Interfejs ODBC Prolog/PostgreSQL

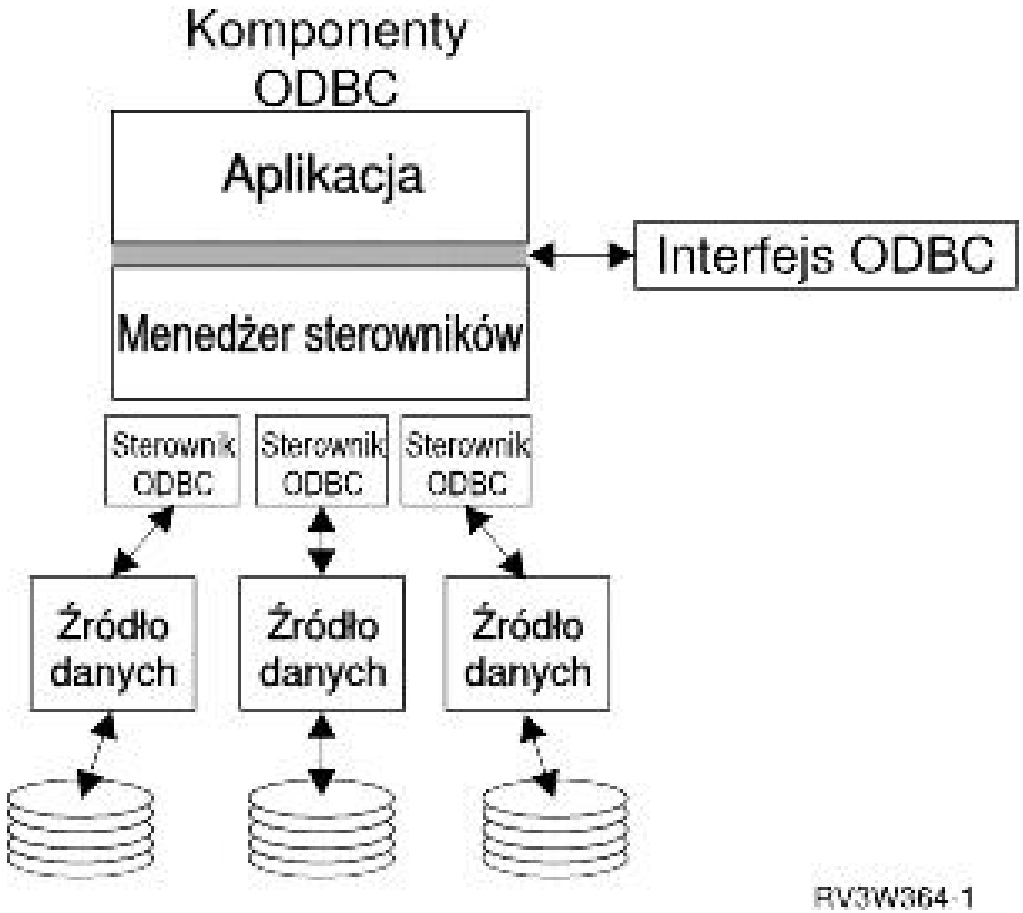
Krzysztof Turek

PWSZ Tarnów, IV rok IS

3 lutego 2009

Prolog umożliwia łączenie się z zewnętrznymi bazami danych poprzez wykorzystanie interfejsu ODBC. Interfejs ODBC jest szeroko rozpowszechnionym interfejsem programowania aplikacji (API) dla baz danych. Jest oparty na specyfikacji interfejsu CLI (Call - Level Interface) dla baz danych i używa SQL jako języka dostępu do danych.

ODBC (Open DataBase Ceonnectivity otwarte łącze baz danych) to interfejs pozwalający programom łączyć sie z systemami zarządzającymi bazami danych. Jest to API (Application Programming Interface) niezależnie od języka programowania, systemu operacyjnego i bazy danych. Standard ten został opracowany przez SQL Access Group we wrześniu 1992 roku jako alternatywa dla DAO i jest używany do dziś. Przedstawiciele kilku firm produkujących zarówno sprzęt jak i oprogramowanie przez kilka lat pracowali w SQL Access Group (SAG) nad zdefiniowaniem uniwersalnego sposobu dostępu do danych w celu uproszczenia oprogramowania typu klient-serwer. Firma Microsoft wykorzystała wyniki pracy grupy SAG i opracowała tak zwany call-level interface, później nazwany Open DataBase Connectivity (ODBC).


Komponenty
ODBC
Aplikacja
Interfejs ODBC
Menedżer sterowników
Sterownik
ODBC
Sterownik
ODBC
Sterownik
ODBC
Źródło
danych
Źródło
danych
Źródło
danych
RV3W364-1

1. Aplikacja - np. SWI-Prolog posługujący się zapytaniami SQL w celu uzyskania niezbędnych danych,
2. Menadżer sterowników zadaniem jego jest udostępnienie aplikacji odpowiedniego sterownika bazy danych,
3. Sterownik ODBC element wykonujący funkcje interfejsu ODBC wywołane przez zarządce sterowników. Przekazuje on również do źródła danych żądania SQL, a do aplikacji uzyskane wyniki. Jeśli jest to wymagane sterownik może modyfikować wykonywane zapytanie SQL w celu dostosowania do specyfiki docelowego źródła danych,
4. Źródło danych najczęściej system zarządzania bazami, np. PostgreSQL.

W celu możliwości wykorzystania interfejsu ODBC, należy posiadać zainstalowane poniżej podane pakiety.
W ramach przykładu korzystamy z bazy danych PostgreSQL, oficjalnego sterownika ODBC PostgreSQL oraz kompilatora języka Prolog - SWI-Prolog.

