Wskazówki optymalizatora kosztowego (1)

ORDERED

zachowanie kolejności tabel, takiej jak w klauzuli FROM

tabelą zewnętrzną jest pierwsza z lewej

```
select /*+ ORDERED */ sk.nazwa, sp.ilosc
from sklepy sk, sprzedaz@lab92 sp
where sk.sklep_id=sp.sklep_id;
```

FULL

wskazanie tabeli odczytywanej za pomocą full table scan

```
select /*+ FULL(sp) */ sk.nazwa, sp.ilosc
from sprzedaz@lab92 sp, sklepy sk
where sk.sklep_id=sp.sklep_id;
```


Wskazówka ORDERED umożliwia zachowanie kolejności tabel w operacji połączenia takiej, jaka została ustalona w klauzuli from. Optymalizator kosztowy wybierze jako zewnętrzną tę tabelę, którą wymieniono jako pierwszą po słowie kluczowym from.

Przykładowo, w zapytaniu na slajdzie, wskazówka ORDERED ustala tabelę *sklepy* jako zewnętrzną.

Wzkazówka FULL umożliwia wskazanie tej tabeli, do której dostęp ma zostać zrealizowany za pomocą liniowego odczytu całej tabeli.

Jako przykład rozważmy zapytanie ze slajdu łączące lokalną tabelę *sklepy* ze zdalną tabelą *sprzedaz*. Wskazówka FULL(sp) ustala w tym przypadku tabelę *sprzedaz* jako przeglądaną liniowo w całości. Nazwa tabeli jest przekazywana do wskazówki za pomocą aliasu *sp*.

Wskazówki optymalizatora kosztowego (2)

DRIVING_SITE

wskazanie bazy danych, w której ma zostać wykonane zapytanie

```
select /*+ DRIVING_SITE(sp) */ kl.nazwisko,
sp.data, sp.ilosc, sp.produkt_id
from klienci kl, sprzedaz@lab92 sp
where kl.klient_id=sp.klient_id
and kl.miasto='Poznań';
```


Wskazówka DRIVING_SITE umożliwia wskazanie bazy danych, w której ma zostać wykonane zapytanie. Jest ona alternatywą dla tworzenia perspektywy (por. slajd 22).

W przykładzie ze slajdu DRIVING_SITE(sp) wskazuje bazę danych, której zostanie wykonane zapytanie. Baza ta jest przekazywana jako argument wskazówki, tj. *sp*, który to argument jest aliasem do tabeli. Zapytanie wykona się w tej bazie danych, w której znajduje się tabela wskazana aliasem.

Wskazówki optymalizatora kosztowego (3)

NO_MERGE

zapytanie w perspektywie wbudowanej nie jest scalane z zapytaniem głównym

```
select /*+ NO_MERGE(v) */ kl.nazwisko, kl.miasto, v.ilosc,
v.nazwa
from klienci kl,
     (select sp.klient_id, sp.ilosc, pr.nazwa, pr.produkt_id
      from sprzedaz@lab92 sp, produkty@lab92 pr
      where sp.produkt_id=pr.produkt_id) v
where kl.klient_id=v.klient_id
and v.produkt_id=30;
```


Wskazówka NO_MERGE ma wpływ na podzapytanie umieszczone w klauzuli FROM, a łączące kilka tabel w tej samej zdalnej bazie danych. Wskazówka ta gwarantuje, że połączenie tych tabel zostanie zrealizowane w tej bazie danych w której te tabele się znajdują. Wynik połączenia zostanie przesłany do bazy danych inicjującej zapytanie. Bez tej wskazówki, zawartość wszystkich tabel z podzapytania zostałaby przesłana niezależnie do bazy danych inicjującej zapytanie i tu dopiero połączona.

Ćwiczenia: analiza planów wykonania zapytań

- Utworzyć w swoim schemacie tabelę PLAN_TABLE uruchamiając skrypt o nazwie `ultxplan10g.sql`
- Włączyć wyświetlanie planów poleceniem
  - set autotrace traceonly explain
- Zbadać plany wykonania poleceń znajdujących się w pliku o nazwie `zapytania_rozproszone_cwicz.txt`

Celem ćwiczenia jest praktyczne zapoznanie się z technikami optymalizacji zapytań rozproszonych. Ćwiczenie będzie polegało na wykonaniu predefiniowanego zbioru zapytań i analizę planu wykonania każdego z tych zapytań.