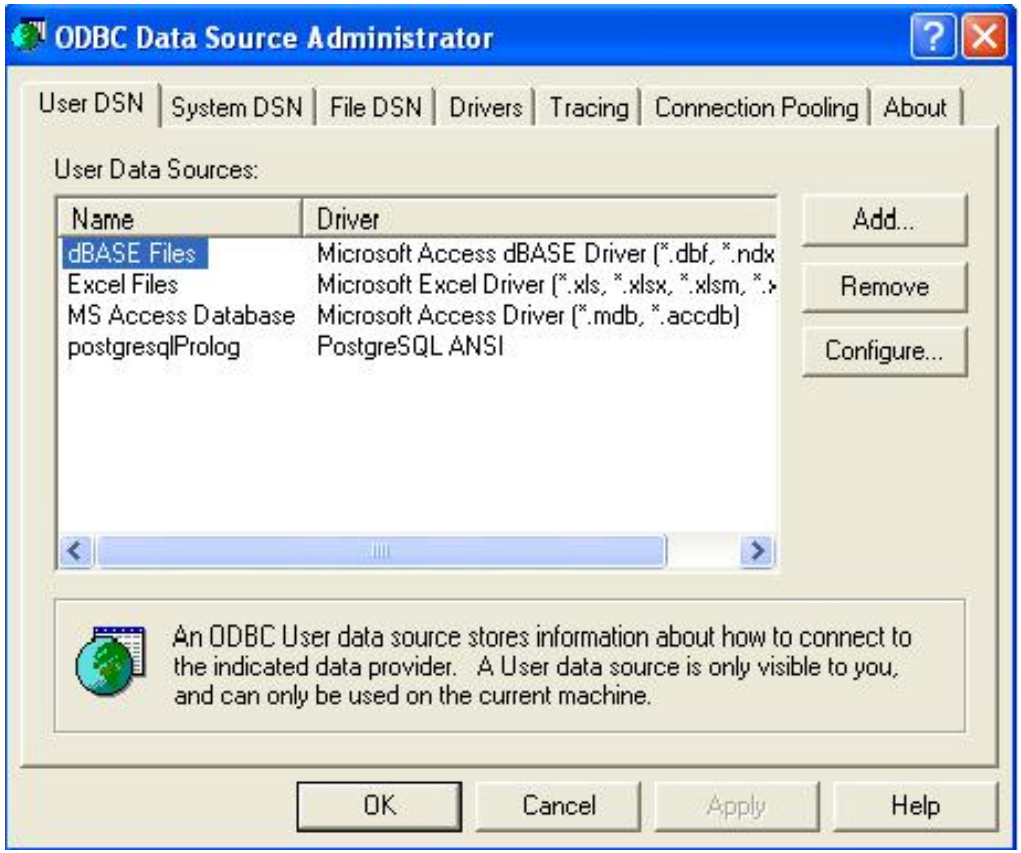
Lista wymaganych aplikacji na platforme Windows:

1. Program SWI-Prolog
   http://www.swi-prolog.org/dl-stable.html
   Instalacja przeprowadzana w sposób standardowy dla platformy Windows(ustawienia domyślne),
2. Baza danych PostgreSQL
   http://www.postgresql.org
   Instalacja przeprowadzana w sposób standardowy dla platformy Windows(ustawienia domyślne),
3. Sterownik bazy danych PostgreSQL
   http://www.postgresql.org/ftp/odbc/versions/msi/
   Instalacja przeprowadzana w sposób standardowy dla platformy Windows(ustawienia domyślne).

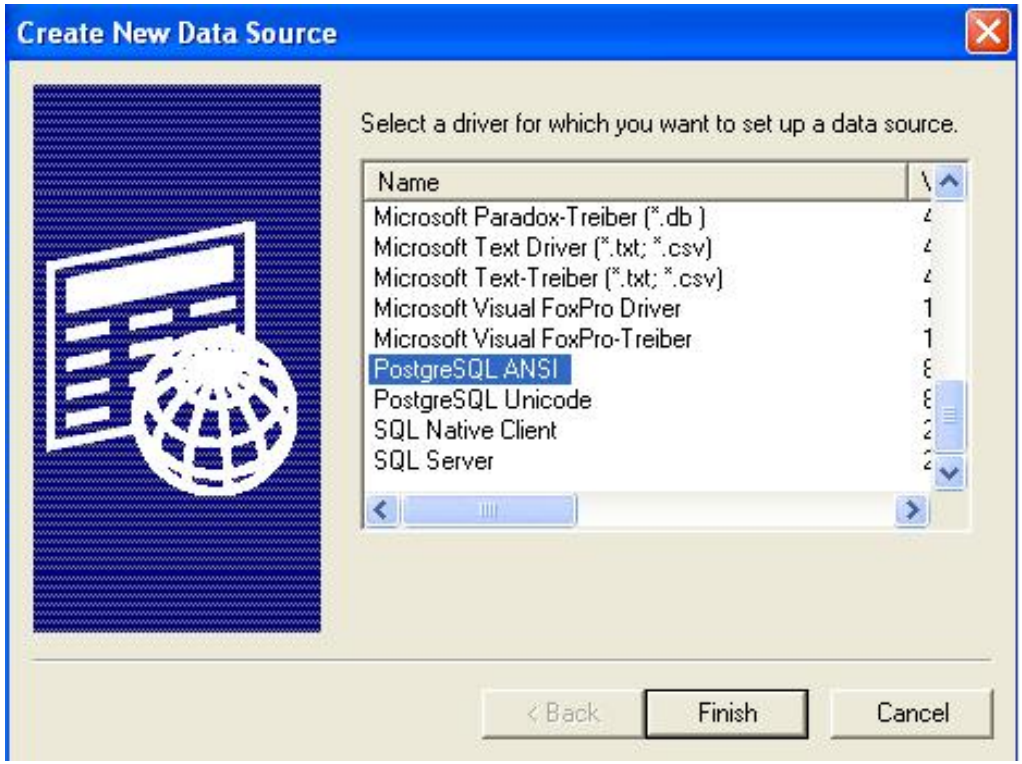
W celu skonfigurowania źródła danych należy uruchomić aplikację z podanej lokalizacji:

Start->Settings->Control Panel->Administrative Tools->Data Sources(ODBC)

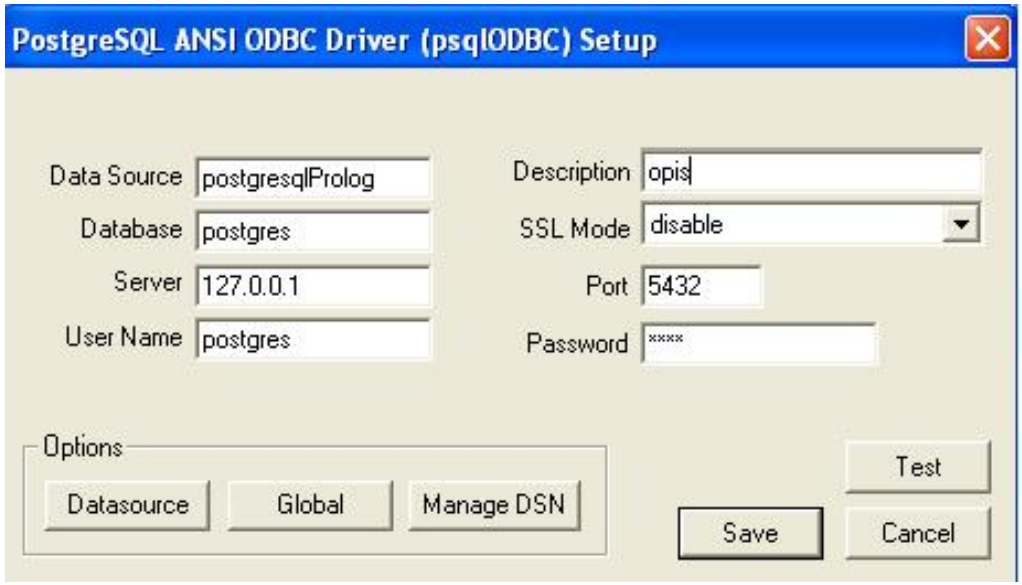
Na ekranie pojawi się okno jak poniżej na rysunku:



Po wybraniu opcji „ADD” pojawia się okno, w którym zaznaczamy interesujący nas sterownik ODBC(PostgreSQL), a następnie wybieramy opcję „Finish”.



Po wybraniu opcji „Finish” pojawia się ostatnie okno, w którym definiujemy parametry połączenia.



W polach:

- Data Source - nazwa połączenia z bazą danych,
- Database - nazwa bazy danych,
- Server adres IP serwera bazy danych,
- UserName- nazwa użytkownika bazy danych,
- Description opis,
- SSL Mode możliwość włączenia połączenia szyfrowanego,
- Port numer portu na którym baza danych nasłuchuje,
- Password- hasło użytkownika bazy danych.

Na koniec konfiguracji wybieramy opcję „TEST” w celu sprawdzenia poprawności połączenia a nastepnie opcję „SAVE”.

Lista wymaganych aplikacji na platforme Linux Kubuntu:

1. Program SWI-Prolog
   Instalacja przeprowadzana w sposób:
   apt-get install swi-prolog,
2. Baza danych PostgreSQL
   Instalacja przeprowadzana w sposób:
   apt-get install postgresql,
3. Sterownik bazy danych PostgreSQL
   Instalacja przeprowadzana w sposób:
   apt-get install odbc-postgresql (jest to sterownik PostgreSQL)
   apt-get install unixodbc (jest to menadżer sterowników).

Aby zainstalować aplikację należy się zalogować jako root- sudo su.

W celu skonfigurowania źródła danych należy zmodyfikować pliki **odbc.ini** oraz **odbcinst.ini**. Standardowo instalowane są do lokalizacji /**etc**.

Przykładowa konfiguracja plików:
odbcinst.ini - plik ten zawiera listę zarejestrowanych sterowników.

```
 [PostgreSQL ANSI]
Description = PostgreSQL ODBC driver (ANSI version)
Driver = /usr/lib/odbc/psqlodbca.so
Setup = /usr/lib/odbc/libodbcpsqlS.so
Debug = 0
CommLog = 1
```

odbc.ini - plik ten zawiera listę zdefiniowanych źródeł ODBC.

```
 [postgresqlProlog]
Description = PostgreSQL
Driver = PostgreSQL ANSI
Database = postgres
Servername = localhost
UserName = postgres
Password = pass
Port = 5432
```

Testowanie zdefiniowanego połączenia można wykonać za pomocą komendy:

```
isql postgresqlProlog
```

Argument „postgresqlProlog” oznacza zdefiniowane źródło danych z pliku **odbc.ini**.

Więcej informacji na stronie:

`http://www.easysoft.com/developer/interfaces/odbc/linux.html`

Interfejs ODBC w prologu dostarcza predykatów do zarządzania połączeniem, wykonywania zapytań SQL, odwzorowania typów danych. Możliwości ich poniżej bedą omówione.

Reprezentacja danych SQL w Prologu

Typy danych:

- Atom -używany domyślnie dla typów SQL char, varchar. longvarchar, binary, varbinary, longvarbinary, decimal, numeric. Może być używany dla wszystkich typów,
- string -rozszerzony typ stringu SWI-Prolog. Używa się w specjalnych przypadkach gdzie trzeba uniknąć atomów-śmieci,
- codes -lista kodów znaków,
- integer -używany domyślnie dla typów SQL bit, tinyint, smallint, integer. SWI-Prolog używa 32-bitowych integerów ze znakiem podczas gdy SQL zezwala na użycie liczb bez znaku. Może być używany dla liczb całkowitych, dziesiętnych jak również dla typów date i timestamp które są reprezentowane jako znaczniki czasu POSIX (sekundy od 1 stycznia 1970),

Typy danych:

- double -używany domyślnie dla typów SQL real, float i double. Może być użyty dla typów całkowitych, dziesiętnych i zmiennoprzecinkowych jak również dla typów date i timestamp które są reprezentowane jako znaczniki czasu POSIX (sekundy od 1 stycznia 1970),
- date -term Prologa w formie date(Year,Month,Day) używany domyślnie dla SQLowego typu date,
- timestamp -term Prologa w formie (Hour,Minute,Second) używany domyślnie dla SQLowego typu time.

Do zarządzania połączeniem służą nastepujące predykaty:

Otwórz połączenie:

```
odbc_connect(+DSN, -Polaczenie, [+Opcje])
```

Predykat tworzy nowe połączenie ODBC dla DSN (Data Source Name), a następnie zwraca uchwyt do tego połączenia do zmiennej Polaczenie. Zamiast tego można użyć aliasu w Opcjach.
Opcje umieszczamy w nawiasach kwadratowych. Dostępne opcje:

- user -nazwa użytkownika dla połączenia,
- password -hasło dla połączenia,
- alias -alias jako identyfikator połączenia,
- open -jeśli open (tryb otwarcia) równy jest „once" ponowne wywołanie DSN zwraca istniejące połączenie. Jeśli „multiple" następne połączenie do źródła danych jest otwierane.

Przykład:

```
 openDatabase :-
odbc_connect('postgresqlProlog',
_,[ user(postgres),password(pass),
alias(postgresqlProlog),open(once)]).
```

Wywołanie w prologu:

```
 openDatabase.
```

Zamknij połączenie:

```
odbc_disconnect(+alias).
```

Predykat zamyka podane połączenie. Usuwa alias połączenia lub jeśli nie ma aliasu blokuje użycie uchwytu dla tego połączenia.

Przykład:

```
closeDatabase:-
odbc_disconnect(postgresqlProlog).
```

Zwróć bieżące połączenia:

```
odbc_current_connection(?Connection, ?DSN)
```

Predykat zwraca wszystkie aktywne połączenia ODBC.
Wynik operacji zapisuje do argumentow Connection i DSN.

Przykład:

```
getNameConnection:-
odbc_current_connection(Alias,Dsn),
write('Alias'-Alias),nl,
write('Dsn'-Dsn).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[1] 7 ?- getNameConnection.
Alias-postgresqlProlog
Dsn-postgresqlProlog
true.
[1] 8 ?-

Zwróć właściwości bieżącego połączenia:

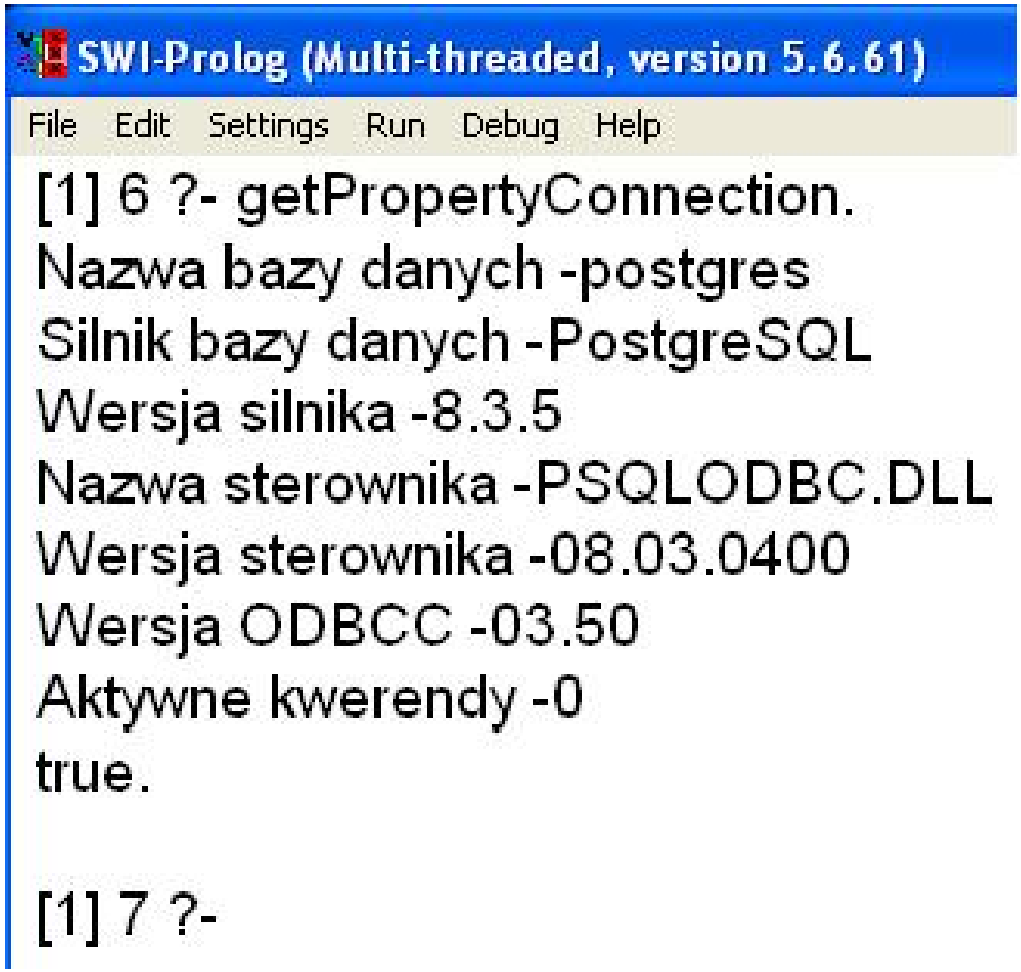
```
odbc_get_connection(+Connection, ?Property)
```

Kwerenda zwracająca właściwości połączenia takie jak np. nazwa bazy danych, nazwa silnika bazy danych, wersja silnika bazy danych, nazwa sterownika, wersja sterownika, wersja ODBC, aktywne kwerendy. Wynik operacji zapisywany jest do argumentu Property, argument Connection jest to zdefiniowany alias.

Przykład:

```
 getPropertyConnection:-
odbc_get_connection('postgresqlProlog',database_name(Zmienna1)),
odbc_get_connection('postgresqlProlog',dbms_name(Zmienna2)),
odbc_get_connection('postgresqlProlog',dbms_version(Zmienna3)),
odbc_get_connection('postgresqlProlog',driver_name(Zmienna4)),
odbc_get_connection('postgresqlProlog',driver_version(Zmienna5)),
odbc_get_connection('postgresqlProlog',driver_odbc_version(Zmienna6)),
odbc_get_connection('postgresqlProlog',active_statements(Zmienna7)),
write('Nazwa bazy danych '-Zmienna1),nl,
write('Silnik bazy danych '-Zmienna2),nl,
write('Wersja silnika '-Zmienna3),nl,
write('Nazwa sterownika '-Zmienna4),nl,
write('Wersja sterownika '-Zmienna5),nl,
write('Wersja ODBCC '-Zmienna6),nl,
write('Aktywne kwerendy '-Zmienna7),nl.
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[1] 6 ?- getPropertyConnection.
Nazwa bazy danych -postgres
Silnik bazy danych -PostgreSQL
Wersja silnika -8.3.5
Nazwa sterownika -PSQLODBC.DLL
Wersja sterownika -08.03.0400
Wersja ODBCC -03.50
Aktywne kwerendy -0
true.
[1] 7 ?-

Zwróć spis dostępnych źródeł danych:

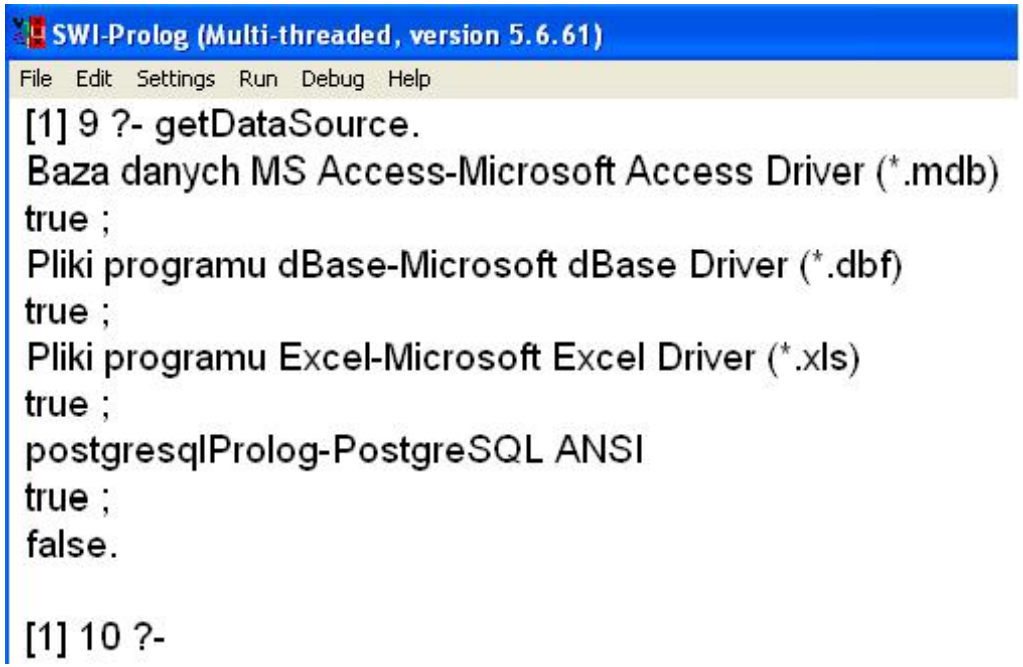
```
odbc_data_source(?DSN, ?Description)
```

Predykat zwraca wszystkie dostępne źródła danych zdefiniowane w systemie. Wynik operacji zapisywany jest do argumentów:
DSN-nazwa źródła danych, Description-opis.

Przykład:

```
getDataSource:-
odbc_data_source(DSN,Description),
write(DSN+Description).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[1] 9 ?- getDataSource.
Baza danych MS Access-Microsoft Access Driver (*.mdb)
true ;
Pliki programu dBase-Microsoft dBase Driver (*.dbf)
true ;
Pliki programu Excel-Microsoft Excel Driver (*.xls)
true ;
postgresqlProlog-PostgreSQL ANSI
true ;
false.
[1] 10 ?-

Ustaw właściwości bieżącego połączenia:

```
odbc_set_connection(+Connection, +Option)
```

Predykat ustawia właściwości połączenia Connection zdefiniowane w argumecie Option.

Opcje:
access_mode(Tryb) -jeśli 'read' dostajemy dostęp do bazy danych w trybie tylko do odczytu. Jeśli 'update' możemy aktualizować bazę danych,
auto_commit(bool) -jeśli prawda (domyślnie) wszystkie kwerendy do bazy są wykonywane natychmiast. Jeśli fałsz żądanie uaktualnienia rozpoczyna transakcję, która może być zatwierdzona lub wycofana.

Zwróć liczbę stworzonych i uwolnionych zapytań:

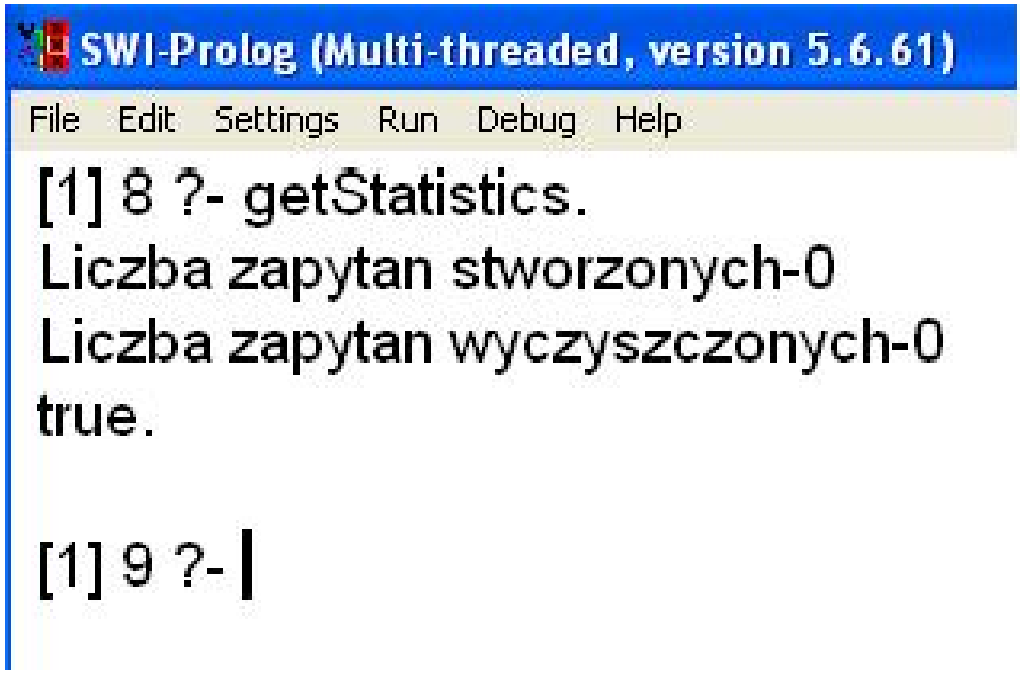
```
odbc_statistics(?Key)
```

Predykat zwraca liczbę stworzonych i zwolnionych zapytań. Wynik operacji zapisywany jest do argumentów: Created-zapytania stworzone, Freed-zapytania zwolnione.

Przykład:

```
getStatistics:-
odbc_statistics(statements(Created,Freed)),
write('Liczba zapytan stworzonych'-Created),nl,
write('Liczba zapytan wyczyszczonych'-Freed).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[1] 8 ?- getStatistics.
Liczba zapytan stworzonych-0
Liczba zapytan wyczyszczonych-0
true.
[1] 9 ?-

Dostęp do słownika bazy danych

Interfejs dostępu do słownika bazy danych w Prologu nie nadaje się do pełnej obsługi bazy danych, dostarcza on tylko podstawowy dostęp do struktury bazy, który można wykorzystać np. do sprawdzenia czy baza spełnia założenia aplikacji.
Interfejs definuje dwa podstawowe predykaty:
odbc_current_table(+Connection, -Table),
odbc_table_column(+Connection, ?Table, ?Column).

Zwróć listę wszystkich dostępnych tabel w bazie danych:

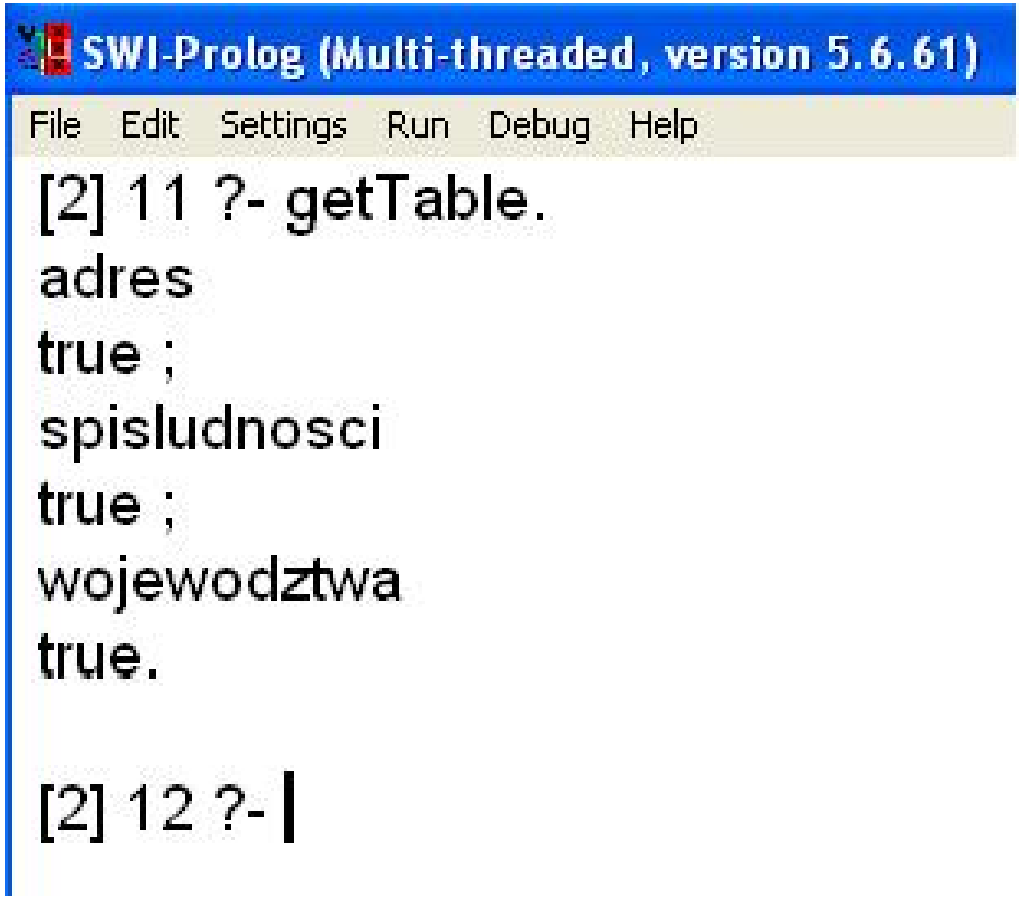
```
odbc_current_table(+Connection, -Table)
```

Predykat zwraca spis dostępnych tabel. Do argumentu Connection przekazuje się uchwyt połączenia bądź alias. Wynik operacji zapisywany jest do argumentu Table.

Przykład:

```
getTable:-
odbc_current_table(postgresqlProlog,Table),
write(Table).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[2] 11 ?- getTable.
adres
true ;
spisludnosci
true ;
wojewodztwa
true.
[2] 12 ?-

Zwróć liste wszystkich kolumn z tabel w bazie danych:

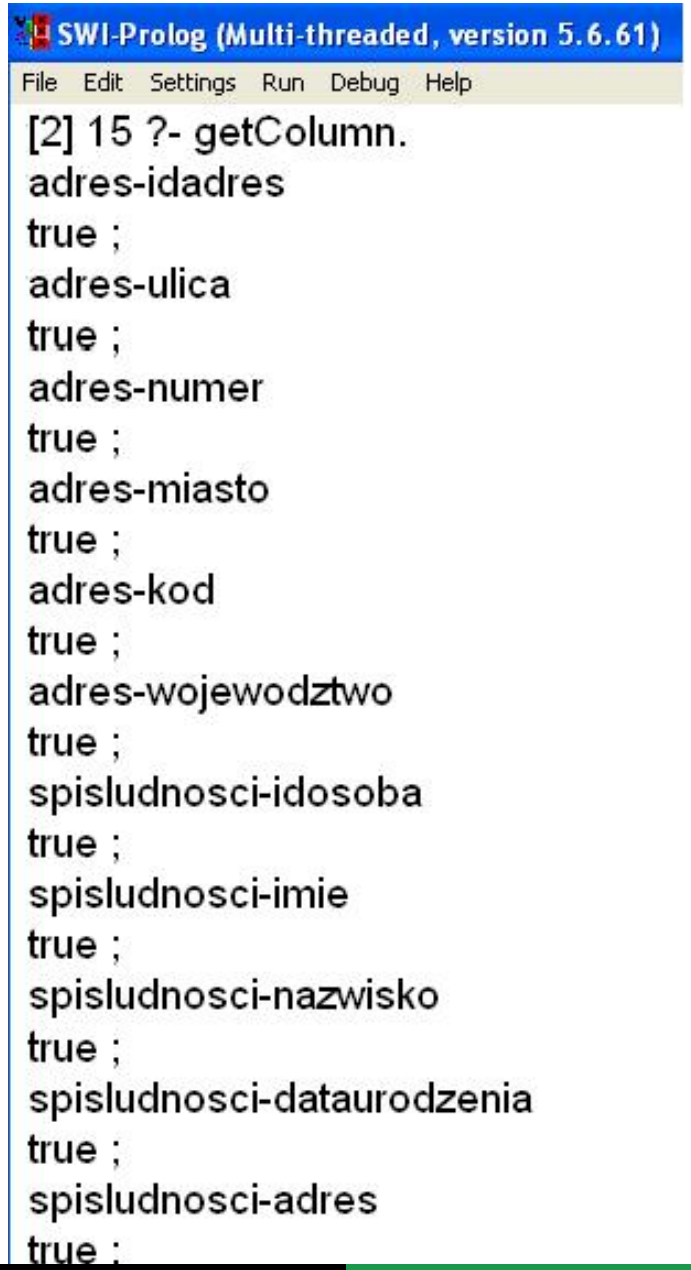
```
odbc_table_column(+Connection, ?Table, ?Column)
```

Predykat zwraca spis dostępnych kolumn w tabelach. Do argumentu Connection przekazuje się uchwyt połączenia bądź alias. Wynik operacji zapisywany jest do argumentów Table oraz Column.

Przykład:

```
getColumn:-
odbc_table_column(postgresqlProlog,Table,Column),
write(Table-Column).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[2] 15 ?- getColumn.
adres-idadres
true ;
adres-ulica
true ;
adres-numer
true ;
adres-miasto
true ;
adres-kod
true ;
adres-wojewodztwo
true ;
spisludnosci-idosoba
true ;
spisludnosci-imie
true ;
spisludnosci-nazwisko
true ;
spisludnosci-dataurodzenia
true ;
spisludnosci-adres
true ;

Jednorazowe zapytania

Interfejs ODBC w prologu definiuje jeden predykat służący do zapytań jednorazowych.

```
odbc_query(+Connection, +SQL, -RowOrAffected)
```

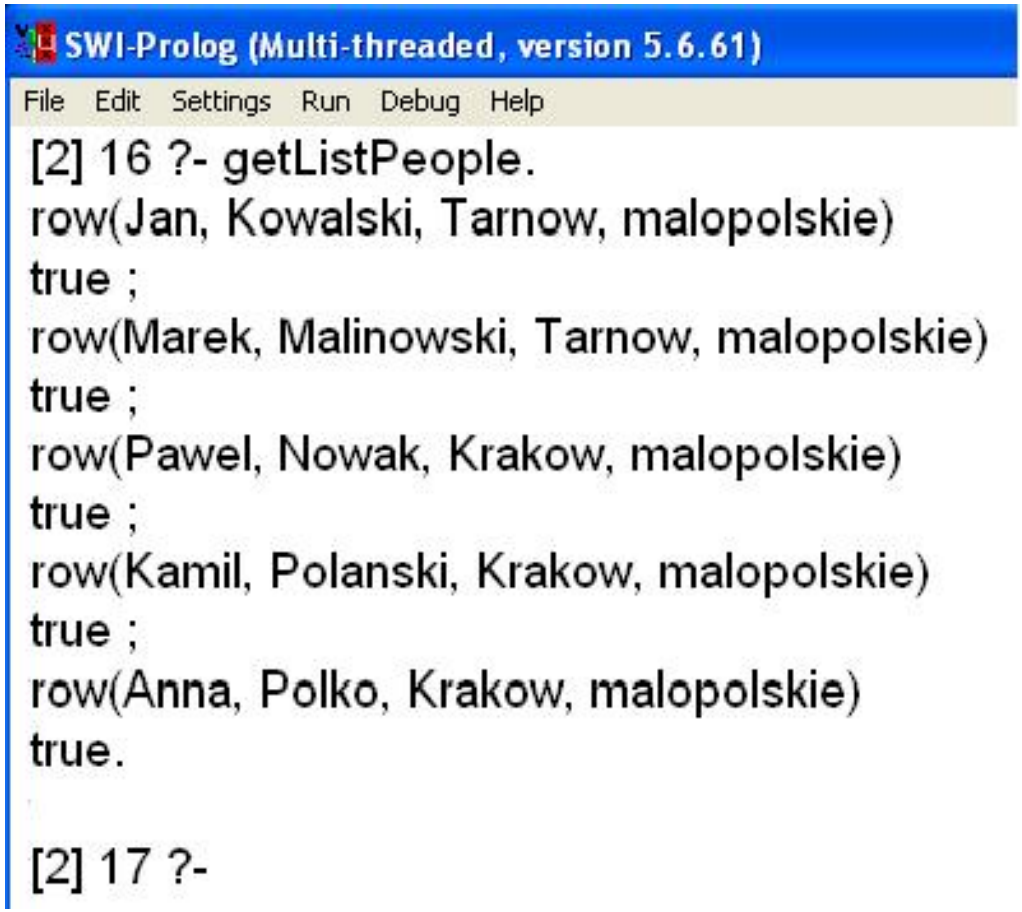
Predykat wykonuje kwerendę SQL na bazie danych reprezentowanej przez Connection. SQL to dowolne prawidłowe zapytanie. Jeśli zapytaniem jest SELECT wynikowe wiersze są zwracane do argumentu Row. Dla innych zapytań argument ten zwraca Affected gdzie reprezentuje liczbę wierszy zmienionych przez zapytanie.

Przykłady:

```
 getListPeople:-
odbc_query(postgresqlProlog,'SELECT
s.imie,s.nazwisko,a.miasto,a.wojewodztwo
FROM spisLudnosci s,adres a WHERE
s.adres=a.idAdres',Row,[]),

write(Row).
```

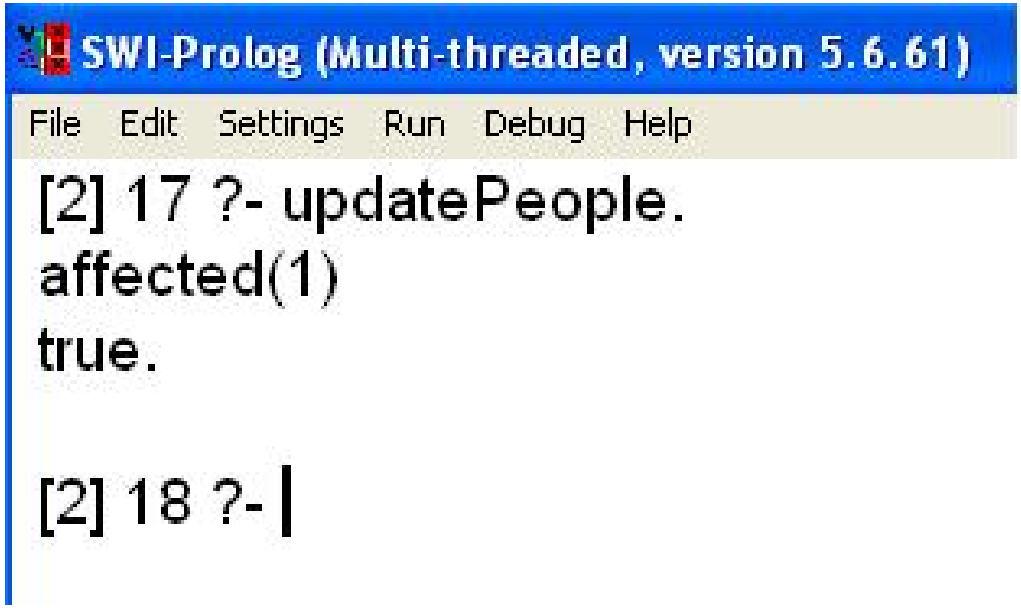
Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[2] 16 ?- getListPeople.
row(Jan, Kowalski, Tarnow, malopolskie)
true ;
row(Marek, Malinowski, Tarnow, malopolskie)
true ;
row(Pawel, Nowak, Krakow, malopolskie)
true ;
row(Kamil, Polanski, Krakow, malopolskie)
true ;
row(Anna, Polko, Krakow, malopolskie)
true.
[2] 17 ?-

Przykłady:

```
 updatePeople:-
odbc_query(postgresqlProlog,'UPDATE adres set
ulica='Mickiewicza' WHERE idAdres=1',Affected,[]),
write(Affected).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[2] 17 ?- updatePeople.
affected(1)
true.
[2] 18 ?-

Przykłady:

```
 deletePeople:-
odbc_query(postgresqlProlog,'DELETE FROM adres WHERE
idAdres=2',Affected,[]),

write(Affected).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
[2] 18 ?- deletePeople.
affected(1)
true.
[2] 19 ?-

Sparametryzowane zapytania

Interfejs ODBC w prologu definiuje trzy predykaty służące do obsługi zapytań sparametryzowanych.

```
odbc_prepare(+Connection,+SQL,+Parameters,-Statement)
odbc_execute(+Statement,+ParameterValues,-RowOrAffected)
odbc_free_statement(+Statement)
```

Predykat wykonuje przygotowanie zapytania sparametryzowanego z podanego zapytania SQL, które ma jeden lub więcej znaków '?', do argumentu Statement zapisywany jest stworzony obiekt, argument Parameters przedstawia typy danych zmiennych reprezentowanych poprzez znak '?', argument Options przedstawia typy danych zwracanych przez zapytanie. Do argumentu Connection przekazuje się uchwyt połączenia bądź alias.

```
 odbc_prepare(+Connection, +SQL, +Parameters,
-Statement, +Options)
```

Przykład:

```
 prepareStmtSelect:-
odbc_prepare(postgresqlProlog,'SELECT
s.imie,s.nazwisko,a.miasto,a.wojewodztwo FROM
spisLudnosci s,adres a WHERE s.adres=a.idAdres AND
s.idOsoba=?',[integer],StmtSelect,[types([string,
string, string, string])]).
```

Predykat wykonuje zapytanie przygotowane przez odbc_prepare. Argument Statement jest to uchwyt do obiektu przechowującego zapytanie, do argumentu ParameterValues przekazuje się wartości które zostaną wstawione na miejsce „?', agrument RowOrAffected zwraca wiersze lub ich ilość w zależności od typu zapytania. Predykat ten może zwrócić wyjątki type_error jeśli podane wartości parametrów nie mogą być skonwertowane do zadeklarowanych typów.

```
 odbc_execute(+Statement, +ParameterValues,
-RowOrAffected)
```

Przykład:

```
executeStmtSelect(Number):-
odbc_handle(StmtSelect),
odbc_execute(StmtSelect,[Number],Row),
write(Row).
```

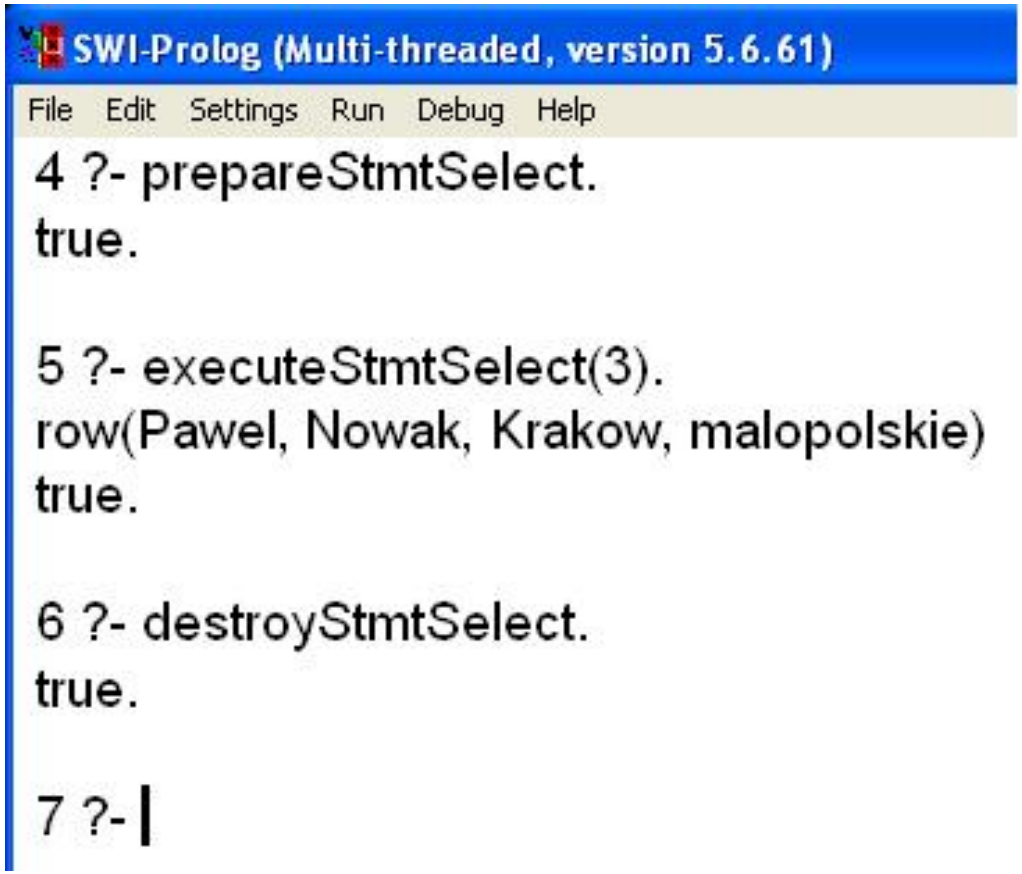
Predykat zwalnia przygotowany zasób, czyli utworzony obiekt Statement. Argument Statement - uchwyt do obiektu.

```
odbc_free_statement(+Statement)
```

Przykład:

```
 destroyStmtSelect:-
odbc_handle(StmtSelect),
odbc_free_statement(StmtSelect).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
4 ?- prepareStmtSelect.
true.
5 ?- executeStmtSelect(3).
row(Pawel, Nowak, Krakow, malopolskie)
true.
6 ?- destroyStmtSelect.
true.
7 ?-

Transakcje SQL

Do obsługi transakcji w prologu wykorzystujemy następujące predykaty:

```
 odbc_set_connection(+Alias,+Option),
odbc_end_transaction(+Alias,+Option).
```

```
odbc_set_connection(+Connection,+Option).
```

Predykat ustawia właściwość połączenia w tryb obsługi transakcji bądź bez obsługi transakcji w zależności od przekazywanego argumentu. Argument Option - auto_commit(bool), false-włącza obsługe transakcji. Argument Connection - jest to uchwyt do połączenia.

Przykład:

```
odbc_set_connection(postgresqlProlog,auto_commit(false)).
```

```
odbc_end_transaction(+Connection,+Option).
```

Predykat ten zatwierdza wszystko to co transakcja obejmuje jeżeli Option jest ustawiony na 'commit', porzuca wszystko jeżeli Option jest ustawiony na 'roolback'. Argument Connection - jest to uchwyt do połączenia.

Przykład:

```
odbc_end_transaction(postgresqlProlog,'commit')
```

Przykład:

```
 testTransaction(Option):-
odbc_set_connection(postgresqlProlog,auto_commit(false)),
prepareStmtInsert1,
executeStmtInsert1(6,'Westerplatte',19,'Opole',
'33-400','opolskie'),
destroyStmtInsert1,
prepareStmtInsert2,
executeStmtInsert2('Janusz','Partyka','1989-01-09',6),
destroyStmtInsert2,
odbc_end_transaction(postgresqlProlog,Option).
```

Rezultat:


SWI-Prolog (Multi-threaded, version 5.6.61)
File Edit Settings Run Debug Help
16 ?- testTransaction('commit').
true.
17 ?-

Źródła:

http://www.swi-prolog.org/
http://pl.wikipedia.org/wiki/ODBC